

CICS Transaction Server for z/OS



CICS アプリケーションの開発

バージョン 5 リリース 5

CICS Transaction Server for z/OS



CICS アプリケーションの開発

バージョン 5 リリース 5

注記

本書および本書で紹介する製品をご使用になる前に、 1101 ページの『特記事項』に記載されている情報をお読みください。

本書は、IBM CICS Transaction Server for z/OS バージョン 5 リリース 5 (製品番号 5655-Y04) 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： CICS Transaction Server for z/OS
Developing CICS Applications
Version 5 Release 5

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

© Copyright IBM Corporation 1989, 2018.

目次

この PDF について	ix
-----------------------	----

第 1 章 CICS アプリケーションとは . . . 1

CICS プログラム、トランザクション、およびタスク	1
CICS プログラミング	2
CICS データ・オブジェクト	3
CICS プログラミング・コマンド	4
EXEC インターフェース・ブロック (EIB)	5
プログラム変換	5
CICS のテスト	5
CICS プログラミング・ロードマップ	6

第 2 章 非同期要求の実行 7

非同期 API を使用する利点	9
非同期 API について	9
非同期 API でのパフォーマンスの管理	11
EXEC CICS 非同期 API コマンド	12
非同期 API での JCICS の使用	12

第 3 章 クラウド対応 CICS Transaction Server for z/OS 15

仕組み: プラットフォーム	19
zFS 上のプラットフォーム・ディレクトリー構造	28
プラットフォームの状態	31
仕組み: アプリケーション	32
アプリケーション・バインディング	37
アプリケーションのエントリー・ポイント	39
アプリケーション・コンテキスト	44
アプリケーションの状態	50
仕組み: 複数バージョン管理のアプリケーション	54
アプリケーションへのバージョンの割り当て	66

第 4 章 アプリケーションの設計 69

タスクの開始方法	69
どのトランザクションか?	70
ビジネス・ロジックと表示ロジックの分離	74
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	75
準再入可能なアプリケーション・プログラム	77
スレッド・セーフ・プログラム	78
アプリケーションのスレッド・セーフ化	81
CONCURRENCY(REQUIRED) プログラム	86
OPENAPI プログラム	87
FORCEQR システム初期設定パラメーターの使用	91
再入不能プログラム	91
トランザクション内のデータの格納	92
トランザクション作業域 (TWA)	92
ユーザー・ストレージ	93

LINK コマンドおよび XCTL コマンドにおける COMMAREA	93
LINK および XCTL コマンドのチャネル	94
プログラム・ストレージ	95
一時記憶域キュー	95
区画内一時データ	97
GETMAIN SHARED コマンド	98
ユーザー独自のデータ・セット	98
CICS コマンドに渡される区域の長さ	99
エラーの最小化	100
アプリケーション・エラーからの CICS の保護	100
アプリケーションのテスト	101
端末を持たないトランザクションのセキュリティ	102
パフォーマンスの設計	103
プログラム・サイズ	103
仮想記憶域	104
リソースの排他制御	108
操作のコントロール	109
オペレーティング・システム待機	110
NOSUSPEND オプション	110
効果的な順次データ・セットのアクセス	112
効率的なロギング	113
非同期要求のための設計	113
トランザクション間のデータの共用	114
共通作業域 (CWA) の使用	114
TCTTE ユーザー域 (TCTUA) の使用	118
RETURN コマンドでの COMMAREA の使用	118
RETURN コマンドにおけるチャネルの使用	119
データを共用する表示画面の使用	119
チャネルによるプログラム間データ転送	120
チャネル: クイック・スタート	121
チャネルの使用: いくつかの典型的なシナリオ	125
チャネルの利点	128
チャネルの作成	130
現行チャネル	131
チャネルの有効範囲	136
プログラムに渡されたコンテナの検出	141
リンクから返されたコンテナの検出	141
チャネルとコンテナの削除とそのストレージの解放	142
チャネルの設計: 最良事例	144
チャネルの構成および使用: 例	146
CICS の読み取り専用コンテナ	147
JCICS からのチャネルの使用	147
チャネルと BTS アクティビティ	148
チャネルを使用した動的ルーティング	150
データ変換	150
COMMAREA からチャネルへのマイグレーション	158
プログラム制御	162
プログラムのリンク	163

他のプログラムへのデータの受け渡し	164
混合アドレッシング・モードの使用	169
LINK を使用したデータの受け渡し	170
RETURN を使用したデータの受け渡し	172
類縁性	175
類縁性のタイプ	176
プログラミング手法と類縁性	177
類縁性を防止するプログラミング手法	179
類縁性を生じさせるプログラミング手法	184
類縁性を生じさせる可能性のあるプログラミング 手法	191
トランザクション間の類縁性の期間と有効範囲	203
リカバリーの設計	212
ジャーナル処理	212
同期点処理	215
例外条件の取り扱い	217
デフォルト CICS 例外処理	217
インライン・コードによる例外条件の処理	218
デフォルト CICS 例外処理の変更	221
HANDLE CONDITION コマンドの使用	223
HANDLE CONDITION ERROR コマンドの使 用	226
IGNORE CONDITION コマンドの使用	227
HANDLE ABEND コマンドの使用	228
PUSH HANDLE および POP HANDLE コマ ンドの使用	229
SOAP 障害メッセージの解析	230
ストーム・ドレイン作用の回避	234
異常終了のリカバリー	236
プログラム・レベルの異常終了プログラムまたは ルーチンの作成	238
操作の再試行	239
トレース	240
アプリケーション・パフォーマンスのモニター ダンプ	242
QUERY SECURITY コマンドを使用した、リソースへ のユーザーのアクセスの判別	243
ユーザー定義リソースを使用するためのアプリケー ションの設計	245
CICS の相互通信	245
トランザクション・ルーティング	247
機能シップ	247
分散プログラム・リンク (DPL)	248
非同期処理	262
分散トランザクション処理 (DTP)	262
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)	262
外部 CICS インターフェース (EXCI)	263
アプリケーション・プログラム用の CICS サービス ファイル制御についての理解	264
端末管理	312
インターバル制御	352
タスク制御	355
CICS ストレージ保護およびトランザクション分 離	358
一時データ管理	375

一時記憶管理	379
CICS の文書および文書テンプレート	380
文書テンプレートの設定	387
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	396
名前付きカウンター・サーバー	409
印刷とスプール・ファイル	431
基本マッピング・サポート	459

第 5 章 非同期要求のための開発 571

第 6 章 アセンブラー言語アプリケーションの開発 575

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	575
EXEC CICS アセンブラー・インターフェースのコ ーディング	580
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	584
アセンブラー言語プログラムの呼び出し	587
動的ストレージの拡張	590
AMODE(64) アセンブラー言語プログラムの開発	591

第 7 章 C および C++ アプリケーションの開発 593

C および C++ プログラムの制約事項および要 件	594
C および C++ での引数の受け渡し	599
C および C++ から EIB へのアクセス	600
C および C++ の地域サポート	601
XPLink と C および C++ プログラム	602
XPLink による X8 および X9 モード TCB の 使用	603
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	603
グローバル・ユーザー出口と XPLink	603
CICS ファウンデーション・クラスの使用	604
C++ オブジェクト	604
ファウンデーション・クラスの概要	606
バッファ・オブジェクト	613
CICS Service の使用	616
コンパイル、実行、およびデバッグ	636
条件、エラー、および例外	639
ポリモフィック動作	647
ストレージ管理	650
パラメーター受け渡し規則	651
「read」メソッドから返される IccBuf 参照内の データの有効範囲	652

第 8 章 COBOL アプリケーションの開発 653

COBOL プログラムの制約事項および要件	654
Language Environment の CBLPSHPOP オプ ション	658
DL/I CALL インターフェースの使用	658

VS COBOL II プログラム	660
COBOL での基底付きアドレッシングの使用	661
COBOL プログラムからのサブプログラムの呼び出し	662
プログラムとサブプログラム間の制御のフロー	663
サブプログラムの呼び出し規則	665
COBOL2 および COBOL3 変換プログラム・オプション	668
COBOL プログラムの CICS 変換プログラム・アクション	669
COBOL プログラムのバッチ・コンパイル	672
ネストされた COBOL プログラム	674

第 9 章 プログラミング言語と言語環境

プログラム (Language Environment) . 679

言語環境プログラムの呼び出し可能サービス	680
言語環境プログラム (Language Environment) の異常終了と条件処理	681
言語環境プログラムのストレージ	683
言語環境プログラムにおける言語の混合	684
ダイナミック・リンク・ライブラリー (DLL)	686
言語環境プログラム (Language Environment) のランタイム・オプションの定義	687
CEEEXIT および CEECSTX ユーザー出口	689
CICSVAR: CICS 環境変数	690
言語環境プログラムの CEEBINT 出口	692

第 10 章 PL/I アプリケーションの開発 693

PL/I プログラミングの制約事項と要件	693
PL/I アプリケーションでの言語環境プログラム (Language Environment) のコーディング要件	695
取り出した PL/I ルーチン	697

第 11 章 Java アプリケーションの開発 699

CICS について必要な知識	699
CICS トランザクション	700
CICS タスク	700
CICS アプリケーション・プログラム	700
CICS サービス	701
CICS での Java ランタイム環境	703
IBM CICS SDK for Java を使用したアプリケーションの開発	704
ターゲット・プラットフォームの更新	705
プラグイン・プロジェクトの作成	706
プラグイン・プロジェクトのマニフェスト・ファイルの更新	708
Java EE アプリケーションの作成	709
CICS バンドル・プロジェクトへのプロジェクトの追加	711
プロジェクト・ビルド・パスの更新	712
その他のツールを使用したアプリケーションの開発	713
共用 JVM に関する考慮事項	714
JCICS を使用した Java の開発	715
CICS 用 Java クラス・ライブラリー (JCICS)	715
データ・エンコード	719
JCICS API サービスと例	720

JCICS の使用	746
Java の制約事項	747
OSGi の使用に関するガイダンス	748
Liberty JVM サーバーで実行する Java アプリケーションの開発	751
開発環境のセットアップ	751
Java EE アプリケーションと Liberty アプリケーション	752
Liberty JVM サーバー内で実行するよう Java EE アプリケーションをマイグレーションする	758
CICS プログラムからの Java EE アプリケーションへのリンク	759
Java Transaction API (JTA)	766
Java Persistence API (JPA)	767
Enterprise JavaBeans (EJB)	770
Java Message Service (JMS)	778
Java Management Extensions API (JMX)	780
Java Authorization Contract for Containers (JACC)	782
Java Authentication Service Provider Interface for Containers (JASPIC)	783
Java EE コネクター・アーキテクチャー (JCA)	785
MicroProfile によるマイクロサービスの開発	801
Liberty Web サーバー・プラグイン	806
Liberty フィーチャー	806
Java アプリケーションからのデータへのアクセス	820
Java からの構造化データとの対話	821
OSGi JVM サーバーで JZOS Toolkit API を使用する Java アプリケーションの開発	822
Java プログラムから IBM MQ へのアクセス	825
CICS Liberty JVM サーバーでの IBM MQ classes for JMS の使用	827
OSGi JVM サーバーでの IBM MQ classes for JMS の使用	832
OSGi JVM サーバーでの IBM MQ classes for Java の使用	837
CICS 内の Java アプリケーションからの接続	838
JCA ローカル ECI サポート	839
JVM サーバーで実行する既存のアプリケーションのパッケージ化	839
既存の Java プロジェクトのプラグイン・プロジェクトへの変換	839
JAR ファイルの内容の OSGi プラグイン・プロジェクトへのインポート	841
バイナリー JAR ファイルの OSGi プラグイン・プロジェクトへのインポート	844

第 12 章 リカバリーのための開発 . . . 847

アプリケーション設計上の考慮事項	847
リカバリー要件に関連する質問	847
リカバリー要件ステートメントの検証	849
ユーザーの再開手順の設計	849
ユーザーの待機手順	850
アプリケーションとユーザーの間の通信	850
セキュリティ	850
リカバリー関連機能のシステム定義	851

資料およびテスト計画	852
リカバリーのためのプログラミング	853
リカバリーのためのアプリケーションの設計	853
プログラム設計	856
トランザクション障害およびシステム障害の管理	863
アプリケーション・プログラムにおけるリソース のロック (エンキュー)	868
トランザクション・バックアウトのユーザー出口	876

第 13 章 変換およびコンパイル 881

統合 CICS 変換プログラム	881
統合 CICS 変換プログラムの使用	882
CICS 変換プログラムのオプションの指定	883
変換のプロセス	884
CICS 提供の変換プログラム	887
個別の変換プログラムの動的な起動	887
CICS 変換プログラムの使用	899
変換プログラムのオプションの定義	901
変換プログラムのオプション・テーブル	901
COPY ステートメントの使用	903
CICS 提供のインターフェース・モジュール	903
AMODE(24) および AMODE(31) アプリケーション での EXEC インターフェース・モジュールの使 用	904
LEASM を使用するアセンブラ言語プログラム のサンプル	908
AMODE(64) アプリケーションでの EXEC インタ ーフェース・モジュールの使用	915
大文字変換	916
国別文字の大文字への変換	916
データ共用メッセージの大文字への変換	918

第 14 章 アプリケーションのセットア ップ 919

プラットフォームへのデプロイメントのための	
CICS アプリケーションの設計	920
クラウド環境でのデプロイメントのための CICS ア プリケーションのパッケージ化	921
複数バージョンに対応するアプリケーションの呼び 出し	923
EXEC CICS INVOKE APPLICATION の例	925

第 15 章 アプリケーションのデバッグ 927

実行診断機能 (EDF)	927
EDF を使用する場合の制約事項	929
EDF 表示画面での実行内容	931
EDF を使用したプログラムのテスト	939
EDF による情報の変更	946
EDF メニュー機能の使用	948
一時記憶域のブラウズ (CEBR)	956
CEBR トランザクションの使用	956
CEBR トランザクションの表示内容	958
CEBR ファンクション・キーの使用	959
CEBR コマンドの使用	960
一時データでの CEBR トランザクションの使用	963
コマンド・レベル・インタープリター (CECI)	964

CECI による表示内容	964
CECI の使用	970
変数の定義	971
コマンドの保管	973
CECI の実行方法	974
共用ストレージ: LENGTH オプションを指定し ない ENQ コマンド	975
CICS アプリケーションでのデバッガーの使用準備	976
デバッグ・プロファイル	977
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	979
デバッグ・プロファイルでの汎用パラメーターの 使用	981
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	982
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	983
CICS アプリケーションでのデバッグ・ツールの使 用	984
デバッグ・ツールについて	984
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	985

第 16 章 アプリケーションのデプロイ 987

プラットフォームへのアプリケーションのデプロイ	987
アプリケーション・プログラムのインストール	989
プログラムのインストール・ステップ	990
動的プログラム LIBRARY リソースの使用	991
MVS 常駐モードおよびアドレッシング・モー ドの定義	1002
リンク・パック域にあるアプリケーションの実 行	1004
読み取り専用 DSA にあるアプリケーション・ プログラムの実行	1005
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	1010
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 提供プロシージャの使用	1011
CICS 提供インターフェース・モジュールの組 み込み	1013
アセンブラ言語アプリケーション・プログラ ムの変換、アセンブル、およびリンク・エディ ット	1014
COBOL アプリケーション・プログラムのイン ストール	1016
PL/I アプリケーション・プログラムのインス トール	1021
C アプリケーション・プログラムのインストー ル	1021
ユーザー独自のジョブ・ストリームの使用	1025
マップ・セットおよび区分セットのインストール	1029
マップ・セットのインストール	1031
区分セットのインストール	1040
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	1041
アプリケーションのテスト	1042

テストに対するアプリケーションの準備 . . .	1043	CICS Build Toolkit による CICS アプリケーショ	
テストに対するシステムの準備	1044	ン・ビルド自動化	1067
JVM サーバーへのアプリケーションのデプロイ	1044	CICS Build Toolkit の使用の準備	1068
JVM サーバーにおける OSGi バンドルのデプ		CICS バンドル、アプリケーション、アプリケ	
ロイ	1045	ーション・バインディング、またはプラットフ	
CICS バンドル内の Java EE アプリケーション		ォームのビルド	1069
の Liberty JVM サーバーへのデプロイ . . .	1047	CICS バンドル内の変数の解決	1071
Liberty JVM サーバーへの Java EE アプリケ		CICS Build Toolkit コマンド行オプション	1073
ーションの直接デプロイ	1049	CICS Build Toolkit の戻りコード	1076
Liberty JVM サーバーへの共通ライブラリーの		DFHDPLOY ユーティリティによる CICS パン	
デプロイ	1051	ドルおよびアプリケーションのデプロイメントお	
JVM サーバー内の Java アプリケーションの呼		よびアンデプロイメントの自動化	1077
び出し	1052	DFHDPLOY ユーティリティ・コマンド	1079
CICS の非 OSGi Java アプリケーションの配		DFHDPLOY ユーティリティ・スクリプトの	
置	1053	例	1091
CICS Db2 プログラムの実行および実動の準備	1054	DFHDPLOY ユーティリティの戻りコード	1094
CICS Db2 テスト環境	1054	日本語または中国語 (簡体字) で出力メッセー	
CICS Db2 プログラムの準備	1055	ジを受け取る	1095
プログラム変更後に何をバインドするか . . .	1059	UrbanCode Deploy によるデプロイメント . . .	1096
プログラムのバインド・オプションと考慮事項	1060	Node.js アプリケーションのデプロイ	1097
CICS Db2 プログラムのテストおよびデバッグ	1062	Node.js ランタイムのインストールの検査 . . .	1098
実動への移行: CICS Db2 アプリケーションの			
チェックリスト	1062	特記事項	1101
Db2 にアクセスする CICS アプリケーション		索引	1107
のチューニング	1065		

この PDF について

この PDF では、EXEC CICS® API を使用して CICS サービスとリソースにアクセスするアプリケーションを設計し、開発する方法について説明します。以下にリストする他の PDF では、CICS の特定の分野のアプリケーション・プログラミングに関する考慮事項について説明していますので、この PDF だけでなく、これらも参照してください (IBM® Knowledge Center では、これらすべての情報が「Developing Applications」という 1 つのセクションにまとめられています)。この PDF と併せて、API (EXEC CICS) リファレンスも参照してください。

CICS のさまざまな分野のプログラミング情報について、以下の PDF に記載しています。

- SOAP および JSON については、*Web サービス・ガイド*を参照してください。
- ONC/RPC インターフェースについては、*外部インターフェース・ガイド*を参照してください。
- EXCI については、CICS での EXCI の使用を参照してください。
- Java™ および Liberty については、CICS での *Java* アプリケーションを参照してください。
- フロントエンド・プログラミング・インターフェースについては、*フロントエンド・プログラミング・インターフェース・ユーザズ・ガイド*を参照してください。
- Db2® については、*Db2 Guide* を参照してください。
- DBCTL については、*IMS DB コントロール・ガイド*を参照してください。
- 共用データ・テーブルについては、*共用データ・テーブルの手引き*を参照してください。
- CICSplex® SM については、*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。
- BTS については、*ビジネス・トランザクション・サービス*を参照してください。
- CICS システム間の接続については、*相互通信ガイド*を参照してください。

CICS アプリケーションの問題の解決方法については、CICS のトラブルシューティングを参照してください。

本書で使用する用語と表記について詳しくは、IBM Knowledge Center の CICS 資料で使用されている表記規則および用語を参照してください。

この PDF の日付

この PDF は、2018 年 12 月 14 日に作成されました。

第 1 章 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、CICS 制御下で実行され、CICS サービスとインターフェースを使用してプログラムとファイルにアクセスします。

CICS は、トランザクション処理サブシステムです。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用する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様に行われます。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管理し、迅速に応答します。

CICS アプリケーションは、従来から、トランザクション 要求を実行依頼することで実行されます。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・プログラムをプログラムと呼ぶことがあります。また、アプリケーション・プログラムが実行する処理を指して、トランザクション という用語を使用しています。

現在、IT 業界では、トランザクション という用語は、リカバリー単位 (CICS という作業単位) を表すものとして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 トランザクションの範囲は単一の作業単位でもあります。CICS 以外の資料を使用するときには、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。

以下の用語は、CICS の資料全般で、また多くのコマンドで使用されます。

プログラム

CICS では、トランザクションの全体または一部分を処理するためにプログラムが使用されます。プログラムは、プログラム・ライブラリーの中に保管されます。プログラム・リソース、および CICS バンドルで使用されるプログラムについて、詳しくは PROGRAM リソースを参照してください。

トランザクション

トランザクションとは、単一の要求によって開始される処理の項目です。この要求は通常、端末のエンド・ユーザーから出されますが、Web ページ、リモート・ワークステーション・プログラム、または別の CICS システムのアプリケーションから出される場合や、事前定義された時刻に自動的にトリガーされる場合もあります。CICS トランザクションを実行するためのさまざまな方法については、インターネット、TCP/IP、および HTTP の概念およびCICS 外部インターフェースの概要を参照してください。

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 で使われる「トランザクション」という用語は、1 つのイベント、および同じタイプの他のすべてのトランザクションの両方を意味します。 TRANSACTION リソース定義を使用して CICS に対してそれぞれのトランザクション・タイプを記述します。この定義により、トランザクション・タイプの名前 (トランザクション ID、つまり TRANSID) が指定され、実行すべき作業についての情報が CICS に提供されます (例えば、最初にどのプログラムを呼び出すか、トランザクションの実行全般でどのような種類の認証が必要か)。

トランザクションの TRANSID を CICS に向けて送信することで、そのトランザクションを実行します。 CICS は TRANSACTION 定義に記録された情報を使っ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、IT 業界では、トランザクション という用語は、リカバリー単位 (CICS でいう作業単位) を表すものとして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 トランザクションの範囲は単一の作業単位でもありますが、CICS 以外の資料を使用するときには、意味の違いに注意する必要があります。

タスク

CICS および CICS 資料では、「タスク」という用語に特殊な意味があります。 CICS は、トランザクションの実行要求を受け取ると、トランザクション・タイプの実行におけるこの 1 つのインスタンスに関連付けられた新しいタスクを開始します。つまりタスクとは、(通常、特定の端末を使用する特定のユーザーのために) 特定のデータ・セットを使ってトランザクションが 1 回実行されることです。タスクは、スレッドのようなものとも考えることもできます。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 プログラムは、他のプログラムと同じような方法で作成します。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言語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 アプリケーション開発の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、JCICS を使用した Java の開発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 ファウンデーション・クラスの使用法を参照してください。

- HTTP の要求と応答を処理する Web アプリケーションの作成については、HTTP アプリケーションの開発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、プログラミング言語と言語環境を参照してください。

プログラムでは、CICS コマンドだけでなく、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) コマンドを使用することができます。関連する詳細情報については、以下を参照してください。

- SQL を使用するには Db2 for z/OS 製品資料内の『SQL: Db2 の言語』および Db2 for z/OS 製品資料内の『Db2 for z/OS のプログラミング』を参照してください。
- DL/I を使用するには IMS 製品資料内の『EXEC DLI によるアプリケーション・プログラミング』および IMS 製品資料内の『アプリケーション・プログラミング設計』を参照してください。
- CPI を使用するには、z/VM アプリケーション・プログラミングを参照してください。
- FEPI を使用するには、FEPI API を使用した開発を参照してください。
- IBM MQ を使用するには、IBM MQ 製品資料を参照してください。

CICS データ・オブジェクト

CICS には、CICS プログラムとトランザクションが相互にデータを交換するために使用できるさまざまなオブジェクトが用意されています。

CICS には、以下のデータ・オブジェクトが用意されています。

通信域

通信域 (COMMAREA) とはストレージの単一ブロックであり、プログラム間のデータの受け渡しに使用されます。プログラムは、別のプログラムに COMMAREA を 1 つのみ渡すことができます。通信域の推奨される最大サイズは 24 KB です。ただし、絶対的な最大サイズは 32 KB です。

コンテナおよびチャネル

コンテナとは名前付きのデータ・ブロックであり、プログラム間で情報を受け渡すために使用されます。プログラムは、チャネルと呼ばれるコンテナの論理グループを使用して、別のプログラムに任意の数のコンテナを渡すことができます。コンテナは、32 KB という最大サイズには制限されません。

ビジネス・トランザクション・サービス (BTS) プロセスに関連付けられたコンテナはリカバリー可能 です。トランザクション開始時の状態にロールバックすることができます。

一時記憶域キュー

一時記憶域キューはデータ項目の名前付きキューであり、任意の順序で書き込み、再書き込み、読み取り、および再読み取りができます。データ項目の推奨される最大サイズは 24 KB です。ただし、絶対的な最大サイズは 32 KB です。

一時記憶域キューは、リカバリー可能として定義できます。

一時データ・キュー

一時データ・キューはデータ項目の名前付きキューであり、一度限り、書き込みと読み取りができます。データ項目の最大サイズは 32 KB です。

一時データ・キューは順番に読み取らなければならない、各項目は一度しか読み取れません。トランザクションが読み取った項目は、キューから削除され、他のトランザクションでは使用できなくなります。

CICS には、次の 2 つのタイプの一時データ・キューが用意されています。

- 区画内一時データ・キュー は、主として CICS プログラム間の通信に使用されます。
- 区画外一時データ・キュー は、主として CICS プログラムと順次デバイス (QSAM データ・セットやプリンターなど) の間の通信に使用されます。

区画内一時データ・キューは、リカバリー可能として定義できます。

CICS には、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保管するためのさまざまな機能も用意されています。詳しくは、トランザクションのデータの格納を参照してください。

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、それらのフィールドは他の方法で設定または使用するべきではありません。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可能性があるため、CICS コマンドで引数として使用しないでください。

INQUIRE コマンド、SET コマンド、および PERFORM コマンドをアプリケーション・プログラムで使用して、CICS リソースに関する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、およびジョブ入力サブシステム (JES) に対する CICS インターフェースのスパール・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。

CICS プログラミング・コマンドの参照情報については、CICS API コマンドを参照してください。

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 フィールドのプログラミング情報については、EIB フィールドを参照してください。

プログラム変換

古いコンパイラ (およびアセンブラ)の中には CICS コマンドを直接処理できないものもあります。プログラムを実行可能コードに変換するための追加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 (またはアセンブラ) が解釈できるようにします。

ほとんどのコンパイラでは統合 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・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します。統合 CICS 変換プログラム方式を使用すると、変換タスクの多くはコンパイル時に実行されるので、変換ステップを追加で実行しなくても済みます。変換ステップにおけるタスクについて詳しくは、を参照してください。

CICS のテスト

プログラムには、CICS 環境で実行されているかどうかを判別する 2 つの方法があります。C 言語の `iscics()` 関数を使用する方法と、DFH3QSS プログラムを呼び出す方法です。

このタスクについて

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

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。

環境 問題プログラム状態でこの機能呼び出す必要があります。

CICS プログラミング・ロードマップ

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、このロードマップのステップを実行してください。

手順

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、アプリケーションの設計および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 コマンド・サマリーを参照してください。
3. プログラムを変換してコンパイルする。統合 CICS 変換プログラムを取り込んだコンパイラを使用している場合は、アプリケーション・プログラムのインストール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、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 アプリケーション・プログラムのインストール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. プログラムおよび関連するトランザクションを CICS に対して定義する。PROGRAM リソースおよび TRANSACTION リソースを使用します。
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してインストールする。
6. プログラムを実行する。CICS 端末からトランザクション ID を入力するか、で説明されている方法のいずれかを使用します。

第 2 章 非同期要求の実行

CICS で提供される非同期 API コマンドを使用すれば、簡単な方法で外部サービスに要求を発行し、それらを非同期で実行できるようになります。非同期 API コマンドを使用すると、各サービスを順次呼び出し、応答を待機するのではなく、簡単かつ強力な方法で非同期アプリケーションを作成できるため、待機時間を節約し、プログラムを解放して他の処理を続行させることができます。

非同期処理の CICS への影響

CICS で非同期 API コマンドを使用すると、複数の子トランザクションを非同期で実行でき、アプリケーションの全体的な応答時間が削減されます。親は、非同期で実行でき、別の CICS タスクで実行される 1 つ以上のローカル子トランザクションを実行できます。CICS チャンネルおよびコンテナーを使用して、親から子にデータを渡したり、データを戻して受け取ったりできます。子トランザクションを実行およびフェッチするフレームワークはすべて、サポートされている非同期 API を使用して行われます。

順次トランザクション処理と非同期トランザクション処理の比較

順次トランザクション処理の場合、CICS は外部サービスを呼び出し、応答を待機します。その後、CICS によってその他の複数の外部サービスが呼び出され、それらからの応答を待機する場合があります。このシーケンスを、ユーザーが必要なすべてのデータを受け取るまで続けることができます。これは、待機時間が長くなる可能性があり、応答を受信するまでプログラムの実行はブロックされます。『第 2 章 非同期要求の実行』に、CICS がどのように外部サービスを呼び出し、応答を待機するのかを示します。

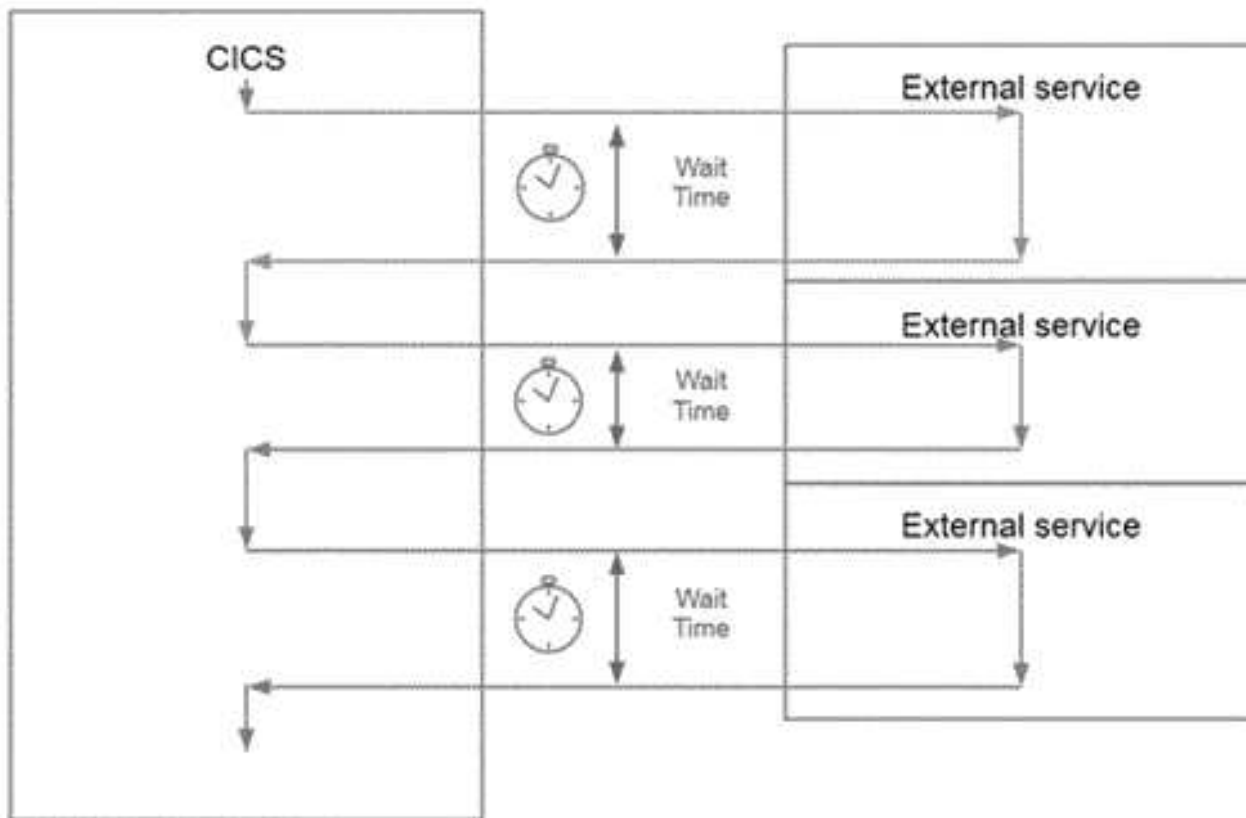


図 1. 順次サービス呼び出し

順次トランザクション処理とは異なり、非同期トランザクション処理では複数の要求を同時に発行できます。外部サービスを非同期で呼び出すことでアプリケーションの全体的な応答時間を削減できます。親は解放され、子のトランザクションとは別個にそれ自体のロジックを継続できます。図 2 は、CICS によって複数の外部サービスが同時に呼び出されている様子を示しています。待機時間が削減され、プログラムは応答を受信するまで処理を続行できます。

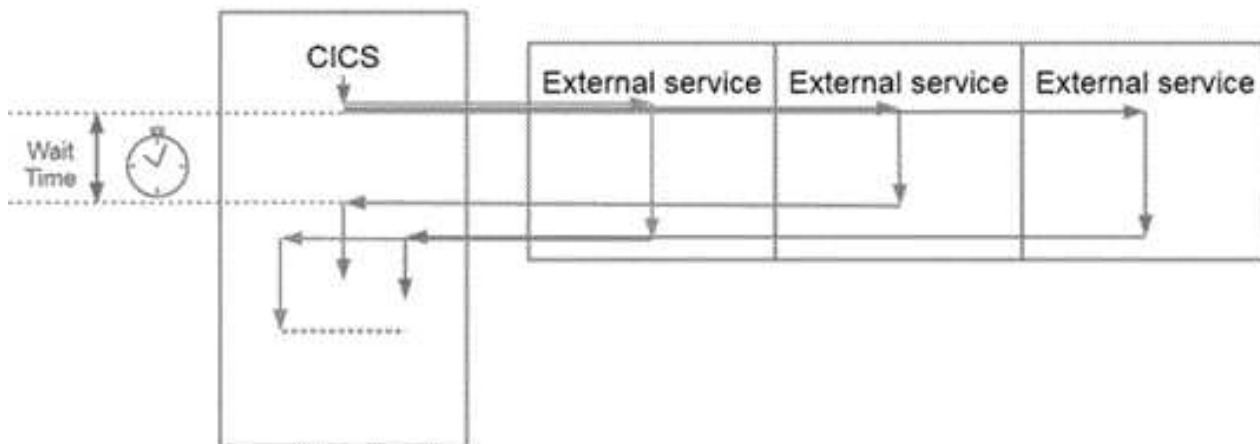


図 2. 並列サービス呼び出し (非同期処理)

非同期処理が使用される状況

非同期処理は、外部要求の処理中にプロセスを停止させたくない状況に適しています。

関連リンク:

- 非同期要求の開発
- 非同期 API のトラブルシューティング

非同期 API を使用する利点

非同期要求処理は CICS TS 5.4 より前から可能でしたが、新しい非同期 API コマンドを使用すれば、非同期処理の設計と実装を簡略化できます。

EXEC CICS RUN TRANSID コマンドと **EXEC CICS FETCH** コマンドを使用することで、アプリケーションは非同期処理に関連する要求、応答、および例外を処理しながら、標準の CICS コマンドと調整を活用し、プログラムの管理と実行の両面で複雑さを最小限に抑えることができます。

非同期処理には以下の 3 つの主な課題がありますが、これらはすべて新しい非同期 API コマンドの一部として対処されています。

- 非同期要求の後に処理を独立して実行する。
- 非同期で実行されるプロセスとサービスの完了状況を追跡する。
- 非同期処理間でデータを通信する。

以前は、大規模な統合環境で非同期処理をサポートするようには設計されていないその他の CICS 機能、サービス、またはサポート製品を組み合わせることで、これらの課題に対処していました。**EXEC CICS START** コマンドと **EXEC CICS DELAY** コマンド、ビジネス・トランザクション・サービス (BTS)、イベント制御ブロック (ECB)、または WebSphere MQ を使用することで、機能する非同期モデルを作成することはできましたが、これはサポートされていない方法であり、多くの場合は複雑な方法でした。

例えば、BTS は長期間の作業単位や、有意義な結果を得るには人による介入が必要な作業単位に関してはうまく機能し、一方で WebSphere MQ などの外部製品を使用すると CICS システム・プログラマーやアプリケーション・プログラマーにかかる負担は軽減されるものの、サポートするインフラストラクチャーの管理に要する負担は増えます。

EXEC CICS RUN TRANSID と **EXEC CICS FETCH** の各コマンドは、通常の CICS ロジックを使用するプログラムやアプリケーションとシームレスに統合されるように設計されており、**EXEC CICS PUT CONTAINER** および **EXEC CICS GET CONTAINER** と組み合わせることで、非同期で実行されているプロセス間のデータ管理を簡略化します。

非同期 API について

CICS TS が提供する一連の新しい非同期 API コマンドにより、アプリケーション開発者は子タスクを非同期で実行できる非同期プログラミング・モデルを使用できます。

非同期 API の要素

非同期 API は、以下の要素で構成されています。

- 要求を送信し、応答を受け取るための API コマンド。
- **EXEC CICS RUN TRANSID** コマンドと **EXEC CICS FETCH** コマンドを発行して子タスクを開始し、対話する親タスク。
- 親プログラムの完了前に子に関連付けられているリソースを解放するための **FREE CHILD** コマンド。
- 別個の作業単位として親タスクとは非同期で実行される子タスク。各親は 1 つ以上の子を持つことができます。
- 必要に応じて親タスクと子タスクの間でデータを渡すためにオプションで利用できるチャンネル。

非同期 API は、親タスクと子タスクの状態を管理して、関連付けられているすべてのリソースをクリーンアップし、チャンネル内のデータを標準として管理します。API を使用すると、Web サービス・アプリケーションなど、従来からの長時間実行タスクで大きなメリットを得られます。このような長時間実行タスクでは、従来の方法の代わりに、必要に応じて子タスクを作成して解放し、システム・フットプリントを最小限に抑えることができますようになります。

非同期 API の仕組み

親タスクとは、**EXEC CICS RUN TRANSID** コマンドと **EXEC CICS FETCH** コマンドを使用して子タスクを開始して対話するプログラムを実行しているタスクです。この例では、親プログラムは **EXEC CICS PUT CONTAINER** コマンドを使用して、子プログラムと通信するためのコンテナとチャンネルを作成し、次に **EXEC CICS RUN TRANSID** コマンドを使用して子タスクを作成します。これにより、*transid* で定義された子プログラムが実行されます。子プログラムは指定されたチャンネルのコピーを受け取り、処理を開始します。その間、親プログラムは他のロジックで処理を続けることができます。子タスクの結果または完了状況が必要な場合、親は **EXEC CICS FETCH CHILD** コマンドを実行できます。子タスクが正常に完了し、チャンネルを渡して戻すと、**EXEC CICS GET CONTAINER** コマンドが使用されて出力が取得されます。

```
EXEC CICS PUT CONTAINER(container) CHANNEL(channel)
                                FROM(struct) FLENGTH(len_struct) BIT
                                RESP(reason) RESP2(response)

EXEC CICS RUN TRANSID(transid) CHILD(child)
                                CHANNEL(channel) RESP(reason) RESP2(response)

...

EXEC CICS FETCH CHILD(child)
                                ABCODE(abcode)
                                COMPSTATUS(child_status)
                                CHANNEL(fetch_channel)
                                RESP(reason) RESP2(response)

EXEC CICS GET CONTAINER(container) CHANNEL(fetch_channel)
                                INTO(struct) FLENGTH(len_struct)
                                RESP(reason) RESP2(response)
```

子プログラムは通常どおりに CICS API コマンドを使用できますが、一般には **EXEC CICS GET CONTAINER** コマンドと **EXEC CICS PUT CONTAINER** コマンドを使用し

て親プログラムと対話します。この例では、子プログラムは **EXEC CICS GET CONTAINER** コマンドを使用して親プログラムから渡されたコンテナを受け取り、他のロジックを実行してコンテンツを変更 (...) した後、**EXEC CICS PUT CONTAINER** コマンドを使用して結果を送信して戻し、親がそれをコンシュームします。

```
EXEC CICS GET CONTAINER(container) CHANNEL(channel)
              INTO(struct) FLENGTH(len_struct)
              RESP(reason) RESP2(response)
```

...

```
EXEC CICS PUT CONTAINER(container) CHANNEL(channel)
              FROM(struct) FLENGTH(len_struct) BIT
              RESP(reason) RESP2(response)
```

チャネルを使用した親プログラムと子プログラムの間の通信はオプションです。例えば、親プログラムで子タスクが正常に完了したかどうかのみを把握する必要がある場合、**EXEC CICS FETCH** コマンドで十分です。

非同期 API でのパフォーマンスの管理

非同期 API により、CICS システム内で多数の同時タスクが発生する可能性があります。CICS は、領域が **MXT** に達するとワークロード管理を自動的に開始します。また、ユーザーは **TRANCLASS** を使用してパフォーマンスを自分で調整できます。トランザクション・トラッキングと統計を使用して、非同期ワークロードを実行している領域のパフォーマンスをモニターすることもできます。

パフォーマンスの調整

TRANCLASS リソースの使用

親トランザクションの **TRANCLASS** を指定することで、任意の時点でシステムで実行される親タスクの最大数を制御できます。また、拡張により、それらの親によって作成される子タスクの数を制御できます。

TRANCLASS の **MAXACTIVE** 属性を使用して、親トランザクションと子トランザクションの合計数がシステムの **MXT** 未満になるようにします。特定の親タスクによって複数の子が作成されることを想定し、子タスクの **MAXACTIVE** 値は親タスクの **MAXACTIVE** 値よりも高い値にする必要があります。

重要: 子トランザクションに **TRANCLASS** を設定する場合は、子トランザクションと親トランザクションに同じ **TRANCLASS** を使用しないでください。使用した場合、システムのスペースが親タスクでいっぱいになり、子タスクを作成するためのスペースがなくなる可能性があります。

CICS システム管理による自動調整

領域が過負荷になると、CICS はワークフローの調整を自動的に開始し、子タスクが作成されすぎないようにします。**RUN TRANSID** コマンドを発行する親タスクは中断され、**ASPARENT** 待機タイプを使用してキューに入れられ、領域のワークロード・レベルが下がると再開されます。

親タスクが再開されると、領域のワークロードが変動する可能性があり、親タスクの中断と再開が間断なく行われる場合があります。これは予期された動作であり、自動システム管理によって CICS が過度な非同期要求から適切に保護されていることを示します。

非同期ワークロードによって領域に定期的な問題が発生する場合、上述のように TRANCLASS を使用してワークロードを調整することを検討してください。

パフォーマンスのモニター

直前のトランザクション・トラッキングの使用

直前のトランザクションのデータの使用して、領域内のタスク間の関係を追跡できます。例えば、子タスクの 1 つが停止している場合、このデータを使用して、その子からの応答を待機している親タスクを特定し、問題の解決法を決定できます。

統計およびモニター・データの使用

非同期サービス統計とモニター・フィールド・データを使用して、非同期ワークロードのモニターと診断に役立てることができます。アプリケーションによって **RUN TRANSID** コマンドと **FETCH** コマンドが発行された数、子タスクが完了するまでの待機時間、および非同期パフォーマンスをモニターして改善するために役立つその他の有用な情報を得ることができます。

CICS ポリシーの使用

CICS ポリシーにより、非同期要求を処理することを目的として導入された新しい規則タイプを使用して非同期ワークロードを管理することもできます。

EXEC CICS 非同期 API コマンド

CICS 非同期 API では、親タスクによる子タスクの作成、取得、およびコントロールを制御するために **EXEC CICS** コマンドが使用されます。

- RUN TRANSID
- FETCH CHILD
- FETCH ANY
- FREE CHILD

非同期 API での JCICS の使用

CICS 非同期 API の JCICS バリエントが、クラスのセット、および `java.util.concurrent.Future` インターフェースの実装として提供されています。

すべての **EXEC CICS** 非同期 API コマンドの Java バリエントが、以下のクラスで提供されています。

- AsyncService
- AsyncServiceImpl
- ChildResponse

Java プログラムから子トランザクションを開始するには、まず、AsyncService の新しいインスタンスを取得してから、子トランザクションを開始する必要があります。AsyncServiceImpl クラスで状態は持続しないため、Future オブジェクトの有効範囲は AsyncService のインスタンスに設定されません。

```
AsyncService asService = new AsyncServiceImpl();
Future<ChildResponse> child = asService.runTransactionId("ABCD");
```

結果の Future オブジェクトは 通常どおりに使用できますが、子タスクのキャンセルに関連するメソッドはサポートされていません。

runTransactionId() はノンブロッキングですが、get() は Future インターフェースに準拠するように設計されており、ブロックします。

CICS API 実装とは異なり、get() または getAny() の呼び出しでは常に子からチャンネルの取得が試行され、チャンネルの所有権は親タスクに即時に移転されます。

get() メソッドを使用して、子タスクから情報を戻して受け取ることができます。以下に例を示します。

```
ChildResponse response = child.get();
```

ChildResponse では、以下のメソッドを使用できます。

- getAbendCode()
- getChannel()
- getCompletionStatus()

ChildResponse には列挙型の CompletionStatus も含まれており、これによって、子タスクから返される可能性のある CVDA 値が列挙されます。

表 1.

Java クラス	メソッド	同等の CICS API
AsyncService	runTransactionId(String transactionId)	RUN TRANSID
	runTransactionId(String transactionId, Channel channel)	RUN TRANSID CHANNEL
AsyncService	freeChild(ChildResponse child)	FREE CHILD
	freeChild(Future<ChildResponse> child)	FREE CHILD
ChildResponse	ChildResponse.getAny()	FETCH ANY CHANNEL COMPSTATUS ABCODE
	ChildResponse.getAny(BlockingAction blockingAction)	FETCH ANY NOSUSPEND CHANNEL COMPSTATUS ABCODE
	ChildResponse.getAny(long timeout, TimeUnit timeUnit)	FETCH ANY TIMEOUT CHANNEL COMPSTATUS ABCODE
Future<ChildResponse>	get()	FETCH CHILD CHANNEL COMPSTATUS ABCODE
	get(long timeout, TimeUnit timeUnit)	FETCH CHILD CHANNEL COMPSTATUS ABCODE TIMEOUT
Future<ChildResponse>	isDone()	FETCH CHILD NOSUSPEND COMPSTATUS

例

```
final String childTransaction = "ABCD";
AsyncService asService = new AsyncServiceImpl();

Future<ChildResponse> child = asService.runTransactionId(childTransaction);

// Logic here to be processed while the child runs asynchronously
```

```
ChildResponse response = child.get();
if (response.getCompletionStatus().equals(CompletionStatus.NORMAL)) {
    System.out.println("Child task completed normally");
}
```

第 3 章 クラウド対応 CICS Transaction Server for z/OS

クラウドは、企業がサービスを提供する方法に関する概念を変えます。クラウドを利用することで、サービスの管理と運用に関する業務効率が向上し、サービスの開発とデプロイの俊敏性も向上します。CICS Transaction Server for z/OS は、既存の CICS TS トポロジーとアプリケーションをクラウド・スタイルのプラットフォームおよびサービスに変換するための、ビルディング・ブロックとしての 3 つの主要な機能を提供します。これらは、プラットフォーム、単一エンティティとしてのアプリケーション、および操作を制御するためのポリシー です。CICS バンドルも、アプリケーションまたはプラットフォームが使用する関連リソースをグループ化および管理する手段としての役割を果たします。

このセクションでは、クラウド対応 CICS TS の主な機能について説明し、クラウド・ソリューションの構築に関係する作業の概要を示します。

プラットフォーム とは

プラットフォームを使用すると、アジャイルなサービス・デリバリー・ランタイムを作成できます。CICS 領域をプラットフォームとしてグループ化することで迅速なアプリケーション・デプロイメントを可能にし、基盤となるトポロジーからアプリケーションを分離することで柔軟性を向上します。プラットフォームで領域を開始すると、それ以降はシステム管理者との対話を必要とせず、それらの領域にアプリケーションがデプロイされます。自動のリソース検証、プロビジョニング、およびプロビジョニング解除により、信頼性が向上します。プラットフォームは、実行時にポリシーを適用することで動的に管理できます。

プラットフォームは、CICS システム・グループ (CSYSGRP) などの、CICSplex SM トポロジー定義を基盤として構築されます。プラットフォームおよびアプリケーションのいずれかをデプロイおよび管理する操作も、CICSplex の単一制御ポイント機能を基に作成されています。CICS クラウド対応機能をすべて利用するには、CICSplex SM をデプロイする必要があります。

詳しくは、19 ページの『仕組み: プラットフォーム』を参照してください。

アプリケーションとは

クラウドのコンテキストでは、アプリケーションによりまったく異なるアプリケーション・リソースが結合されます。これらのリソースはアプリケーションであるため、単一のエンティティとして管理できます。このエンティティをバージョン管理することで、開発、テスト、および実動ライフサイクルを素早く移動させることができます。アプリケーションを使用すると依存関係の管理が向上し、リソース使用量および内部請求に関してアプリケーション全体を測定できます。アプリケーションは、実行時にポリシーを適用することで動的に管理できます。

詳しくは、32 ページの『仕組み: アプリケーション』を参照してください。

ポリシーとは

CICS ポリシー・タスクおよびシステム規則を使用して、重要なシステム・リソースに対する制御を自動化できます。例えば、タスク規則を定義して、タスクが実行するファイル要求数、タスクが使用するストレージ、タスクが使用するプロセッサ時間などのしきい値を設定できます。しきい値を超えた場合に、メッセージの発行、タスクの異常終了、さらにアクションを起動させるイベントの発行といった、いくつかの自動化されたアクションの 1 つが実行されるようにすることができます。ポリシーは実行時に動的に適用できます。

詳しくは、CICS ポリシーを参照してください。

CICS バンドルとは

CICS バンドルとは、成果物や、バンドルとその依存関係を記述したマニフェストが含まれているディレクトリーです。CICS バンドルは、関連したリソースをグループ化して管理する手段を提供します。

また、CICS バンドルは、リソース更新管理のためのバージョン管理方式を提供し、バンドル外部にある他のリソースへの依存関係を宣言できます。アプリケーション開発者は、アプリケーションのパッケージ化とデプロイメント、ビジネス・イベント、およびサービスのために CICS バンドルを使用できます。システム・プログラマーは、CICS バンドルを使用して CICS ポリシーを定義できます。

詳しくは、Defining CICS bundlesを参照してください。

クラウド・ソリューションの構築

CICS クラウド・ソリューションの構築に誰が関与するかと、生成されるトポロジーを見てみましょう。17 ページの図 3 に、さまざまな役割が、どのように連携してソリューションを構築するかを示します。18 ページの図 4 には、システム・トポロジー、および関与する主な成果物を示します。

- ソフトウェア・アーキテクトは、プラットフォームに使用される CICS システム・グループを選択または設計し、プラットフォームにデプロイするためにアプリケーションのコンポーネントをパッケージ化する方法を評価します。
- 開発者はアプリケーションを開発し、アプリケーション・バンドルにそのアプリケーションをパッケージ化して、システム管理者が zFS にセットアップしたプラットフォーム・ホーム・ディレクトリーにエクスポートします。
- システム管理者は、zFS 上にディレクトリーを作成し、それを保護することでプラットフォームを構成し、さらに必要であるがアプリケーションにパッケージ化されていないリソースを作成してインストールします。開発者またはシステム管理者のいずれかがアプリケーションを CICS バンドルにパッケージ化し、その後システム管理者がアプリケーションをプラットフォームにデプロイします。
- システム管理者は、他の CICSplex SM コンポーネントのセキュリティと同様の方法で RACF® セキュリティ・プロファイルをセットアップすることにより、プラットフォームとそこにデプロイされたアプリケーションを保護することができます。

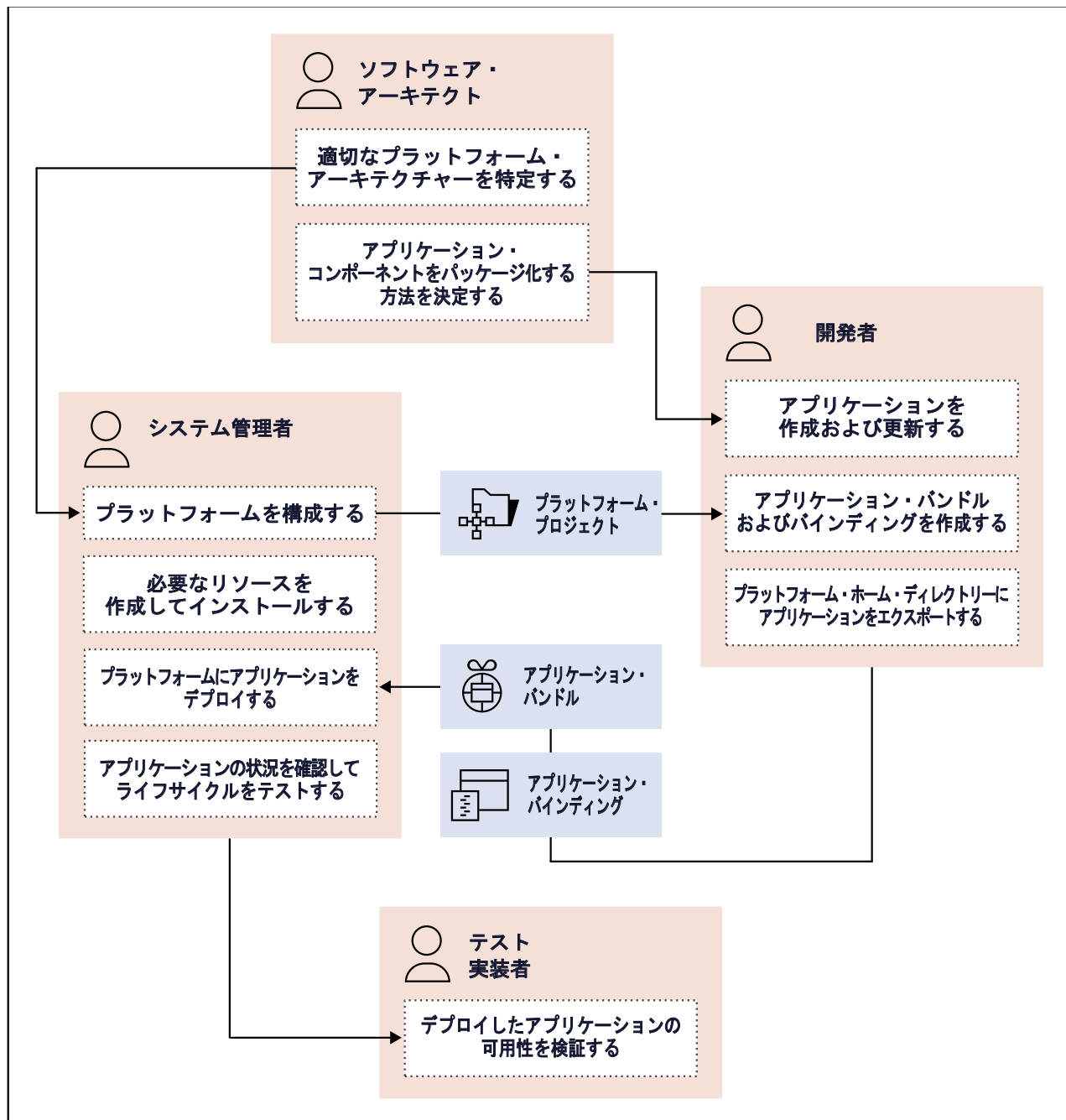


図 3. CICS TS でのクラウド・ソリューションのセットアップには誰が関与するか

以下のトポロジーは、1 つのバージョンのアプリケーション用です。複数バージョン管理を使用する場合は、仕組み: アプリケーションの複数バージョン管理にある同等の図を参照してください。

システム管理者が、CICS Explorer® のプロジェクトを使用してプラットフォームを定義し、アプリケーションをパッケージ化しました。プロジェクトは zFS プラットフォーム・ホーム・ディレクトリーにエクスポートされます。システム管理者はまた、CICSplex SM データ・リポジトリ内のプラットフォームおよびアプリケーションの PLATDEF 定義および APPLDEF 定義も作成しました。PLATDEF 定義は、ターゲット CICS 領域を含む領域タイプでプラットフォームを作成するため

に、CICSplex にインストールされます。APPLDEF 定義は、CICS 領域内のアプリケーションの CICS バンドルを作成するためにプラットフォームにインストールされ、バンドルで定義されているリソースが CICS 領域に動的に作成されます。

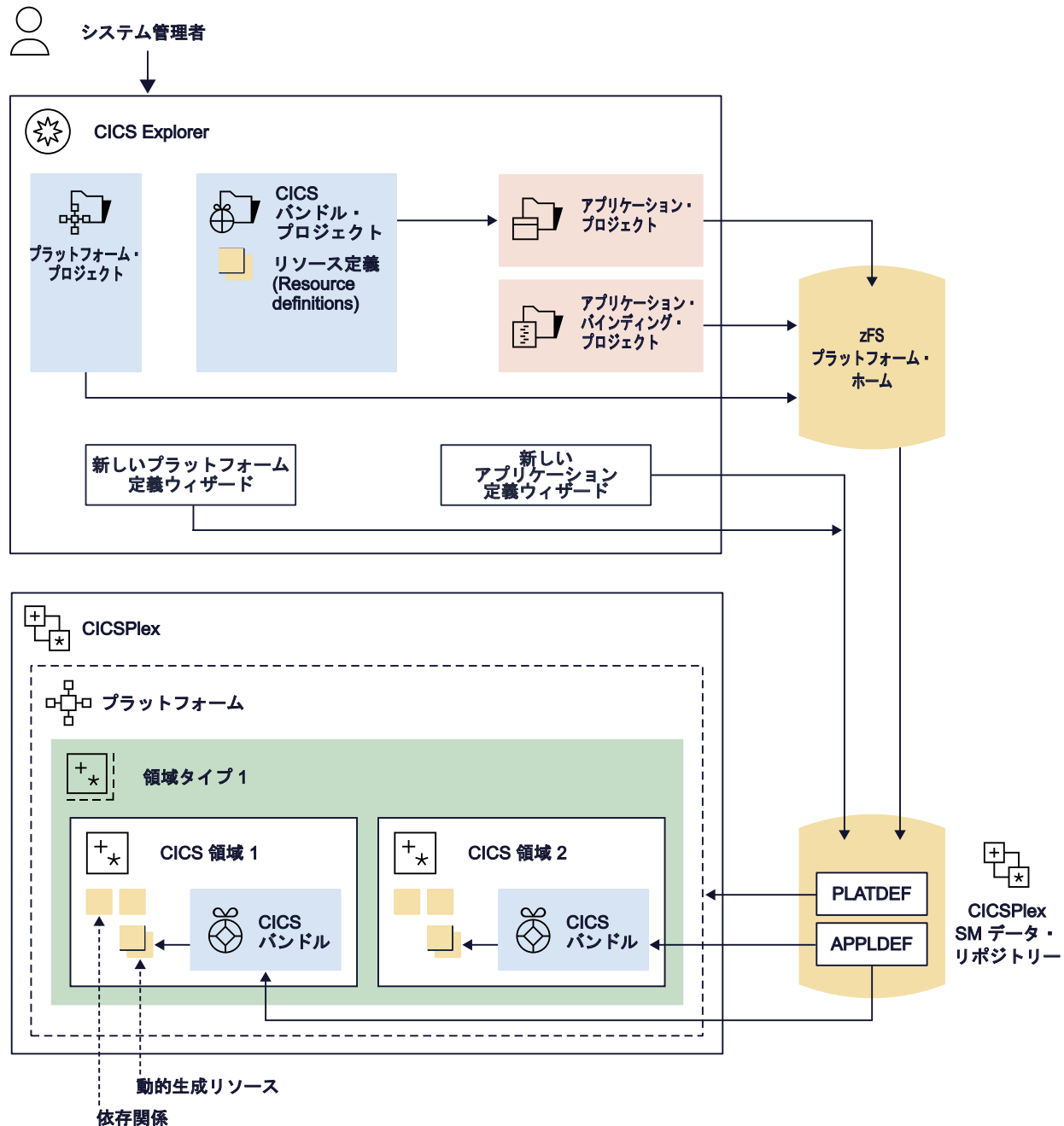


図 4. クラウド・トポロジー

プラットフォームおよびアプリケーション: 必要な情報の参照先

CICS プラットフォームおよびアプリケーションを使用するには、IBM Knowledge Center で以下の項目を参照する必要があります。IBM Redbooks: IBM CICS を使用可能にするクラウド も参照することをお勧めします。この資料では、既存の

COBOL-VSAM アプリケーションを CICS クラウド機能に移行する例として、CICS サンプルの General Insurance Application (GENAPP) が使用されています。

表 2. CICS プラットフォームおよびアプリケーションに関する情報の参照先

トピック	参照先
プラットフォームおよびアプリケーションの動作を学ぶ	<ul style="list-style-type: none">• 仕組み: アプリケーション• 仕組み: プラットフォーム• 仕組み: アプリケーションの複数バージョン管理
プラットフォームをセットアップする	Setting up a platform
アプリケーションをセットアップする	アプリケーションのセットアップ
プラットフォームまたはアプリケーションを保護する	プラットフォームとアプリケーションのセキュリティー
プラットフォームをデプロイする	CICS プラットフォームのデプロイ
アプリケーションをデプロイする	プラットフォームへのアプリケーションのデプロイ
プラットフォームを管理する	プラットフォームの管理
アプリケーションを管理する	アプリケーションの管理
プラットフォームまたはアプリケーションの問題をトラブルシューティングする	プラットフォーム、アプリケーション、およびポリシーのトラブルシューティング
サンプルを見つける	IBM Redbooks: IBM CICS を使用可能にするクラウド

仕組み: プラットフォーム

プラットフォームとは、基盤となる領域トポロジからアプリケーションを分離するための抽象化レイヤーを提供する CICS リソースです。プラットフォームには 1 つ以上の領域タイプが含まれており、それぞれに 1 つ以上の CICS 領域が含まれています。プラットフォームでは、領域定義を作成することも、既存の領域定義を再利用することもできます。アプリケーションをプラットフォーム内の関連する領域タイプに動的にデプロイし、不要になると再び削除することができます。プラットフォームを使用してリソースを制限および制御するためのポリシーをデプロイし、アプリケーションを分離できます。

領域は、CICS TS インストール済み環境におけるサーバー・デプロイメントの基本単位です。CICS TS 領域は、ジョブまたは開始済みタスクとして実行するように定義された IBM z/OS アドレス・スペースです。これらは、コンテキストに応じたさまざまな属性 (ジョブ名、IBM VTAM アプリケーション ID、CICS TS システム ID など) によって識別されます。領域には、データ・セットなど、さまざまな専用リソースが必要です。

ワークロードの増大と領域の数の増加に伴い、スケールと可用性を確保するために、単一の領域の機能を複数領域のグループによって提供する必要があります。これらの領域は、元の単一領域の複製です。これらの複製を 1 つのグループとして処

理することには意味があります。グループに対して実行される各アクションは、そのグループのすべての領域に対して実行されます。プラットフォーム内で、同じ機能を提供する複製された領域のグループは、**領域タイプ** と呼ばれます。

プラットフォームを使用して、以下のことができます。

- プラットフォーム・サービスの状況や宣言された依存関係などのシステムの特性を、単一のポイントから管理します。
- アプリケーションおよびプラットフォーム・サービスをホストします。
- プラットフォームにインストールされているすべてのアプリケーションにポリシーをプロビジョンします。
- プラットフォームがインストールされている場合に、サービスおよび依存関係を動的に追加および削除します。
- 管理された方法で、領域タイプやプラットフォームの間でシステムを共有します。
- プラットフォームが既にインストールされている場合、変更を加えることなく、ホストされているアプリケーションをスケールアップします。

このセクションでは、21 ページの『プラットフォームのコンポーネント』、23 ページの『プラットフォームの例』、および 26 ページの『プラットフォームのセットアップ方法』について説明します。

プラットフォームと CICSplex の違い

CICS TS V5.1 より前は、インフラストラクチャーの境界にマークを付けるために使用できるのは、CICSplex または CICS システム・グループのみでした。プラットフォームを使用すれば、より記述的で明確な境界を指定できます。

CICSplex は、実動やテストなどの環境や、異なるビジネス単位の分離をサポートします。CICSplex 内で、複数のプラットフォームを定義できます。プラットフォームでは、CICSplex と比較して、より記述的な方法を使用します。つまり、プラットフォームのスコープ内にある領域をカプセル化するための一連の領域タイプを提供します。また、プラットフォームでは、領域タイプ間での領域の共有もサポートします。プラットフォームを使用することで、CICSplex 内の問題を分離できます。例えば、開発 CICSplex では、開発者ごとにプラットフォームを割り当てることができます。テストおよび実動 CICSplex では、各業務に個別のアプリケーションを使用できます。領域を統合する場合は、プラットフォームが他のプラットフォームと領域を共有する機能を利用できます。例えば、同じ CICSplex 内のプラットフォームのサブセットに対するルーティング領域を結合できます。

領域にスコープを設定する際、CICSplex またはプラットフォームのいずれを使用するかを選択する場合は、以下のことを考慮してください。

- 異なる業務用、あるいは開発領域とテスト領域用に使用するためなどの理由で、領域を他の領域から分離する必要がある場合は、一般に CICSplex が適切な選択肢です。個別の CICSplex を使用すると、あるスコープで行われた変更が、他のスコープに影響しません。
- 領域を分離する必要がない場合は、プラットフォームの方が適切な選択肢となります。プラットフォームは、要求された場合に限り、他のプラットフォームとソースや領域を共有することもできます。

プラットフォームのコンポーネント

プラットフォームは、以下のもので構成されます。

- プラットフォーム・リソース自体
- 領域タイプ
- PLATDEF リソース定義
- オプションで、プラットフォームで実行されるアプリケーションに提供されるリソースを定義するための CICS バンドル
- オプションで、そのプラットフォームにあるアプリケーションのコンテキスト内で実行されるタスクの実行を制御するためのポリシー。

デプロイされた CICS プラットフォームの成果物は、zFS のディレクトリー内に格納されます。詳しくは、28 ページの『zFS 上のプラットフォーム・ディレクトリー構造』を参照してください。

CICS Explorer を使用して、プラットフォームのコンポーネントを作成および管理します。27 ページの図 9 に、これらのコンポーネント間の関係を示します。他の CICSplex SM コンポーネントのセキュリティと同様の方法で RACF セキュリティー・プロファイルをセットアップすることにより、プラットフォームとそこにデプロイされたアプリケーションを保護することができます。詳細については、プラットフォームとアプリケーションのセキュリティを参照してください。

プラットフォーム

CICS Explorer で定義されるプラットフォーム・バンドルは、プラットフォームとその領域タイプを記述し、プラットフォーム・レベルでデプロイされる CICS バンドルを参照します。プラットフォーム定義をインストールすると、プラットフォーム・バンドルを表す PLATFORM リソースが作成されます。このリソースを使用して、単一のエンティティーとしてプラットフォームを削除できます。

プラットフォーム・リソースは、プラットフォームの現在の状態を反映します。これは、領域タイプ内の領域の全体的なアクティビティー、およびインストールされたプラットフォーム・サービスの全体的な状態の両方を決定します。例えば、各領域タイプの少なくとも 1 つの領域がアクティブで、CICSplex SM に接続されている場合、プラットフォームは ACTIVE です。

プラットフォームは、管理対象のサービスおよび宣言された依存関係の追跡も行います。プラットフォーム・サービスおよび依存関係は CICS バンドルとしてインストールされ、プラットフォームはプロパティー ENABLESTATUS によって、これらのバンドルの正常性を表します。例えば、プラットフォームの各領域タイプにわたり、すべてのバンドルが使用可能になっている場合、プラットフォームはこれを ENABLED と示します。

アプリケーションは環境変更からの保護を必要とし、この保護はプラットフォームのライフサイクルに反映されます。インストールされたプラットフォームは、その中にインストールされているすべてのアプリケーションも破棄されるまで、破棄できません。プラットフォームがインストールされている間は、領域タイプは変更も破棄もできませんが、領域タイプに領域を追加したり、領域タイプから領域を削除したりすることはできます。プラットフォ

ーム・サービスもプラットフォームのライフサイクルに従います。プラットフォームがインストール、有効化、無効化、または破棄されると、定義されているすべてのサービスでも同じアクションが実行されます。

領域タイプ

1 つのプラットフォームには 1 つ以上の領域タイプが含まれます。領域タイプは、タイプに応じて CICS 領域を分類し、格納するために使用されます。例えば、Db2 への接続を処理するすべての CICS 領域を同じグループに含めることができます。

領域タイプは、デプロイメント前にアプリケーションをバインドする必要がある、プラットフォームに対するインターフェースを定義します。プラットフォームがインストールされ、領域タイプが作成されると、定義されたすべての領域に対して領域定義が作成されます。または、領域タイプでは、既存の CICS システム・グループ (CSYSGRP) とそこに含まれる領域を採用することで、既存の環境を再利用することもできます。

領域タイプのレベルで属性を指定することで、作成された領域タイプ内のすべての CICS 領域に対して特定の領域属性値を複製できます。それらの属性に同じ値が指定されている定義、またはそれらの属性の値が指定されていない定義を持つ CICS 領域のみを、その領域タイプに含めることができます。

領域タイプは、同じプラットフォームまたは異なるプラットフォームにある領域タイプと、互いに領域を共用できます。この機能は、複数の異なるプラットフォームを処理するルーティング領域のグループがある場合や、各プラットフォームのファイルをホストするファイル所有領域がある場合の統合シナリオで有用です。

プラットフォーム (PLATDEF) リソース定義

プラットフォーム定義 (CICSplex のデータ・リポジトリに存在する PLATDEF リソース定義) は、プラットフォームのターゲット CICSplex を識別します。PLATDEF のインストール時に、ターゲット CICS 領域を含む領域タイプでプラットフォームが作成されます。

CICS バンドル

プラットフォームでは、アプリケーションに提供されるリソースをインストールおよび管理できます。例えば、プラットフォームでは、URIMAP を使用してアプリケーションが使用できる TCP/IP サービスを提供できます。プラットフォーム・サービスおよび依存関係は、CICS バンドルにインストールされます。この方法でインストールされるサービスはすべて、ADDBUNDLE アクションおよび REMOVEBUNDLE アクションを使用して、プラットフォームで追加、有効化、無効化、および削除することができます。

ADDBUNDLE アクションが実行されると、その領域タイプ内のすべての領域にバンドルがインストールされます。このバンドルは、プラットフォームが以前に有効になっていれば、有効化されます。REMOVEBUNDLE アクションは、その領域タイプ内のすべての領域でバンドルを無効化し、破棄することで、このプロセスを取り消します。

プラットフォームと、インストールされた各 CICS バンドルとの関係は、管理パートで保管されます。管理パートは、プラットフォームのインストー

ル処理中に各 CICS バンドルに対して自動的に作成される MGMTPART レコードです。管理パートには、バンドルがインストールされた CICS 領域が記録され、CICS 領域内でのバンドルの状況が追跡されます。

プラットフォーム・サービスは、プラットフォーム自体が提供する必要はありません。プラットフォームはリソースをバンドルにインポートできます。バンドル・インポートは、プラットフォームの外部で提供されるサービスに対する依存関係を宣言します。例えば、複数のプラットフォームがファイルを共用する必要がある場合、1 つのプラットフォームがバンドル内にファイル定義を作成すると、他のプラットフォームはそのファイル定義をそれぞれのバンドルにインポートします。

ポリシー

プラットフォームは、CICS TS ポリシーをインストールすることもできます。ポリシーは、その特定のプラットフォーム内にあるアプリケーションのコンテキスト内で実行されるタスクの実行に対する契約を提供します。例えば、タスクが契約されたプロセッサ使用量を超えた場合、ポリシーによって異常終了を強制的に実行できます。リソース定義と同様に、ポリシーとその関連ルールが CICS バンドルにインストールされます。

詳しくは、CICS ポリシーを参照してください。

プラットフォームの例

CICSplex 内のすべての管理領域を単一のプラットフォームに定義したり、CICSplex 内の管理領域を複数のプラットフォーム、切り離されたプラットフォーム、または重複するプラットフォーム (複数のプラットフォームに同じ CICS 領域が存在するために生じる) にわたって配置したりすることができます。

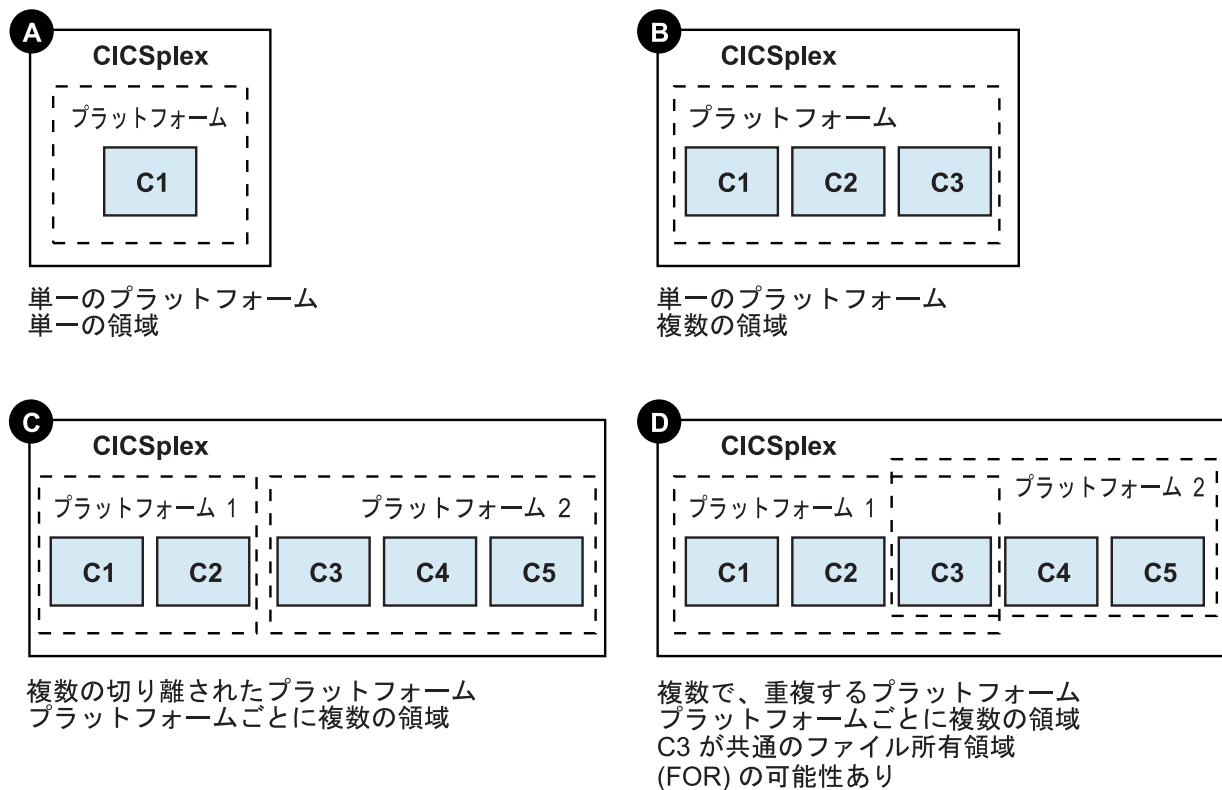


図 5. CICS プラットフォーム・アーキテクチャーの例

次の例では、プラットフォームの管理アドレス・スペース (MAS) である CICS 領域は、Web 所有領域 (WOR)、アプリケーション所有領域 (AOR)、およびファイル所有領域 (FOR) です。CMAS は、WOR、AOR、および FOR だけではなく、プラットフォームも管理する CICSplex SM MAS です。

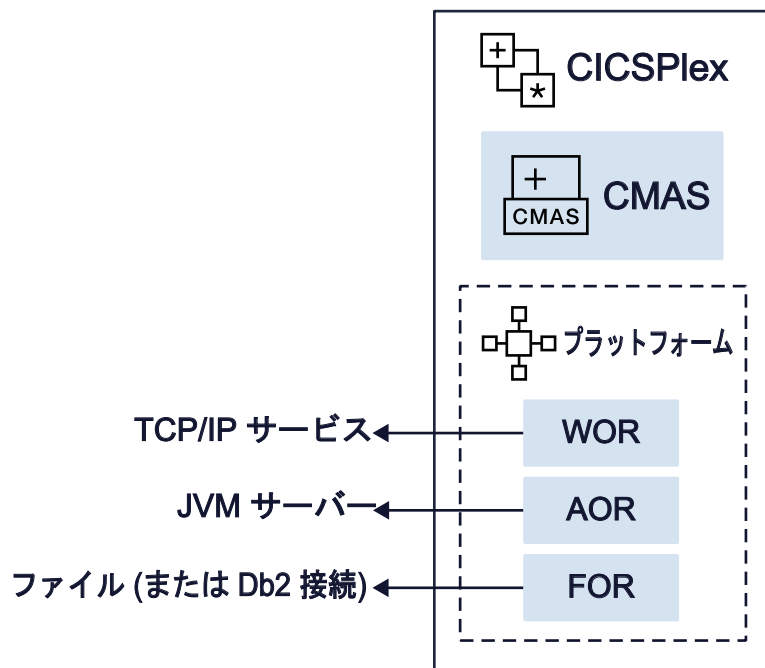
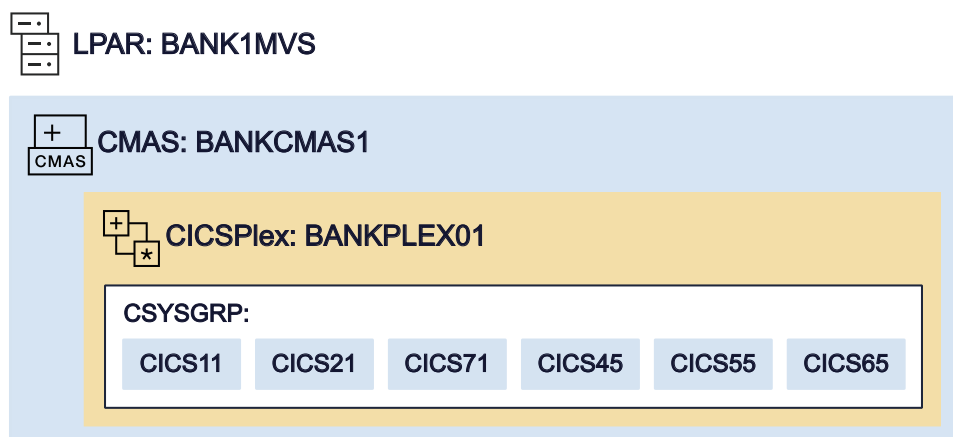


図 6. プラットフォームの機能

論理エンティティとしてプラットフォームを配置し、アプリケーションを分離できます。銀行用 CICSplex の簡略化されたビューについて考えてみましょう。

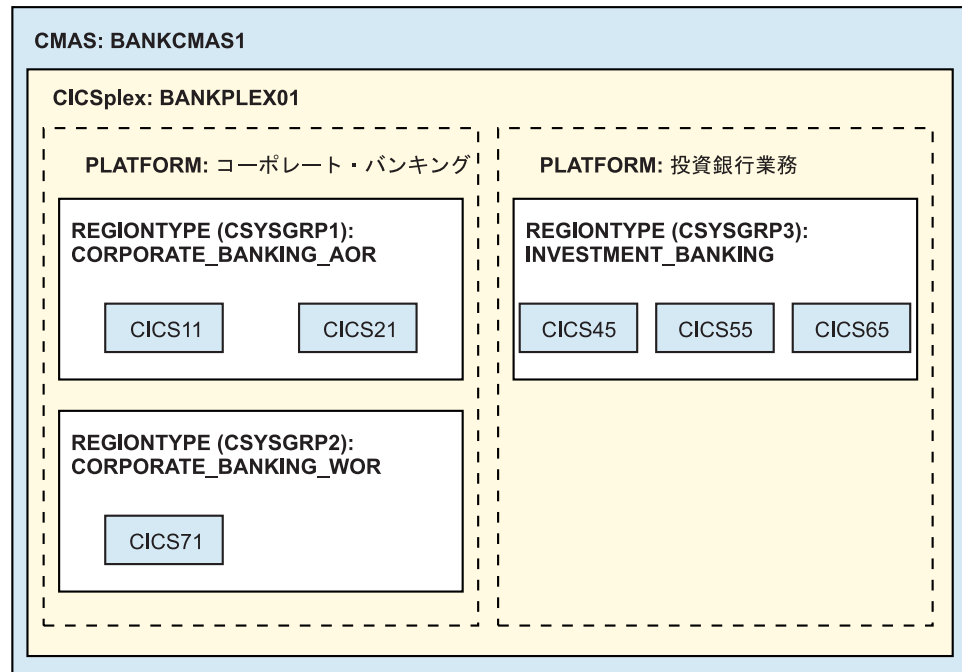
図 7. 銀行用 CICSplex の例



この銀行用 CICSplex を複数のプラットフォームに分割して、コーポレート・バンキングと投資銀行業務のアプリケーションをホスティングするための環境を提供できます。

図 8. 複数のプラットフォームに分割された銀行用 CICSplex の例

LPAR: BANK1MVS



プラットフォームのセットアップ方法

以下に手順の概要を示します。 27 ページの図 9 は、この手順の結果を示しています。

1. プラットフォームを設計します。プラットフォームにデプロイするアプリケーション、ポリシー、およびリソース、および新しい CICS 領域と領域タイプを作成するか、または既存の CICS 領域とシステム・グループ定義を採用するかを検討します。詳しくは、を参照してください。
2. zFS で、プラットフォーム・ホーム・ディレクトリーを構成します。詳しくは、プラットフォーム用の zFS の準備を参照してください。
3. CICS Explorer で、プラットフォーム・プロジェクトを作成します。このプロジェクトに、領域タイプを追加し、デプロイする CICS バンドルとデプロイ先の領域タイプを指定します。詳しくは、Setting up a platformを参照してください。
4. CICS Explorer から、プラットフォーム・プロジェクトを zFS にエクスポートします。エクスポート処理では、CICS プラットフォーム・プロジェクトで参照される CICS バンドルをパッケージ化し、プラットフォーム・バンドルおよび CICS バンドルのすべてのファイルを zFS のプラットフォーム・ホーム・ディレクトリーにエクスポートします。詳しくは、CICS プラットフォームのデプロイを参照してください。
5. CICS Explorer で、プラットフォーム定義を作成します。プラットフォーム定義は CICSplex SM PLATDEF リソース定義であり、これは zFS のプラットフォーム・ホーム・ディレクトリー内のプラットフォーム・バンドルをポイントし、プラットフォームのターゲット CICSplex を識別します。詳しくは、CICS プラットフォームのデプロイを参照してください。

6. プラットフォーム・プロジェクト内の領域タイプで作成したそれぞれの CICS 領域定義に関して、実際の CICS 領域をセットアップします。詳しくは、CICS プラットフォームのデプロイを参照してください。
7. CICS Explorer で、プラットフォームを実行する CICSplex にプラットフォーム定義をインストールします。CICSplex SM はプラットフォーム・バンドル内の情報を使用して、プラットフォームにインストールされているすべての CICS バンドルとともに、プラットフォームをターゲット CICSplex にインストールします。詳しくは、CICS プラットフォームのデプロイを参照してください。
8. CICS 領域を開始します。
9. プラットフォームとともにデプロイされた CICS バンドルがある場合は、CICS Explorer でそれらを有効化し、プラットフォームで使用できるようにします。

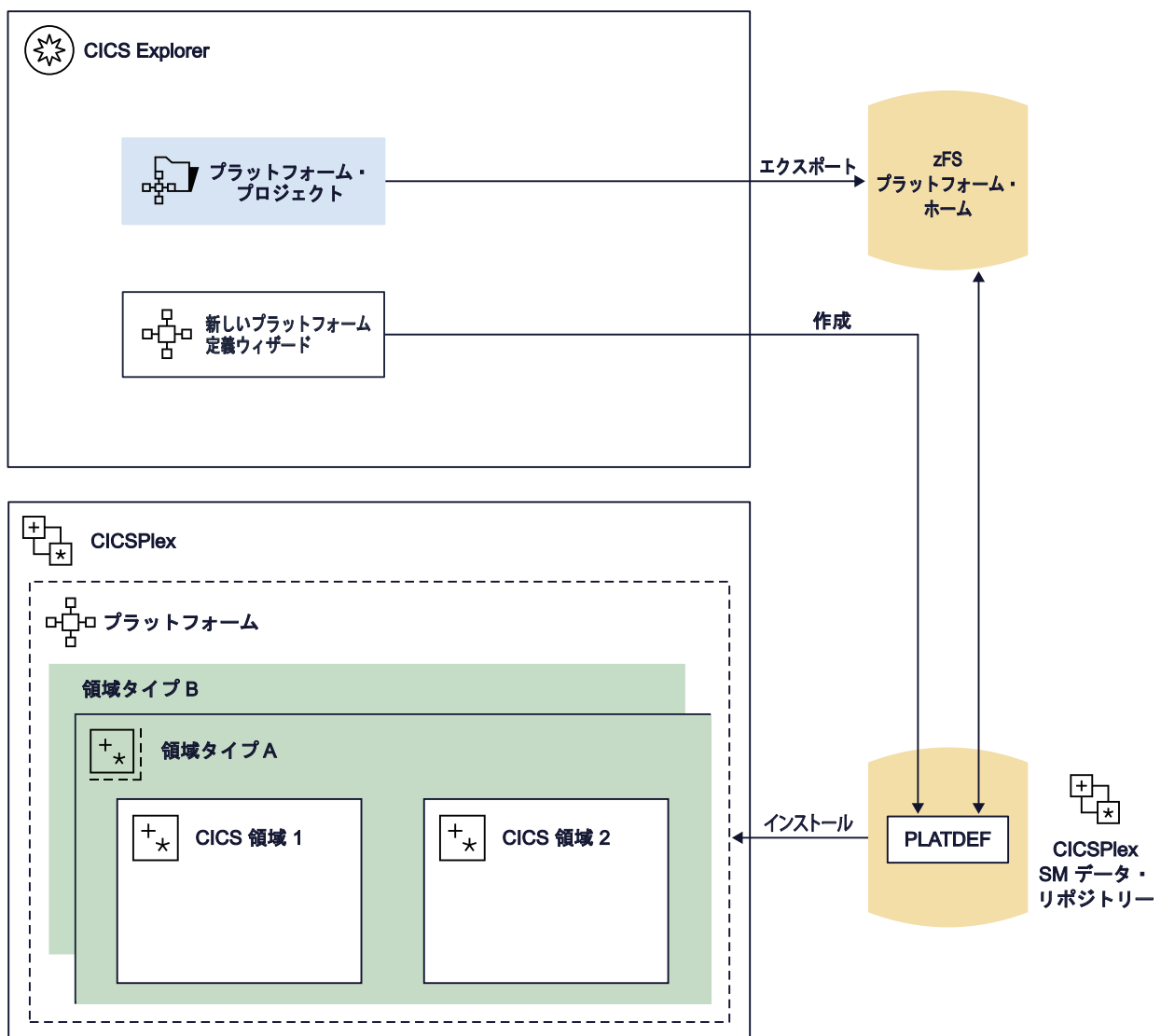


図 9. プラットフォームの要素の作成

zFS 上のプラットフォーム・ディレクトリー構造

デプロイされる CICS プラットフォーム成果物のデフォルトの場所は、ディレクトリー `/var/cicsts/CICSplex/platform1/` です。*CICSplex* はプラットフォームがインストールされている CICSplex の名前で、*platform1* は使用するプラットフォームの名前です。この場所は、プラットフォーム・ホーム・ディレクトリーと呼ばれます。

プラットフォーム・ホーム・ディレクトリーには、標準のディレクトリー構造で、プラットフォーム・バンドル本体のためのディレクトリーと、プラットフォームに関連付けられているアプリケーション、アプリケーション・バインディング、および CICS バンドルのためのディレクトリーが含まれています。プラットフォームのホーム・ディレクトリーを作成する手順については、プラットフォーム用の zFS の準備を参照してください。

プラットフォーム・ホーム・ディレクトリーの標準構造の例を以下に示します。

```
/var/cicsts/CICSPLX1/  
    /platform1/applications/..  
        /bindings/..  
        /bundles/..  
        /platform/platform1  
    /platform2/..  
/CICSPLX2/..
```

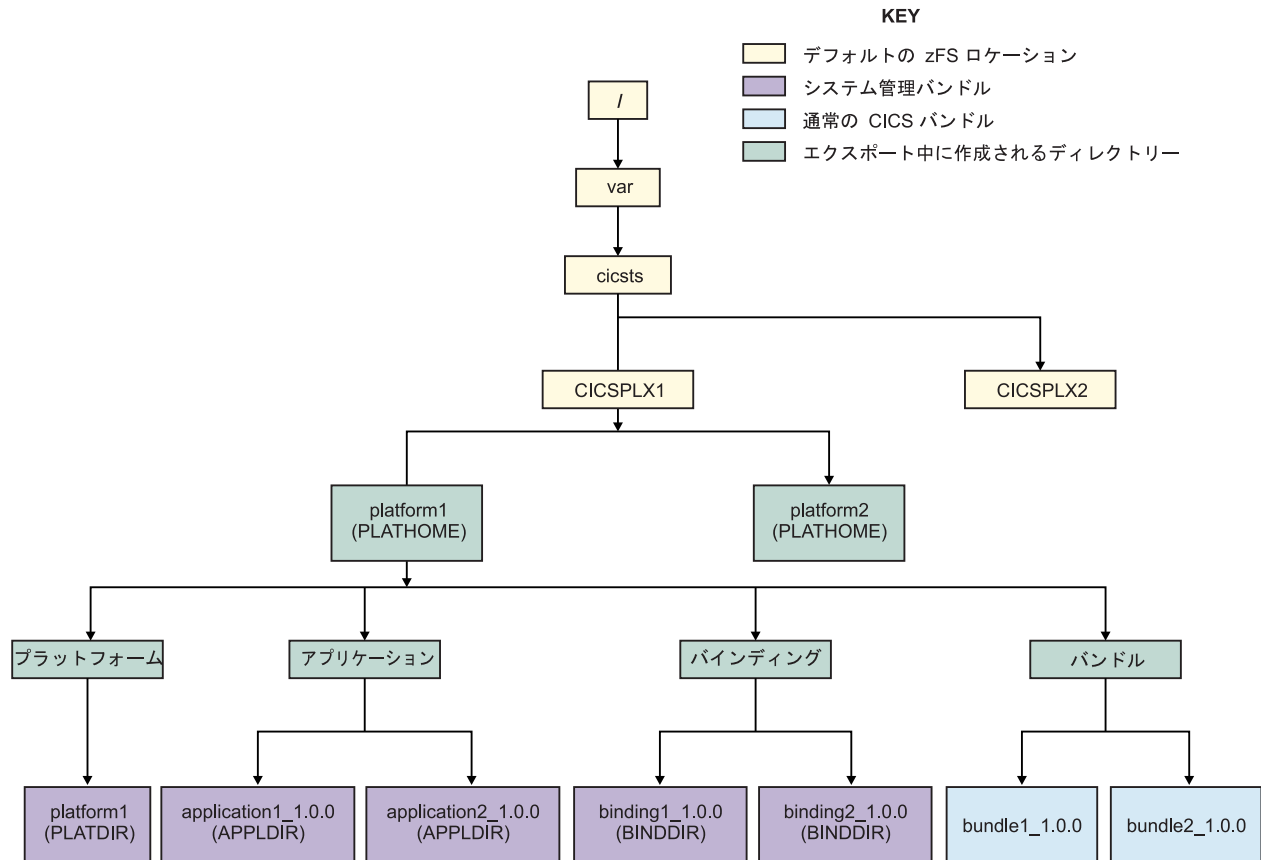


図 10. 標準のプラットフォーム・ホーム・ディレクトリー構造

プラットフォーム・ホーム・ディレクトリー (PLATHOME)

プラットフォーム・ホーム・ディレクトリー (PLATHOME) は、デプロイに備えてアプリケーション・バンドル、アプリケーション・バインディング・バンドル、および CICS バンドルがサブディレクトリーに格納されている zFS 内のディレクトリーです。プラットフォーム・バンドルもこのディレクトリーのサブディレクトリーの 1 つに格納されます。プラットフォームごとに独自のホーム・ディレクトリーがあります。

デフォルトのプラットフォーム・ホーム・ディレクトリーは `/var/cicsts/CICSplex/platform1/` です。*CICSplex* はプラットフォームがインストールされている CICSplex の名前、*platform1* は使用するプラットフォームの名前です。このデフォルトをそのまま使用することをお勧めします。別のディレクトリーをプラットフォーム・ホーム・ディレクトリーとして使用する必要がある場合は、CICS プラットフォーム・プロジェクトの作成後に、プラットフォーム記述子エディターを使用してプラットフォーム・バンドルを変更し、代わりのディレクトリー名を指定する必要があります。

プラットフォーム・ホーム・ディレクトリーには、各種リソースを保持するためのいくつかの異なるディレクトリーがあります。以下のディレクトリーが CICS Explorer からエクスポートされます。

- `applications` には、アプリケーション・バンドルが含まれています。

- `bindings` には、このプラットフォームのアプリケーション・バインディングが含まれています。
- `bundles` には、CICS バンドルが含まれています。これらのバンドルは、アプリケーション内でまたはプラットフォームの一部として使用される場合があります。
- `platform` には、プラットフォーム・バンドルが含まれています。

プラットフォーム・ディレクトリー (PLATDIR)

プラットフォーム・ディレクトリー (PLATDIR) は、プラットフォーム・バンドルが置かれる場所です。エクスポートされたプラットフォーム・プロジェクトが含まれています。エクスポートされたプラットフォーム・プロジェクトには、META-INF ディレクトリーとサブディレクトリーが含まれており、その中には `bundles.xml`、`deployment.xml`、`manifest.xml`、`platform.xml`、`regions.xml`、`regionTypes.xml`、および `regionTypeLinks.xml` の各ファイルがあります。

プラットフォーム・ディレクトリーは `/var/cicsts/CICSplex/platform1/platform/platform1` にあります。

プラットフォームの一部としてデプロイされるバンドルは、`/bundles` サブディレクトリーにコピーされます。

アプリケーション・ディレクトリー

アプリケーション・ディレクトリーは CICS アプリケーション・バンドルが置かれる場所です。これは、CICS Explorer からのプラットフォームのエクスポート中に作成されます。デプロイ済みアプリケーションごとに 1 つのディレクトリー (APPLDIR) が含まれています。このディレクトリーの中には、META-INF ディレクトリーとサブディレクトリーがあり、その中には `applications.xml`、`bundles.xml`、および `manifest.xml` の各ファイルがあります。

アプリケーション・ディレクトリーは `/var/cicsts/CICSplex/platform1/applications/` にあります。

アプリケーションの一部としてデプロイされるバンドルは、`/bundles` サブディレクトリーにコピーされます。

バインディング・ディレクトリー

バインディング・ディレクトリーは、アプリケーション・バインディングが置かれる場所です。これは、CICS Explorer からのプラットフォームのエクスポート中に作成されます。デプロイ済みアプリケーション・バインディングごとに 1 つのディレクトリー (BINDDIR) があります。このディレクトリーの中には、META-INF ディレクトリーとサブディレクトリーがあり、その中に `appbinding.xml`、`bundles.xml`、`deployment.xml`、および `manifest.xml` の各ファイルがあります。

バインディング・ディレクトリーは `/var/cicsts/CICSplex/platform1/bindings/` にあります。

アプリケーション・バインディングの一部として配置されるバンドルは、`/bundles` サブディレクトリーにコピーされます。

バンドル・ディレクトリー

バンドル・ディレクトリーは、CICS バンドルが置かれる場所です。これ

は、CICS Explorer からのプラットフォームのエクスポート中に作成されます。これには、CICS Explorer からエクスポートされた CICS バンドルが含まれます。これらのバンドルは、アプリケーション内でまたはプラットフォームの一部として使用される場合があります。

バンドル・ディレクトリーは `/var/cicsts/CICSplex/platform1/bundles/` にあります。

プラットフォームの状態

CICS Explorer には、プラットフォームの状態情報が示されます。プラットフォームの STATUS 設定値は、領域タイプ内の CICS 領域の状況から導き出されます。この値は、CICS 領域がアクティブであるかないかに応じて、プラットフォームがアクティブか、部分的にアクティブか、非アクティブかを示します。プラットフォームの ENABLESTATUS 設定値は、プラットフォームにインストールされている CICS バンドルの管理パートの状況から導き出されます。この値は、プラットフォームにインストールされた CICS バンドルが、そのプラットフォームの CICS 領域に存在していて、使用可能であるかどうかを示します。

管理パートは、プラットフォームのインストール処理中に自動的に作成される MGMTPART レコードです。管理パートは、プラットフォームとインストールされた各 CICS バンドルの間の関係、およびプラットフォームの中で各 CICS バンドルがインストールされた領域タイプを記録します。

CICS Explorer 製品資料内の『プラットフォームの状況の確認』の CICS Explorer の説明に従ってください。以下の表に、プラットフォームおよび関連する管理パートの状況を表す値を示します。

表 3. プラットフォームに関連付けられている CICS 領域の状態を反映するプラットフォーム状態

STATUS 値	意味
ACTIVE	プラットフォームに関連付けられたすべての領域タイプ内に少なくとも 1 つのアクティブな CICS 領域があります。
PARTIAL	少なくとも 1 つの (ただし、すべてではない) 領域タイプに、アクティブな CICS 領域やプラットフォーム・リソースを管理できる領域がありません。
INACTIVE	すべての領域タイプに、PLATFORM オブジェクトをサポートするアクティブな CICS 領域がありません。

表 4. プラットフォームの使用可能状況を表すプラットフォーム状態

ENABLESTATUS 値	意味
ENABLING	プラットフォームのすべての管理パートは、使用可能にするための処理中です。
ENABLED	プラットフォームのすべての管理パートは使用可能です。
DISABLING	プラットフォームのすべての管理パートは、使用不可にするための処理中です。
DISABLED	プラットフォームのすべての管理パートは使用不可です。
SOMEDISABLED	プラットフォームの一部の管理パートは使用不可です。

表 4. プラットフォームの使用可能状況を表すプラットフォーム状態 (続き)

ENABLESTATUS 値	意味
INSTALLING	プラットフォームはインストール中なので、この時点で使用可能や使用不可にすることはできません。
DISCARDING	プラットフォームは破棄中なので、この時点で使用可能や使用不可にすることはできません。
FAILED	プラットフォームのインストール中または破棄中に問題が発生しました。
INCOMPLETE	プラットフォームの一部の管理パートが空であるか、無効なスコープがあります。
EMPTY	プラットフォームの管理パートはインストールされていません。

表 5. プラットフォームにインストールされている CICS バンドルの管理パートの状態

状況値	意味
ENABLING	CICS バンドルは、使用可能にするための処理中です。
ENABLED	CICS バンドルは、すべてのアクティブな CICS 領域にインストールされていて使用可能です。
DISABLING	CICS バンドルは使用不可にされる過程にあります。
DISABLED	CICS バンドルは、すべての CICS 領域で使用不可です。
SOMEDISABLED	CICS バンドルは、一部の CICS 領域で使用不可です。
IMPORTONLY	CICS バンドルは、すべての CICS 領域にインストールされていて使用可能ですが、インポート・ステートメントしか含まれていないため、プラットフォームの状況には影響しません。
INCOMPLETE	CICS バンドルは、いくつかの (ただし全部ではない) CICS 領域にインストールされています。
INVALIDSCOPE	CICS バンドルをインストールするために指定された CICS システム・グループが存在しないため、CICS バンドルはインストールされません。
EMPTY	CICS バンドルは、いずれの CICS 領域にもインストールされません。

仕組み: アプリケーション

CICS® でビジネス・アプリケーションを構成する大規模なリソースのセットを論理的に単一のエンティティとして定義し、単一のリソースとしてプラットフォームにデプロイできます。この方法で定義されたアプリケーションは、そのライフサイクル全体を通して単一のエンティティとして管理できるため、CICS アプリケーションの管理をさらに迅速かつ容易に行い、エラーの発生を抑えることができます。

アプリケーションは、CICS TS バンドル・リソースを格納する強力で高機能のコンテナを提供します。これらのリソースは、インストールから最終的な破棄まで、アプリケーションのライフサイクルを追跡します。また、リソース、アプリケーションの依存関係、およびポリシーをホストできます。それらの全体的な正常性は、アプリケーションの状況によって追跡できます。さらに、アプリケーションはエントリー・ポイントを提供できます。お客様の照会や支払いの許可など、エントリー

ー・ポイントの宣言済み操作を使用することで、システム管理機能 (SMF) レコードを使用した単純かつ正確なリソース・モニタリングおよび請求が可能になります。また、タスク規則を定義する CICS ポリシーをアプリケーションの一部としてデプロイし、特定の操作に関連するタスクに対してのみ有効にすることもできます。

アプリケーションでは複数バージョン管理をサポートするため、同じプラットフォーム・インスタンス上で、アプリケーションの複数のバージョン (例えば、現行バージョンと新たにパッチが適用されたバージョン) を同時に実行できます。アプリケーションには専用プログラムおよびライブラリー・リソースを含めることができるため、複数のプログラム・リソースが同じ名前を持っている場合でも、各アプリケーション・バージョンが異なるバージョンのプログラムを実行できます。詳しくは、仕組み: アプリケーションの複数バージョン管理を参照してください。

アプリケーションには、以下のような利点があります。

- 既存アプリケーションのコストをアプリケーション・レベルで測定します。
- ポリシーを使用して、リソース使用量が多すぎる、問題のあるアプリケーションから CICS 領域を保護します。
- アプリケーションのライフサイクル全体を通じて、およびアプリケーションが実行されるすべての CICS 領域にわたって、アプリケーション・リソースを単一のエンティティーとして管理します。
- 同じアプリケーションの複数のバージョンを同じ領域にデプロイしたり、異なるアプリケーションのプログラム名が競合する場合でも、それらのアプリケーションを同じ CICS 領域にデプロイしたりすることができます。
- アプリケーションを変更することなく、開発、テスト、および実動環境に移動します。

このセクションでは、『アプリケーションのコンポーネント』および 35 ページの『アプリケーションのセットアップ方法』について説明します。

アプリケーションのコンポーネント

デプロイされた CICS アプリケーションは、以下のもので構成されています。

- CICS アプリケーション本体
- アプリケーションが使用するリソースを定義またはインポートする 1 つ以上の CICS バンドル
- CICS アプリケーション・バインディング
- APPLDEF リソース定義
- 1 つ以上のアプリケーション・エントリー・ポイント (これは、アプリケーション・コンテキストも設定します)
- オプションで、アプリケーションが定義された制限内で実行されるようにするための、タスク規則を定義する CICS ポリシー。

デプロイされた CICS アプリケーション成果物は、アプリケーションが実行されるプラットフォームに関連付けられている zFS のディレクトリーにあります。詳しくは、28 ページの『zFS 上のプラットフォーム・ディレクトリー構造』を参照してください。

CICS Explorer を使用して、アプリケーションのコンポーネントを作成および管理します。37 ページの図 11 に、これらのコンポーネント間の関係を示します。他の CICSplex SM コンポーネントのセキュリティと同様の方法で RACF セキュリティ・プロファイルを設定アップすることにより、プラットフォームとそこにデプロイされたアプリケーションを保護することができます。詳細については、プラットフォームとアプリケーションのセキュリティを参照してください。

Application

CICS Explorer では、CICS アプリケーション・プロジェクトがアプリケーション・バンドルを定義します。アプリケーション・バンドルは、(名前やバージョン情報など) CICS アプリケーションについて記述する一種の管理バンドルです。この管理バンドルは、アプリケーションの依存関係とリソースを含む CICS バンドルを参照します。すべてのアプリケーション・リソースは一緒にインストールおよび管理されます。アプリケーションは、テストの準備ができると zFS にエクスポートされます。zFS は、実行時にアプリケーションが CICS TS に使用される場所です。

CICS バンドル

CICS TS アプリケーションには 1 つ以上の CICS バンドルが含まれ、各バンドルが、アプリケーションが使用する一連のリソースを定義またはインポートします。CICS バンドルは、一連の CICS TS リソース用のコンテナです。CICS バンドルが CICS TS 領域にインストールされると、そこに含まれる一連のリソースもインストールされます。

インポートされたバンドル内のリソースを使用して、アプリケーションは、そのアプリケーションによって定義されていない特定のリソース (例えば、複数のアプリケーションによって共用されるファイル) に対する依存関係を宣言できます。依存関係はインストール時に CICS TS によってチェックされ、依存関係が使用可能でない場合、アプリケーションは有効化されません。

CICS TS アプリケーションは、いくつかの異なるタイプの CICS TS 領域にまたがる場合があります。例えば、アプリケーションは Web 所有領域 (WOR) で要求を処理してから、アプリケーション所有領域 (AOR) でビジネス・ロジックを実行する場合があります。この場合、CICS TS アプリケーションには、WOR のリソースの CICS バンドル、および AOR のリソースの 2 番目の CICS バンドルを含めることができます。CICS バンドルは、アプリケーションのインストール時に zFS で使用可能である必要があります。

アプリケーション・バインディング

CICS アプリケーション・バインディングは、CICS TS アプリケーションと CICS TS プラットフォームの間のリンクです。CICS TS アプリケーションはプラットフォームにデプロイされます。アプリケーションを異なるプラットフォーム (例えば、開発プラットフォームやテスト・プラットフォーム) にデプロイする場合、アプリケーションが変更されないようにする必要があります。これを可能にするには、CICS アプリケーション・バインディングを使用します。アプリケーション・バインディングは、アプリケーション内のバンドル (そのニーズ) を、デプロイ先の特定のプラットフォームにマップします。アプリケーション・バインディングでは、CICS バンドルをアプリケーションと一緒にデプロイしたり、プラットフォームに追加したりする

方法の代替手段として、アプリケーションに CICS バンドルを追加できます。これらのバンドルには、ターゲット・プラットフォームに合わせてアプリケーションの動作を制御またはカスタマイズするポリシーまたはリソース定義を含めることができます。アプリケーション・バインディングは zFS にエクスポートされます。詳しくは、アプリケーション・バインディングを参照してください。

アプリケーション (APPLDEF) リソース定義

アプリケーション定義 (CICSplex のデータ・リポジトリ内にある APPLDEF リソース定義) は、zFS 上でアプリケーションおよびアプリケーション・バインディングを探す場所を CICS TS に伝えます。APPLDEF のインストール時に、アプリケーションがインストールされます。

アプリケーションのエントリー・ポイント

アプリケーション・エントリー・ポイントは、アプリケーションへのアクセス・ポイントとなるリソースを識別します。アプリケーションのエントリー・ポイントは、プラットフォームにデプロイされた異なるバージョンのアプリケーションに対するユーザーのアクセスを制御するために使用されます。PROGRAM、URIMAP、および TRANSACTION リソースをアプリケーションのエントリー・ポイントとして宣言できます。

アプリケーションの一部としてパッケージ化してデプロイする CICS バンドルの 1 つ以上のアプリケーション・エントリー・ポイントを宣言できます。アプリケーションの各エントリー・ポイントは、リソースに対して宣言され、そのリソースによって実行される操作 (作成、読み取り、更新、または削除などの操作) の名前を指定します。詳しくは、アプリケーションのエントリー・ポイントを参照してください。

また、アプリケーションのリソース使用量をモニターし、実行中のアプリケーションを識別するためのアプリケーション・コンテキストを作成するためにも使用されます。タスクがアプリケーションのエントリー・ポイントをパススルーすると、アプリケーション・コンテキスト・データがそのタスクに関連付けられます。アプリケーション・コンテキスト・データは、タスクが使用する他の CICS TS 領域に伝搬されます。アプリケーション・コンテキスト・データをモニター・レコードで使用可能にすることで、アプリケーションによるシステム・リソース使用量を測定する方法を提供できます。詳しくは、アプリケーション・コンテキストを参照してください。

オブジェクトのポリシー

タスク規則を定義する CICS ポリシーは、CICS アプリケーションの一部である CICS バンドルで定義できます。これらのポリシーによって、アプリケーションが特定の要件を満たすようにします。例えば、アプリケーションが指定時間内に完了する必要がある場合です。

詳しくは、CICS ポリシーを参照してください。

アプリケーションのセットアップ方法

以下に手順の概要を示します。 37 ページの図 11 は、この手順の結果を示しています。

アプリケーションはプラットフォームにデプロイされます。

1. どのアプリケーション・エン트리・ポイント、ポリシー、およびリソースをアプリケーションの一部としてデプロイするかを考慮して、アプリケーションを設計します。詳しくは、を参照してください。
2. CICS Explorer で、アプリケーション・リソース、アプリケーションのエン트리・ポイントの依存関係、およびアプリケーションに関連するすべての CICS ポリシーを含む CICS バンドルを作成します。詳しくは、Defining CICS bundlesを参照してください。
3. CICS Explorer で、アプリケーション・バンドルを定義するアプリケーション・プロジェクトを作成します。アプリケーション・バンドルは、必要な CICS バンドルを参照します。詳しくは、クラウド環境でのデプロイメントのための CICS アプリケーションのパッケージ化を参照してください。
4. CICS Explorer で、プラットフォーム内の領域タイプに対するアプリケーションのデプロイメント・ルールを記述するためのアプリケーション・バインディング・プロジェクトを作成します。
5. CICS Explorer から、アプリケーション・プロジェクト、アプリケーション・バインディング・プロジェクト、および CICS バンドルを zFS にエクスポートします。エクスポート処理では、アプリケーション、バインディング、およびバンドルのサブディレクトリーで、zFS のプラットフォーム・ホーム・ディレクトリーにファイルをエクスポートします。詳細については、プラットフォームへのアプリケーションのデプロイを参照してください。
6. CICS Explorer で、アプリケーション定義を作成します。アプリケーション定義は、アプリケーションが実行されるプラットフォームのプラットフォーム・ホーム・ディレクトリーにあるアプリケーション・バンドルとアプリケーション・バインディングをポイントする CICSplex SM APPLDEF リソース定義です。詳細については、プラットフォームへのアプリケーションのデプロイを参照してください。
7. CICS Explorer で、アプリケーションをプラットフォームにインストールします。CICSplex SM は、アプリケーション・バンドルとアプリケーション・バインディングの情報を使用して、アプリケーションを構成する CICS バンドルをプラットフォーム内のすべてのターゲット CICS 領域にインストールします。各領域で、バンドルで指定されたリソースが動的に作成され、さらに CICS によって、依存関係として指定されたリソースが存在することが確認されます。詳細については、プラットフォームへのアプリケーションのデプロイを参照してください。
8. CICS Explorer で、使用可能なアプリケーション・エン트리・ポイントを使用して、ユーザーがアプリケーションを開始できるようにします。詳細については、アプリケーションの管理を参照してください。

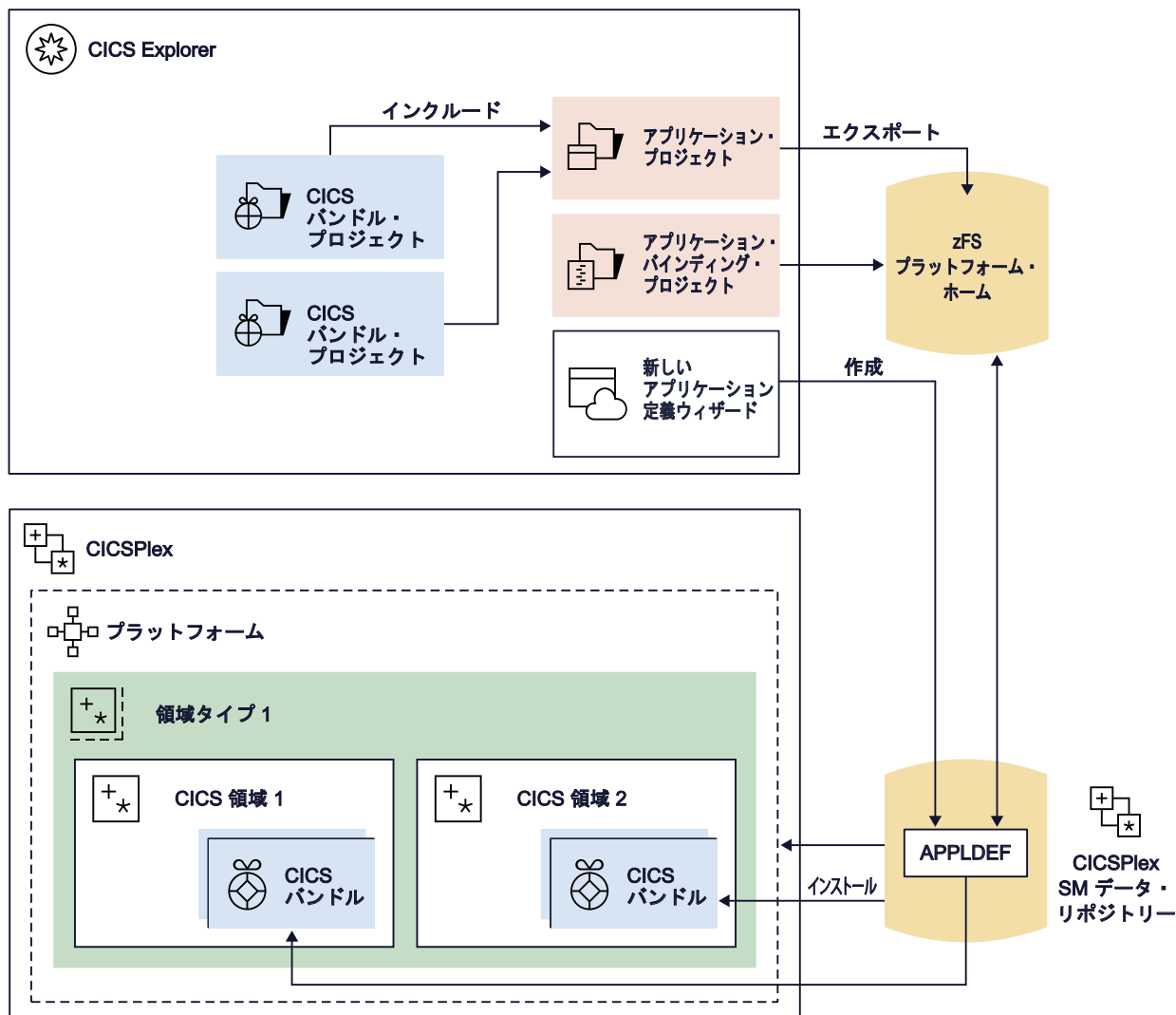


図 11. アプリケーションの要素の作成

アプリケーション・バインディング

アプリケーションをデプロイする前に、ターゲット・プラットフォームにバインドまたはマップする必要があります。アプリケーションの各 CICS バンドルは、プラットフォーム内の 1 つ以上の CICS 領域タイプにバインドする必要があります。アプリケーション・バインディングは、アプリケーションのプラットフォームへのバインド方法を指定し、CICS バンドルがターゲット・プラットフォームに正しくデプロイされるようにします。

単純な CICS トポロジーの例では、単一の CICS 領域に 3 つの CICS バンドルをインストールする必要がある場合があります。より高度なトポロジーの例では、1 つの CICS バンドルをプラットフォーム内の 1 つの領域タイプにデプロイし、他の 2 つの CICS バンドルはいずれもそのプラットフォーム内の別の領域タイプにデプロイする必要がある場合があります。アプリケーション・バンドルではなく、アプ

リケーション・バインディングは、CICS バンドルのデプロイメントを制御するため、アプリケーション自体にデプロイメント情報を組み込みません。CICS バンドルを領域タイプにマップするアプリケーション・バインディングは、アプリケーション・バンドルをインストールするために必要です。

アプリケーション・バインディングでは、異なる CICS リソースを異なるプラットフォームや領域タイプにデプロイする、より複雑なシナリオもサポートされます。例えば、異なる URI を含む URIMAP リソースを、異なるプラットフォームや領域タイプにデプロイできます。このタイプのカスタマイズは、アプリケーション・バインディングに追加の CICS バンドルを含めることによって実現されます。各バンドル (この場合、異なる URIMAP リソースを含む) は、アプリケーション・バインディングで指定しますが、アプリケーション自体を変更する必要はありません。このタイプのアプリケーション・バインディングはオプションです。

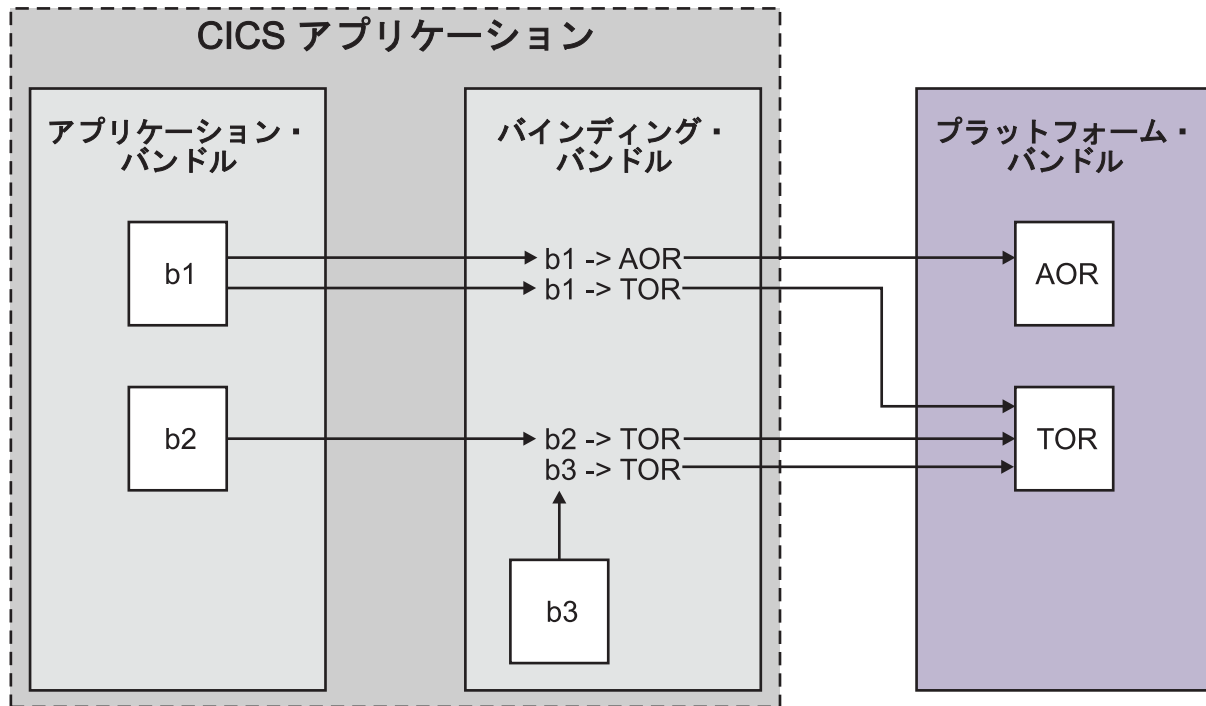


図 12. プラットフォームへのアプリケーションのバインディング例： b1、b2、および b3 は CICS バンドルです。

CICS Explorer でアプリケーション・バインディングを作成します。CICS アプリケーション・バインディング・プロジェクトは、管理バンドルの一種であるアプリケーション・バインディング・バンドルを生成し、バインド・ディレクトリーの詳細でアプリケーション定義を更新します。以下の XML ファイルが作成されます。

- `bundles.xml` には、このアプリケーション・バインディングによってデプロイされる CICS バンドルがリストされています。
- `appbinding.xml` は、アプリケーション・バインディングを特定のアプリケーションおよび特定のプラットフォームに関連付けます。

- `deployment.xml` は、アプリケーションとアプリケーション・バインディングの両方から CICS バンドルをリストします。また、それらをプラットフォームからの領域タイプに関連付けます。

アプリケーション・バインディングの作成後、CICS Explorer でアプリケーション・バインディング記述子エディターを使用して、アプリケーションとアプリケーション・バインディングの両方から CICS バンドルをプラットフォーム内の領域タイプにデプロイする方法を変更できます。アプリケーション・バインディングによってデプロイされる CICS バンドルを、追加したり、削除したりすることもできます。

アプリケーションの CICS バンドルに変更を加えた場合、CICS Explorer を使用してアプリケーションの新規バージョンを指定するようにアプリケーション・バインディングを変更し、さらにアプリケーションのバージョンと一致するようにアプリケーション・バインディングのバージョン番号を更新します。アプリケーション・バインディングの既存のバージョンと、アプリケーション・バンドルの新しいバージョンを一緒に使用することはできません。アプリケーションの CICS バンドルを更新した場合は、必ずアプリケーション・バンドルとアプリケーション・バインディングのバージョンを更新する必要があります。

CICS Explorer を使用してアプリケーション・バインディングを作成、デプロイ、および管理する方法については、CICS Explorer 製品資料内の『CICS アプリケーション・バインディング・プロジェクトの作成』を参照してください。

アプリケーションのエントリー・ポイント

アプリケーション・エントリー・ポイントは、アプリケーションへのアクセス・ポイントとなるリソースを識別します。アプリケーションのエントリー・ポイントは、プラットフォームにデプロイされた異なるバージョンのアプリケーションに対するユーザーのアクセスを制御するために使用されます。また、アプリケーションのリソース使用量をモニターし、実行中のアプリケーションを識別するためのアプリケーション・コンテキストを作成するためにも使用されます。

PROGRAM、URIMAP、および TRANSACTION リソースをアプリケーションのエントリー・ポイントとして宣言できます。

プラットフォームにデプロイされるアプリケーションの場合、アプリケーション・エントリー・ポイントは異なるバージョンのアプリケーションに対するユーザーのアクセスを制御します。アプリケーション・エントリー・ポイントは、ユーザーに対して使用可能または使用不可に設定することができます。アプリケーションとそのリソースは都合の良い任意の時点でプラットフォーム内の CICS 領域にインストールすることができます。その後、CICS バンドルを有効にしてインストールを検証します。アプリケーションのバージョンをユーザーに提供する場合は、アプリケーション・エントリー・ポイントを作成することにより、それらのエントリー・ポイントがアプリケーションのために制御するリソースを呼び出し元に対して使用可能にします。

アプリケーションの一部としてパッケージ化してデプロイする CICS バンドルの 1 つ以上のアプリケーション・エントリー・ポイントを宣言できます。アプリケーシ

ョンの各エン트리・ポイントは、リソースに対して宣言され、そのリソースによって実行される操作 (作成、読み取り、更新、または削除などの操作) の名前を指定します。

- アプリケーションの 1 つのリソースに対してアプリケーション・エン트리・ポイントは 1 回しか宣言できません。そのとき、1 つの操作を指定します。同じリソースに対して複数のアプリケーション・エン트리・ポイントを宣言することはできません。
- オペレーション名はアプリケーション内で固有でなければなりません。
- オペレーション名では大/小文字が区別されるため、大/小文字のみが異なる複数のオペレーション名を使用することもできます。

アプリケーション・エン트리・ポイントのリソースを、アプリケーション・エン트리・ポイントと同じ CICS バンドル内に定義する必要はありません。アプリケーションを使用できるようになると、CICS によって指定されたリソースにアプリケーション・オペレーションが追加されます。アプリケーション・エン트리・ポイントのリソースとエン트리・ポイントの両方がアプリケーションの一部として定義されている場合、エン트리・ポイントによって、リソースで提供されるサービスへのアクセスが制御されます。例えば、アプリケーション・エン트리・ポイントとして定義されている TRANSACTION は、そのアプリケーションが使用可能になるまで呼び出せません。

アプリケーションの外部で定義されているリソースに対してアプリケーション・エン트리・ポイントを宣言できます。このリソースはアプリケーションの一部ではないため、アプリケーション・エン트리・ポイントによって、リソースで提供されるサービスの可用性が変わることはありません。アプリケーションがデプロイされる CICS 領域に自動インストールできる PROGRAM リソースに対して、アプリケーション・エン트리・ポイントを宣言することもできます。アプリケーションをインストールするときに、アプリケーション・エン트리・ポイントのターゲットであるリソースが存在せず、自動インストールできない場合、そのアプリケーション・エン트리・ポイントの宣言が含まれている CICS バンドルは有効になりません。

プラットフォーム・バンドルの一部としてインストールされる、または実行中のプラットフォームに追加される CICS バンドルに、アプリケーション・エン트리・ポイントの宣言を含めることはできません。プラットフォームに直接インストールされる CICS バンドルでは、アプリケーション・エン트리・ポイントはサポートされず、CICS バンドルとそのリソースがインストールされていても、CICS はこの状態でアプリケーション・エン트리・ポイントを有効化しません。CICS 領域に直接インストールされたスタンドアロン CICS バンドルには、アプリケーション・エン트리・ポイントの宣言を含め、領域レベルのポリシーを有効にすることができます。

ユーザー・アクセス権限は、アプリケーション・エン트리・ポイントとして定義されているリソースで制御できます。アプリケーション・エン트리・ポイントとして指定されていない公用リソースがアプリケーションに含まれている場合に、そのアプリケーションをインストールして有効にすると、アプリケーションの使用可能状況とは無関係に、プラットフォーム上または CICS 領域内にインストールされ

た他のアプリケーションからそれらの公用リソースをアクセスできます。アプリケーション・バージョンの専用リソースは、他のアプリケーションからアクセスできません。

アプリケーション・エントリー・ポイントは、タスクに対してアプリケーション・コンテキストを作成するためにも使用されます。アプリケーション・エントリー・ポイントを持つリソースをタスクが呼び出すと、CICS によってアプリケーション・コンテキストが作成され、そのタスクに関連付けられます。アプリケーション・コンテキストは、プラットフォームにデプロイされたアプリケーションの専用リソースを識別、ロード、表示するために使用したり、タスクにポリシーを適用するために使用したり、複数の CICS 領域およびタスクにわたってアプリケーションにより使用されるリソースをモニターするために使用したりします。アプリケーション・コンテキストについて詳しくは、44 ページの『アプリケーション・コンテキスト』を参照してください。

CICS Explorer を使用すると、アプリケーションに対してアプリケーション・エントリー・ポイントを宣言し、アプリケーションを使用可能/使用不可にしたり、有効/無効にしたり、インストールまたは破棄したりできます。アプリケーション・エントリー・ポイントの設定について詳しくは、CICS Explorer 製品資料内の『Defining application entry points』を参照してください。

アプリケーション・エントリー・ポイントとしての **PROGRAM** リソース

アプリケーション・エントリー・ポイントとして宣言されるプログラムには、使用環境内で固有の PROGRAM リソース名が必要です。これらのプログラムをアプリケーション外から呼び出せるようにするためには、それらが公用リソースでなければなりません。専用 PROGRAM リソースに対するアプリケーション・エントリー・ポイントを含んだアプリケーションを使用可能にすると、アプリケーション・エントリー・ポイントとして指定されている PROGRAM リソースが専用リソースから公用リソースに変更されます。特定の名前で CICS 領域内に存在できる公用リソースは 1 つのインスタンスのみです。そのため、専用 PROGRAM リソースは、CICS 領域にインストールされている公用プログラムと同じ名前や、インストールされている別のアプリケーションがアプリケーション・エントリー・ポイントとして定義している公用プログラムと同じ名前にすることはできません。ただし、アプリケーション・エントリー・ポイントとして定義された同じ専用 PROGRAM リソースの複数のバージョンは、同じアプリケーションの複数のバージョンに対してインストールできます。これは、アプリケーションの各バージョンにおける、専用 PROGRAM リソースの公用状態へのプロモーションを CICS が管理しているためです。

PROGRAM リソースへのアプリケーション・エントリー・ポイントを含んだアプリケーションを有効にすると、CICS は以下の検査を実行して、そのリソースをそのアプリケーションで使用するかどうかを確認します。

- アプリケーション・エントリー・ポイントとして指定した PROGRAM リソースが、アプリケーションにパッケージ化された CICS バンドルの 1 つの中で専用 PROGRAM リソースとして定義されている場合、CICS は、その PROGRAM リソースの名前が CICS 領域にインストールされている公用プログラムと同じでないことと、インストールされて有効になっている別のアプリケー

ションによってアプリケーション・エン트리・ポイントとして定義されている
公用プログラムと同じでないことを検査します。

- アプリケーション・エン트리・ポイントとして指定した PROGRAM リソースが、アプリケーションにパッケージ化された CICS バンドルの 1 つの中で定義されておらず、まだ公用プログラムとしてインストールされていない場合は、CICS はそのプログラムの自動インストールを試行した後、そのプログラムをアプリケーション用に予約し、それに対するアプリケーション・エン트리・ポイントを有効にします。この処置が実行された場合、自動インストールされたプログラムは、アプリケーションのバージョンを使用不可にした時点で専用プログラムになるので、アプリケーションの将来のバージョンに対して同じ処置を実行できます。
- アプリケーション・エン트리・ポイントとして指定した PROGRAM リソースが公用プログラムとして既にインストールされている場合は、CICS はインストールされて有効になっている別のアプリケーションによってその PROGRAM リソースがアプリケーション・エン트리・ポイントとしてまだ定義されていないことを検査した後、公用プログラムをアプリケーション用に予約し、それに対するアプリケーション・エン트리・ポイントを有効にします。この処置が実行された場合、アプリケーションの将来のバージョンに対して CICS がアプリケーション・エン트리・ポイントを自動的に管理することはできません。そのアプリケーションの前にインストールされていた公用プログラムを専用プログラムにすることはできないからです。アプリケーションを新しいバージョンに更新するには、既存のバージョンを無効にして破棄する必要があります。この状態を回避するには、公用プログラムがアプリケーションのインストール・プロセスで自動インストールされるように調整するという方法があります。そうすれば、そのプログラムは専用プログラムにすることができるため、将来のアプリケーション・バージョンも同時にインストールできるようになります。別の方法として、アプリケーション・バージョンと一緒にデプロイされる CICS バンドルの 1 つの中でプログラムを定義し、専用プログラムにするために固有の名前を持つようにするという方法もあります。

有効化プロセスが正常に完了すると、アプリケーション・エン트리・ポイントが ENABLED 状態になりますが、まだこのアプリケーション・エン트리・ポイントを使用して呼び出し元からプログラムを使用できる状態にはなっていません。CICS Explorer を使用してアプリケーションを使用可能にすると、CICS でアプリケーション・エン트리・ポイントを使用して呼び出し元からアプリケーションにアクセスすることが可能になり、アプリケーション・エン트리・ポイントの専用 PROGRAM リソースが公用リソースになります。呼び出し元では、**EXEC CICS LINK** コマンドを使用して利用可能な最も高いアプリケーション・バージョンにアクセスするか、**EXEC CICS INVOKE APPLICATION** コマンドを使用して任意の利用可能なアプリケーション・バージョンを指定することができます。同じアプリケーションの複数のバージョンが使用可能な場合は、アプリケーション・バージョンが最も高い PROGRAM リソースが公用になり、それ以外のものが専用になります。

1 つのアプリケーション・バージョンを使用不可にすると、そのアプリケーション・エン트리・ポイントに対する PROGRAM リソースはアプリケーションのために予約された状態を保ちますが、呼び出し元からは使用できなくなります。そのアプリケーションの一部として定義された専用リソースとして開始された PROGRAM リソース、または有効化プロセスの際に自動インストールされた

PROGRAM リソースは、公用プログラムから専用プログラムに戻ります。1 つのアプリケーション・バージョンを無効にすると、そのアプリケーション・エントリー・ポイントに対する PROGRAM リソースはアプリケーションのために予約された状態ではなくなります。公用リソースとして開始されたすべての PROGRAM リソースが、他のアプリケーションおよび呼び出し元から使用可能になります。

アプリケーション・エントリー・ポイントとして宣言されたプログラムは、CICS によって生成されたプログラム定義および JVM プログラムに関する公用リソース統計と専用リソース統計の両方で識別されて報告されます。これは、エントリー・ポイントは公用としてアクセス可能であると同時に、アプリケーションの一部でもあるからです。ローダー・ドメインによって生成されるプログラム統計に関しては、アプリケーション・エントリー・ポイントは識別されず、1 つの専用プログラム統計レコードのみが書き込まれます。

CICS 領域に直接インストールされているスタンドアロン・バンドルにアプリケーション・エントリー・ポイントを定義し、特定の初期 PROGRAM の CICS タスクに対する領域レベルのポリシーのスコープ設定を有効にすることもできます。ただしこの場合、PROGRAM によって提供されるサービスは PROGRAM リソースをインストールして有効にするとすぐに使用可能になり、アプリケーション・エントリー・ポイントを定義するバンドルの可用性によって制御されません。

アプリケーション・エントリー・ポイントとしての **TRANSACTION** リソース

アプリケーション・エントリー・ポイントとして宣言されるトランザクションには、使用環境内で固有の TRANSACTION リソース名が必要です。

TRANSACTION リソースは CICS バンドルに定義し、アプリケーションの一部としてパッケージ化できますが、アプリケーションの専用リソースとしてはサポートされないため、複数のアプリケーション・バージョンにインストールすることはできません。アプリケーションの複数のバージョンを同時にデプロイする予定の場合、CICS バンドルに定義されている TRANSACTION リソースを含むアプリケーションを更新するときに、アプリケーションのバージョンごとに TRANSACTION リソースの名前を変更する必要があります。

CICS アプリケーション・バンドルに TRANSACTION リソースを定義する場合、アプリケーション・エントリー・ポイント宣言を使用して、TRANSACTION リソースで提供されるサービスへのユーザーのアクセス権限を制御できます。アプリケーションをインストールして有効にした時点ではまだ、TRANSACTION リソースで提供されるサービスを呼び出し元が使用することはできません。

TRANSACTION リソースで提供されるサービスとアプリケーション・エントリー・ポイントを呼び出し元が使用できるようにするには、CICS Explorer を使用してアプリケーションを使用可能にすることで、アプリケーション・エントリー・ポイントと TRANSACTION リソースを使用可能にします。

また、TRANSACTION リソースがアプリケーションの外部で定義されている場合、TRANSACTION リソースをアプリケーション・エントリー・ポイントとして宣言することもできます。この場合、TRANSACTION リソースをインストールして使用可能にするとすぐに、ユーザーはサービスを使用できるようになります。

アプリケーション・エントリー・ポイントをスタンドアロン・バンドルに定義して (バンドルが CICS 領域に直接インストールされている場合)、特定の TRANSACTION の CICS タスクに対する領域レベルのポリシーのスコープ設定を有効にすることもできます。TRANSACTION リソースで提供されるサービスに対するアクセスもアプリケーション・エントリー・ポイントによって制御する場合、アプリケーション・エントリー・ポイントと TRANSACTION を同じ CICS バンドルで宣言します。この場合、TRANSACTION で提供されるサービスは、バンドルが使用可能になるまで、呼び出し元が使用することはできません。アプリケーション・エントリー・ポイントと TRANSACTION 定義が同じスタンドアロン CICS バンドル内でない場合、TRANSACTION によって提供されるサービスは TRANSACTION を使用可能にするとすぐに呼び出し可能になります。

アプリケーション・エントリー・ポイントとしての **URIMAP** リソース

アプリケーション・エントリー・ポイントとして宣言される URIMAP リソースには、使用環境内で固有の名前が必要です。URIMAP リソースは CICS バンドルに定義し、アプリケーションの一部としてパッケージ化できますが、アプリケーションの専用リソースとしてはサポートされないため、複数のアプリケーション・バージョンにインストールすることはできません。アプリケーションの複数のバージョンを同時にデプロイする予定の場合、CICS バンドルに定義されている URIMAP リソースを含むアプリケーションを更新するときに、アプリケーションのバージョンごとに URIMAP リソースの名前を変更する必要があります。

アプリケーション・エントリー・ポイントをスタンドアロン・バンドルに定義して (バンドルが CICS 領域に直接インストールされている場合)、特定の URIMAP リソースの CICS タスクに対する領域レベルのポリシーのスコープ設定を有効にすることもできます。URIMAP リソースで提供されるサービスに対するアクセスもアプリケーション・エントリー・ポイントによって制御する場合、アプリケーション・エントリー・ポイントと URIMAP を同じ CICS バンドルで宣言します。この場合、URIMAP で提供されるサービスは、バンドルが使用可能になるまで、呼び出し元が使用することはできません。アプリケーション・エントリー・ポイントと URIMAP が同じバンドルに定義されていない場合、URIMAP によって提供されるサービスは、URIMAP リソースをインストールして使用可能にするとすぐに使用可能になります。

アプリケーション・コンテキスト

アプリケーション・コンテキストは、アプリケーションまたはプラットフォームのコンテキストで実行されているタスクを識別する一連のデータです。アプリケーションがそのアプリケーション・エントリー・ポイントを介してアクセスされると、そのアプリケーションに関連するタスクのアプリケーション・コンテキスト・データが CICS 領域によって生成されます。このタスク・アプリケーション・コンテキストは、CICS モニター機能が SMF に書き込むパフォーマンス・レコードで使用でき、プラットフォーム、アプリケーション、アプリケーションのバージョン、および操作に関するデータ・フィールドが含まれます。この情報は、アプリケーション (またはアプリケーションに入る特定のルート) によるリソース消費量を計測したり、アプリケーションに対してポリシー・ベースの管理を行うために使用したり、タスクを特定のアプリケーションに関連付けて問題診断に利用したりできます。

タスクは、実行時に 1 つ以上のアプリケーションをパススルーできます。各タスクには、任意の時点で最大で次の 2 つのアプリケーション・コンテキストを関連付けることができます。

タスクの初期 アプリケーション・コンテキスト

CICS 領域全体および複数のタスクにわたり、アプリケーションまたは特定のアプリケーション操作が使用しているリソース量のモニターおよび測定に使用します。初期アプリケーション・コンテキストは、アプリケーションの一部であるタスクにポリシーを適用するときに、タスクの動作を管理するためのしきい値条件を定義するために使用できます。初期アプリケーション・コンテキストは、呼び出し元タスクから継承するか、またはタスクが最初にアプリケーション・エントリー・ポイントをパススルーする時点で設定することができます。

タスクの現行アプリケーション・コンテキスト

専用ライブラリーおよび WLM ユーザー出口をロードするために使用されます。現行アプリケーション・コンテキストを照会するには、XPI 呼び出し、SPI 呼び出し、および API 呼び出しを使用できます。現行アプリケーション・コンテキストは、タスクがアプリケーション・エントリー・ポイントをパススルーするたびに変更されます。

初期アプリケーション・コンテキストまたは現行アプリケーション・コンテキストはいずれも、CICS Explorer のトランザクション・トラッキング機能とともに使用して、アプリケーション関連の問題を迅速に識別し、診断することができます。初期アプリケーション・コンテキストおよび現行アプリケーション・コンテキストは、どちらもタスクからタスクへと伝搬していきます。

アプリケーション・コンテキストの作成

アプリケーション・コンテキストを作成するには、CICS リソースを、プラットフォームにデプロイされたアプリケーションのアプリケーション・エントリー・ポイントとして宣言する必要があります。アプリケーション・エントリー・ポイントについて詳しくは、39 ページの『アプリケーションのエントリー・ポイント』を参照してください。

注: **EXEC CICS XCTL** コマンドまたは **COBOL** 呼び出し (静的または動的) を使用したアプリケーション・コンテキストの作成はサポートされていません。

アプリケーション・コンテキストを持たないタスクがアプリケーション・エントリー・ポイントとして宣言されているリソースを呼び出すと、CICS がアプリケーション・コンテキストを作成し、それが初期アプリケーション・コンテキストとしてタスクに関連付けられます。さらに、このアプリケーション・コンテキストは、そのタスクが呼び出す後続のプログラムおよびそのタスクが開始するタスクにも関連付けられます。タスクが既にアプリケーション・コンテキストを持っている場合は、新しいアプリケーション・コンテキストはそのタスクの現行アプリケーション・コンテキストになります。タスクの初期アプリケーション・コンテキストは、その後も引き続きポリシーのモニターおよびスコープ設定のために使用されます。

アプリケーション・コンテキストには以下のデータが含まれます。

- アプリケーション名
- アプリケーションのメジャー・バージョン番号

- アプリケーションのマイナー・バージョン番号
- アプリケーションのマイクロ・バージョン番号
- アプリケーションがデプロイされているプラットフォームのプラットフォーム名
- アプリケーション・エントリー・ポイントのオペレーション名

専用リソースを表示するためのアプリケーション・コンテキストを指定するには、**EXEC CICS INQUIRE** システム・プログラミング・コマンドを使用します。デフォルトでは、CICS は、**EXEC CICS INQUIRE** コマンドが発行されたプログラムで使用可能な専用リソースおよび公用リソースを検索します。異なるアプリケーション・コンテキストを指定すると、別のアプリケーションで使用可能な専用リソースおよび公用リソースを表示できます。アプリケーションによって使用されるリソースを表示するためにアプリケーション・コンテキストを使用する場合、オペレーション名は指定しません。

アプリケーション・コンテキスト・データは DFHDYPDS パラメーター・リストおよび EYURWCOM パラメーター・リストに入れて受け渡しされ、カスタム動的ルーティング・アルゴリズムで使用することができます。動的ルーティングについて詳しくは、CICSplex SM を使用した動的ルーティングを参照してください。

現行アプリケーション・コンテキストの表示

現行アプリケーション・コンテキストを表示するには、以下の方法があります。

- CICS Explorer の「タスク関連 (Task Association)」ビューを使用します。
- **EXEC CICS INQUIRE ASSOCIATION** コマンドを使用します。
- **EXEC CICS ASSIGN** コマンドを使用します。
- モニター XPI 関数 `INQUIRE_APP_CONTEXT` を使用してグローバル・ユーザー出口内から現行アプリケーション・コンテキストを照会します。
- `JCICS Task.getApplicationContext()` メソッドを使用して現行アプリケーション・コンテキストを照会します。

アプリケーション・コンテキストが伝搬されない場合

デフォルトでは、初期アプリケーション・コンテキストおよび現行アプリケーション・コンテキストはタスクからタスクへと伝搬します。ただし、一部の CICS コマンド、インターフェース、接続タイプ、およびその他のプロセスでは、タスク間のアプリケーション・コンテキストの伝搬がサポートされません。

以下のシナリオでは、アプリケーション・コンテキストの伝搬はサポートされません。

- **TERMID** オプションを指定する **START** コマンドによってタスクがアタッチされる場合
- **DTP** または **CPIC** 要求によってタスクがアタッチされる場合。
- **APPC** 接続を介してタスクがアタッチされる場合。
- トランザクション開始 **EP** アダプターを使用してタスクがアタッチされる場合。
- **ThreadExecutor** サービスが新しいスレッドを作成する際に **JVM** サーバーによってタスクがアタッチされる場合。

- Web サービスのパイプライン・ハンドラー・トランザクションが MRO 接続上でルーティングされる場合。
- CICS Web サポートを使用して CICS に対するアウトバウンド HTTP 要求が行われる場合。
- Java プログラム内でスレッドを開始するために `CICSExecutorService` を呼び出す Liberty のアプリケーション・エントリー・ポイントとして、URIMAP リソースを使用してタスクが作成される場合。

タスクにアプリケーション・コンテキストを設定する条件

PROGRAM リソース

PROGRAM リソースをアプリケーション・エントリー・ポイントとして定義すると、以下のいずれかの条件が当てはまる場合にアプリケーション・コンテキストがタスクに設定されます。

- PROGRAM リソースの名前を指定する CICS トランザクションが実行されます (コンテキストは指定のリソースが実行される前に設定されます)。
- プログラムが PROGRAM リソースに対して **EXEC CICS LINK** コマンドを実行します。
- プログラムが、PROGRAM リソースと等価にする操作を指定する **EXEC CICS INVOKE APPLICATION** コマンドを実行します。
- プログラムが PROGRAM リソースに対して **EXEC CICS RUN TRANSID** コマンドを実行します。
- Java プログラムが PROGRAM リソースに対して `Program.link()` コマンドを実行します。
- Java プログラムが、PROGRAM リソースと等価にする操作を指定する `Application.invoke()` コマンドを実行します。

注:

- アプリケーション・エントリー・ポイントとして定義されたプログラムについてタスクにアプリケーション・コンテキストが設定されるのは、そのプログラムがローカルで実行される場合のみです。プログラムがリモートで実行されている場合、要求はリモート CICS 領域に渡され、その CICS 領域においてもリソースがアプリケーション・エントリー・ポイントとしてフラグが立てられていると、そのリモート CICS 領域にコンテキストが設定されます。
- タスクのアプリケーション・コンテキストは、別のプログラムに対して動的呼び出しを行う COBOL プログラムでも、別のプログラムを呼び出すために **FETCH** および **CALL** を発行する PL/I プログラムでも (つまり、**EXEC CICS LINK** 呼び出しがバイパスされる場合)、変更されません。

TRANSACTION リソース

TRANSACTION リソースをアプリケーション・エントリー・ポイントとして定義すると、以下のいずれかの条件が当てはまる場合にアプリケーション・コンテキストがタスクに設定されます。

- TRANSACTION リソースが実行される (指定されたリソースが実行される前にコンテキストが設定されます)。
- プログラムによって、TRANSACTION リソースに対して **EXEC CICS START** コマンドが発行される。

URIMAP リソース

URIMAP リソースをアプリケーション・エントリー・ポイントとして定義すると、次のようにアプリケーション・コンテキストがタスクに設定されます。

- HTTP/HTTPS 要求の URL が USAGE(JVMSEVER) を指定された URIMAP リソースと一致する場合は、URIMAP リソース内で指定されている CICS トランザクションを実行するタスクにアプリケーション・コンテキストが設定されます。
- HTTP/HTTPS 要求の URL が USAGE(SERVER) を指定された URIMAP リソースと一致し、静的応答を提供することになっている場合は、CWXXN タスクにアプリケーション・コンテキストが設定されます。静的応答は、HFSFILE 値または TEMPLATENAME 値を指定することによって提供されます。
- HTTP/HTTPS 要求の URL が USAGE(SERVER) を指定された URIMAP リソースと一致し、静的応答を提供することになっていない場合は、CWXXN タスクにアプリケーション・コンテキストが設定され、このアプリケーション・コンテキストは URIMAP リソースで指定されている CICS トランザクションを実行するタスクにも伝搬されます。トランザクションを直接接続することができる要求の場合、アプリケーション・コンテキストは URIMAP リソースに指定されている CICS トランザクションに設定されます。詳しくは、直接接続されたユーザー・トランザクションにより HTTP 要求が処理されるを参照してください。デフォルトの別名トランザクションは CWBA トランザクションです。
- HTTP/HTTPS 要求の URL が USAGE(PIPELINE) を指定された URIMAP リソースと一致する場合は、CWXXN タスクにアプリケーション・コンテキストが設定され、URIMAP リソースで指定されている CICS トランザクションを実行するタスクにも伝搬されます。トランザクションを直接接続することができる要求の場合、アプリケーション・コンテキストは URIMAP リソースに指定されている CICS トランザクションに設定されます。詳しくは、直接接続されたユーザー・トランザクションにより HTTP 要求が処理されるを参照してください。デフォルトのトランザクションは CPIH トランザクションです。

表 6. トランザクションとその初期プログラムの両方がエントリー・ポイント (EP) として定義されている場合のタスクのアプリケーション・コンテキスト・データ (ACD) 設定

トランザクションがアプリケーション・エントリー・ポイントとして定義されている	初期プログラムがアプリケーション・エントリー・ポイントとして定義されている	初期プログラムがパブリックまたはプライベートのどちらであるか	タスクの初期 ACD	タスクの現在の ACD
はい	はい	適用外	トランザクション・エントリー・ポイントによって設定	プログラム・エントリー・ポイントによって設定

表 6. トランザクションとその初期プログラムの両方がエントリー・ポイント (EP) として定義されている場合のタスクのアプリケーション・コンテキスト・データ (ACD) 設定 (続き)

トランザクションがアプリケーション・エントリー・ポイントとして定義されている	初期プログラムがアプリケーション・エントリー・ポイントとして定義されている	初期プログラムがパブリックまたはプライベートのどちらであるか	タスクの初期 ACD	タスクの現在の ACD
はい	いいえ	パブリック	トランザクション・エントリー・ポイントによって設定	空
はい	いいえ	プライベート	トランザクション・エントリー・ポイントによって設定	初期 ACD と同じ
いいえ	はい	適用外	プログラム・エントリー・ポイントによって設定	初期 ACD と同じ
いいえ	いいえ	適用外	未設定	未設定

表 7. URIMAP とその別名トランザクションの両方がエントリー・ポイント (EP) として定義されている場合のタスクのアプリケーション・コンテキスト・データ (ACD) 設定

URIMAP がアプリケーション・エントリー・ポイントとして定義されている	別名トランザクションがアプリケーション・エントリー・ポイントとして定義されている	初期プログラムがパブリックまたはプライベートのどちらであるか	タスクの初期 ACD	タスクの現在の ACD
はい	はい	パブリック	トランザクション・エントリー・ポイントによって設定	空
はい	はい	プライベート	トランザクション・エントリー・ポイントによって設定	初期 ACD と同じ
はい	いいえ	パブリック	URIMAP エントリー・ポイントによって設定	空
はい	いいえ	プライベート	URIMAP エントリー・ポイントによって設定	初期 ACD と同じ
いいえ	はい	パブリック	TRANSACTION エントリー・ポイントによって設定	空
いいえ	はい	プライベート	TRANSACTION エントリー・ポイントによって設定	初期 ACD と同じ
いいえ	いいえ	適用外	未設定	未設定

注: URIMAP の別名トランザクションの初期プログラムをアプリケーションのエントリー・ポイントとして設定することはできません。これは、その初期プログラムが DFH で始まり、エントリー・ポイントとして定義できないためです。このため、表 7 のすべてのプログラムはエントリー・ポイントとして定義されません。

表 8. トランザクションが EXEC CICS START または EXEC CICS RUN TRANSID によって接続される場合のタスクのアプリケーション・コンテキスト・データ (ACD) 設定

開始されるトランザクションがアプリケーション・エントリー・ポイントとして定義されている	開始されるトランザクションの初期プログラムがアプリケーション・エントリー・ポイントとして定義されている	初期プログラムがパブリックまたはプライベートのどちらであるか	開始タスクに ACD セットがある	タスクの初期 ACD	タスクの現在の ACD
はい	はい	適用外	適用外	TRANSACTION エントリー・ポイントによって設定	PROGRAM エントリー・ポイントによって設定
はい	いいえ	パブリック	適用外	TRANSACTION エントリー・ポイントによって設定	空
はい	いいえ	プライベート	適用外	TRANSACTION エントリー・ポイントによって設定	初期 ACD と同じ
いいえ	はい	適用外	適用外	PROGRAM エントリー・ポイントによって設定	初期 ACD と同じ
いいえ	いいえ	適用外	はい	開始タスクから ACD を継承	空
いいえ	いいえ	適用外	適用外	未設定	未設定

表 9. トランザクションが DYNAMIC として定義されている場合のタスクのアプリケーション・コンテキスト・データ (ACD) 設定

トランザクションがアプリケーション・エントリー・ポイントとして定義されている	開始タスクに ACD セットがある	ルーティング・プログラムの COMMAREA に渡される ACD
はい	はい	トランザクションのエントリー・ポイント
はい	いいえ	トランザクションのエントリー・ポイント
いいえ	はい	開始タスクの現在の ACD
いいえ	いいえ	ACD なし

アプリケーションの状態

CICS Explorer には、アプリケーションの状態情報が表示されます。アプリケーション・バージョンの状況情報は、アプリケーションの CICS バンドルが存在するかどうか、使用可能になっているかどうか、プラットフォーム内の領域タイプに関連付けられた CICS 領域で選択可能かどうかを示します。アプリケーション・バージョンの状況情報は、そのアプリケーション・バージョンの個々の管理パートの状況から導き出されます。管理パートは、アプリケーションのインストール処理中に自動的に作成される MGMTPART レコードです。管理パートは、アプリケーションとインストールされた各 CICS バンドルの間の関係、およびプラットフォームの中で各 CICS バンドルがインストールされた領域タイプを記録します。

CICS Explorer 製品資料内の『Checking the status of an application』の CICS Explorer の説明に従ってください。以下の図に、アプリケーションと関連する管理パートの状況を表す値を示します。

図 13 は、アプリケーションのライフサイクルのプロビジョニング段階で実行するアクションと、各シチュエーションに該当する一時的状況、使用可能化状況、選択可能性状況を、可能性のあるエラー状態と共に示しています。エラー状態の処理について詳しくは、プラットフォーム・エラーの診断を参照してください。

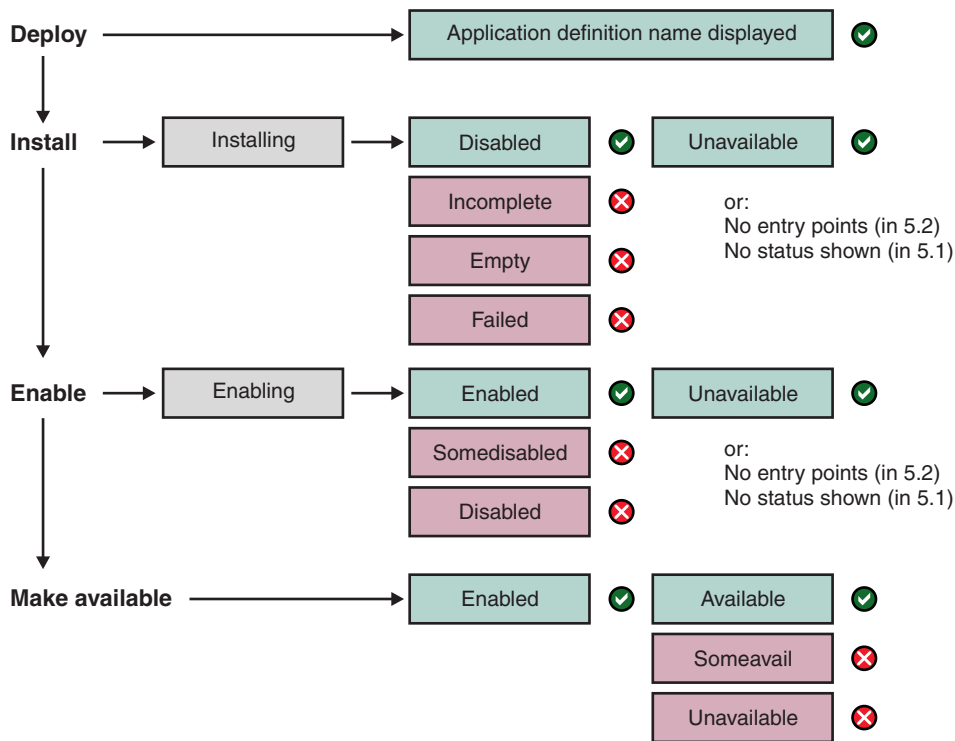


図 13. アプリケーション・ライフサイクル: プロビジョニング

- CICS アプリケーション・プロジェクトをデプロイします。そのためには、zFS 上のプラットフォームのホーム・ディレクトリーに CICS アプリケーション・プロジェクトをエクスポートして、CMAS のデータ・リポジトリー内にアプリケーション・バージョンの APPLDEF リソース定義を作成します。このプロセスを完了すると、アプリケーション・バージョンのアプリケーション定義の名前が「クラウド・エクスプローラー」ビューに表示されます。
- アプリケーションをインストールするとき、インストールの進行中は「クラウド・エクスプローラー」ビューに状況 **INSTALLING** が表示されます。インストールが完了したときの予期されるアプリケーション状況は、**DISABLED**、**UNAVAILABLE** です。アプリケーションにアプリケーション・エントリー・ポイントがない場合、そのことが「クラウド・エクスプローラー」ビューに選択可能性状況として表示されます。アプリケーションが CICS TS 5.1 領域にインストールされている場合、選択可能性状況は表示されません。

- アプリケーションを使用可能にするとき、使用可能化の進行中は「クラウド・エクスプローラー」ビューに状況 **ENABLING** が表示されます。使用可能化が完了したときの予期されるアプリケーション状況は、**ENABLED**、**UNAVAILABLE** です。
- SOMEDISABLED** 状態のアプリケーションを必要に応じて選択可能にすることができますが、まだ使用可能になっていないバンドルのアプリケーション・エントリー・ポイントは、このアクション時に選択可能になりません。
- 使用可能状況が **SOMEDISABLED** のアプリケーションを選択可能にすることができます。ただし、注意すべき要素が 2 つあります。使用可能状況が **SOMEDISABLED** の場合は、領域を始動または再始動しても、アプリケーションの選択可能性状況は復元されません。また、**DFHDPLOY** ユーティリティを使用してターゲット状態 **AVAILABLE** へのアプリケーションのデプロイメントを自動化することはできません。詳しくは、**DFHDPLOY** ユーティリティによる **CICS** アプリケーションのデプロイメントおよびアンデプロイメントの自動化を参照してください。
- アプリケーションを選択可能にすると、アプリケーションの状況は **ENABLED**、**AVAILABLE** に変わることが予想されます。アプリケーションにアプリケーション入り口点がないこと、またはアプリケーションが **CICS TS 5.1** 領域にインストールされていることをアプリケーションの状況が示している場合、「選択可能にする」アクションは不要です。

図 14 は、アプリケーションのライフサイクルのプロビジョン解除段階で実行するアクションと、各シチュエーションに該当する一時的状況、使用可能化状況、選択可能性状況を、可能性のあるエラー状態と共に示しています。

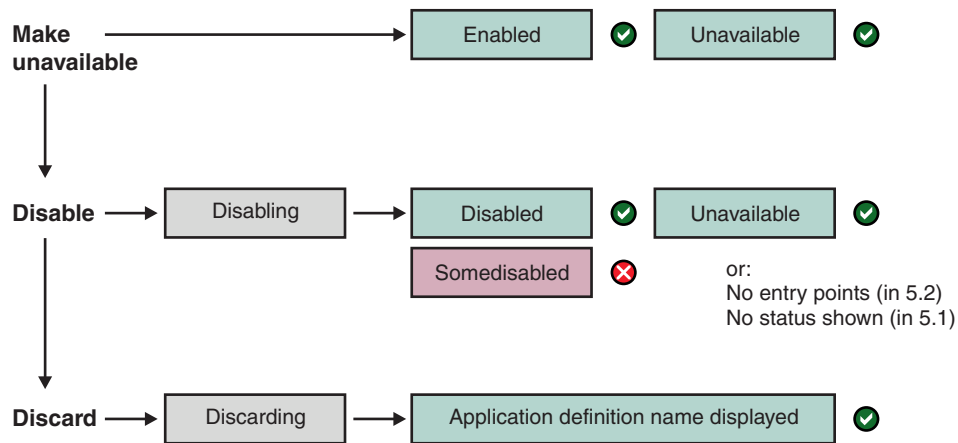


図 14. アプリケーション・ライフサイクル: プロビジョン解除

- アプリケーションを選択不可にすると、アプリケーションの状況は **ENABLED**、**UNAVAILABLE** に変わります。アプリケーションにアプリケーション入り口点がないこと、またはアプリケーションが **CICS TS 5.1** 領域にインストールされていることをアプリケーションの状況が示している場合、「選択不可にする」アクションは不要です。
- アプリケーションを使用不可にするとき、使用不可化の進行中は「クラウド・エクスプローラー」ビューに状況 **DISABLING** が表示されます。使用不可化が完

了したときの予期されるアプリケーション状況は、DISABLED、UNAVAILABLEです。あるいは、アプリケーションにアプリケーション・エントリー・ポイントがないこと、またはアプリケーションが CICS TS 5.1 領域にインストールされていることが状況で示される場合があります。

- アプリケーションを破棄するとき、破棄の進行中は「クラウド・エクスプローラー」ビューに状況 DISCARDING が表示されます。破棄が完了すると、「クラウド・エクスプローラー」ビューにアプリケーション名が表示されなくなります。代わりに、アプリケーション・バージョンのアプリケーション定義の名前が表示されます。

表 10. アプリケーションの状態

状況値	意味
AVAILABLE	アプリケーション・バージョンはそのアプリケーション・エントリー・ポイントを介して呼び出し元で選択できるようになっています。
DISABLED	アプリケーション・バージョンのすべての管理パートは使用不可です。
DISABLING	アプリケーション・バージョンのすべての管理パートは、使用不可にするための処理中です。
DISCARDING	アプリケーション・バージョンは破棄中なので、この時点で使用可能や使用不可にすることはできません。
EMPTY	アプリケーション・バージョンの管理パートはインストールされていません。
ENABLED	アプリケーション・バージョンのすべての管理パートは使用可能です。
ENABLING	アプリケーション・バージョンのすべての管理パートは、使用可能にするための処理中です。
FAILED	アプリケーション・バージョンのインストール中または破棄中に問題が発生しました。
INCOMPLETE	アプリケーション・バージョンの一部の管理パートが空であるか、無効なスコープがあります。
INSTALLING	アプリケーション・バージョンはインストール中なので、この時点で使用可能や使用不可にすることはできません。
NONE (・エントリー・ポイントなし)	アプリケーション・バージョンにアプリケーション・エントリー・ポイントがありません。
SOMEAVAIL	アプリケーション・バージョンに対して「選択可能にする」または「選択不可にする」アクションが実行されましたが、アプリケーション・エントリー・ポイントの中には選択可能なものと選択不可のものがあります。
SOMEDISABLED	アプリケーション・バージョンの一部の管理パートは使用不可です。
UNAVAILABLE	アプリケーション・バージョンは呼び出し元に対して選択不可と設定されています。

表 11. 管理パートの状況値

状況値	意味
AVAILABLE	CICS バンドルで宣言されたアプリケーション・エントリー・ポイントは呼び出し元で選択できるようになっています。

表 11. 管理パートの状況値 (続き)

状況値	意味
DISABLED	CICS バンドルは、すべての CICS 領域で使用不可です。
DISABLING	CICS バンドルは、使用不可にするための処理中です。
EMPTY	CICS バンドルは、いずれの CICS 領域にもインストールされません。
ENABLED	CICS バンドルは、すべての CICS 領域にインストールされていて使用可能です。
ENABLING	CICS バンドルは、使用可能にするための処理中です。
IMPORTONLY	CICS バンドルは、すべての CICS 領域にインストールされていて使用可能ですが、インポート・ステートメントしか含まれていないため、アプリケーションの状況には影響しません。
INCOMPLETE	CICS バンドルは、いくつかの (ただし全部ではない) CICS 領域にインストールされています。
INVALIDSCOPE	CICS バンドルをインストールするために指定された CICS システム・グループが存在しないため、CICS バンドルはインストールされません。
NONE (エントリー・ポイントなし)	CICS バンドルにアプリケーション・エントリー・ポイントのステートメントが含まれていません。
SOMEAVAIL	CICS バンドルに対して「選択可能にする」または「選択不可にする」アクションが実行されましたが、アプリケーション・エントリー・ポイントの中には選択可能なものと選択不可のものがあります。
SOMEDISABLED	CICS バンドルは、一部の CICS 領域で使用不可です。この状況は、インストールされているいずれかの BUNDLE リソースの ENABLEDCOUNT 値が 0 より大きい (CICS バンドルによって作成された 1 つ以上のリソース、アプリケーション・エントリー・ポイント、またはポリシー・スコープが CICS 領域で現在使用可能になっていることを示す) 場合に、すべての CICS 領域で CICS バンドルが使用不可になっているときにも発生します。
UNAVAILABLE	CICS バンドルで宣言されたアプリケーション・エントリー・ポイントは呼び出し元に対して選択不可と設定されています。

仕組み: 複数バージョン管理のアプリケーション

複数のバージョンのアプリケーションを同時に同じプラットフォームのインスタンスにインストールして管理できます。複数バージョン管理により、以前のバージョンを無効にしたり除去したりしなくても、アプリケーションの新しいバージョンをプラットフォームにデプロイして、サービスを中断せずにユーザーが使用できるようことができます。複数バージョン管理を使用すれば、アプリケーションの異なるバージョンを異なるユーザーが起動することができ、新規バージョンにエラーがあると判明した場合に、安定したバージョンのアプリケーションに簡単に切り替えることができます。実行時には、プログラムは使用可能な任意のバージョンのアプリケーションを起動できます。複数バージョン管理を認識しないプログラムは、プラットフォームで使用可能になっている最上位のアプリケーション・バージョンに自動的にリンクされます。

同じアプリケーションの 2 つのバージョンを同時にホストして、異なるユーザー・グループが使用できるようにすることも、アプリケーションの 1 つのバージョンを別のバージョンに置き換えて、すべてのユーザーが使用できるようにすることもできます。概要については、62 ページの『例: アプリケーションの 2 つのバージョンを同時にホストする』および 57 ページの『例: アプリケーションのバージョンの置換』を参照してください。リソースは、アプリケーションの各バージョン専用にすることができます。

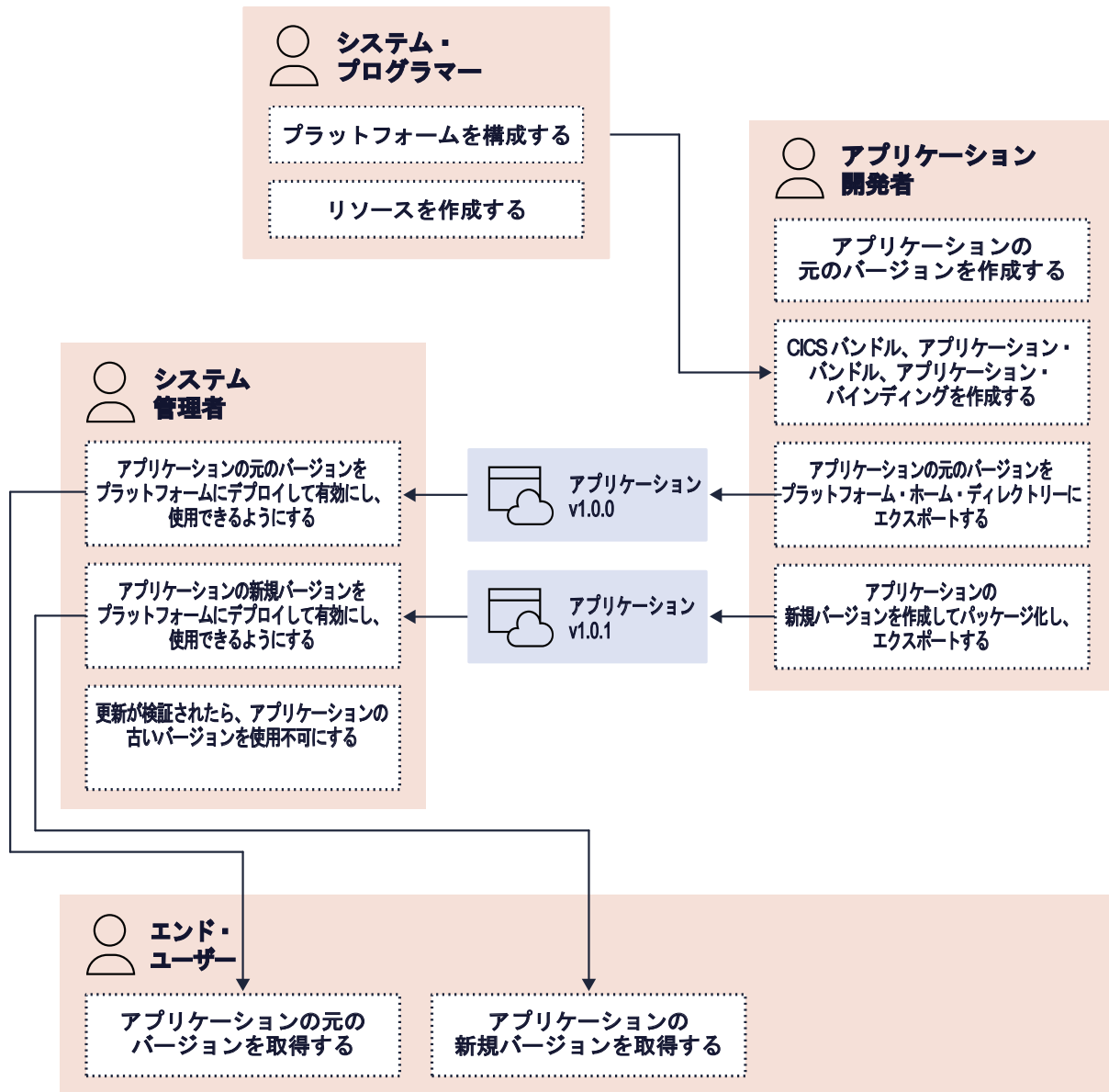


図 15. クラウド・ソリューションでのアプリケーションの複数バージョンの使用

リソースをアプリケーションの各バージョンに専用のものとして扱う

アプリケーションで、ユーザーはライフサイクル管理のためにキー・リソースをアプリケーションにパッケージ化します。これらのキー・リソースには、アプリケーション・エントリー・ポイントおよびアプリケーション・バージョン固有のロード・ライブラリーとして識別されるプログラムが含まれます。CICS TS リソース定義を CICS TS アプリケーションに取り込むことによって、ライフサイクル全体を通してアプリケーションをより適切に管理できるようになります。アプリケーション開発中には、単一の開発環境でリソースを記述し、アプリケーション開発者が編集できるようにすると役立ちます。アプリケーションがソース・コード管理の対象である場合は、アプリケーション・デプロイメントを通じて、関連するすべての成果物を網羅し、追跡できます。アプリケーションは、単一制御ポイントを使用してプロビジョンおよびプロビジョン解除できますが、関連するすべての CICS TS リソースの状態はシステムによって管理されます。

従来、アプリケーションは、キー CICS TS リソースの提供を CSD などのアプリケーション外部の機構に依存していました。しかし、リソースが CSD または CICSplex SM ビジネス・アプリケーション・サービス (BAS) によって提供されている場合、1 つのアプリケーションの異なるバージョンのアプリケーション・エントリー・ポイントでは、リソース名を共用できません。複数バージョン管理を使用すれば、アプリケーションの複数のバージョンが、特定の CICS TS リソースを同じ名前宣言できます。各リソースは、アプリケーションのバージョンに固有かつ専用のものとして扱われます。

サポートされるリソース・タイプにおいては、CICS リソースが、アプリケーションの一部として (つまり、アプリケーション・バンドルの一部として、またはアプリケーション・バインディングの一部として) パッケージされてインストールされる CICS バンドルで定義される場合、専用リソースになります。この方法で CICS リソースを作成した場合、そのリソースは、プラットフォームにインストールされている他のアプリケーションやバージョンからは使用できず、CICS 領域内の他のアプリケーションからも使用できません。そのリソースが定義されたバージョンのアプリケーションからのみ使用できます。このようなリソースは、専用リソース と呼ばれます。詳しくは、アプリケーション・バージョンの専用リソースを参照してください。

以下のリソースは、複数バージョンに対応したアプリケーションの一部としてサポートされています。

- アプリケーションの一部である CICS バンドルで定義された PROGRAM リソース。
- アプリケーションの一部である CICS バンドルで定義された LIBRARY リソース。
- アプリケーションの一部である CICS バンドルで定義された PACKAGESET リソース。
- POLICY リソース
- アプリケーション・エントリー・ポイントのステートメント
- アプリケーションの依存関係 (またはバンドル・インポート) として定義されたすべてのリソース。

その他のリソースは、複数バージョン対応のアプリケーションで使用できます (アプリケーションの異なるバージョン間でリソース名の競合がないようにリソースを管理している場合)。例えば、アプリケーションの一部である URIMAP リソースは、アプリケーションの新しいバージョンをパッケージ化およびインストールするときに、名前変更できます。または、複数バージョン管理がサポートされないリソースをアプリケーションの外部で管理しながら、アプリケーションの依存関係 (またはバンドル・インポート) として宣言するように、アプリケーションを設計できます。JVMSERVER リソースなど、異なるアプリケーションによって現に使用されているか使用される可能性のあるリソースの場合、プラットフォームのレベルでリソースをデプロイおよび管理します (つまり、プラットフォームにデプロイされるすべてのアプリケーションのすべてのバージョンでそのリソースを使用できます)。

アプリケーションの単一のバージョンで行う場合と同様に、アプリケーション・エントリー・ポイントを宣言することによって、複数バージョン対応アプリケーションのリソースへのユーザー・アクセスを制御します。

EXEC CICS INVOKE APPLICATION コマンドを使用することによって、複数バージョンに対応したアプリケーションの特定のバージョンを呼び出すことができます。アプリケーションの特定のバージョンの呼び出しについて詳しくは、複数バージョンに対応するアプリケーションの呼び出しを参照してください。

例: アプリケーションのバージョンの置換

企業の CICS アプリケーションに、ユーザーが CICS トランザクションを使用してアクセスするシナリオを考えます。システム管理者は、アプリケーションのバグを修正するために、アプリケーションの新しいバージョンをデプロイする必要がある、すべてのユーザーが同時に新しいバージョンの使用に移行することを期待しています。

システム管理者またはアプリケーション開発者は、CICS Explorer を使用して、アプリケーションの新しいバージョンをパッケージ化してエクスポートします。システム管理者は、アプリケーションの新しいバージョンをプラットフォームにデプロイし、インストールされたアプリケーションを有効化します。アプリケーションが使用可能な状態になると、システム管理者は、インストールが成功したことが分かります。この時点では、ユーザーが CICS トランザクションを開始したとき、アプリケーションの古いバージョンが引き続き提供されます。システム管理者がアプリケーションの新しいバージョンを使用可能にすると、CICS によって、即時に、そのバージョンが古いバージョンの代わりにユーザーに提供されます。ユーザーは以前と同じ CICS トランザクションを開始します。ほかにユーザーが行う必要のあるアクションはありません。

アプリケーションの新しいバージョンに問題がある場合、ユーザーはアプリケーションの古いバージョンに素早くロールバックできます。デフォルトでは、CICS は、プラットフォームで使用可能になっている最上位のアプリケーション・バージョンを呼び出し元 (CICS トランザクションやその他のリンク・アプリケーション) に提供します。システム管理者が古いバージョンを使用可能にしたか、または使用可能なまま保持しているときに、新しいバージョンを使用不可にすると、ユーザーには自動的に古いバージョンが再度提供されます。アプリケーションへのユーザー・アクセスは、アプリケーション内のリソースに対して宣言されたアプリケーション・エントリー・ポイントによって制御されます。これは、リソースを使用可能または

使用不可に設定できます。ユーザーによるアクセスを防ぐために、システム管理者がアプリケーションの代替バージョンを使用不可にしたり、破棄したりする必要はありません。

このシナリオでは、アプリケーションの古いバージョンに問題があるため、新しいバージョンの検証後に無効化するか、破棄することが適切です。ただし、プラットフォームにデプロイされたアプリケーションの複数のバージョンを、ユーザーに対して同時に使用可能なままにしておくことができます。複数バージョン管理を認識するようにコーディングされているプログラムは、**EXEC CICS INVOKE APPLICATION** コマンドを使用して、アプリケーションの特定のメジャー・バージョンまたはマイナー・バージョンにアクセスできます。

59 ページの図 16 に、CICS Explorer のプロジェクトを使用した、プラットフォームの定義およびアプリケーションのパッケージ化を示します。サンプル・アプリケーションには、LIBRARY リソースおよび PROGA と PROGB という名前の 2 つの PROGRAM リソースの定義を含む CICS バンドルが含まれています。アプリケーションの TRANSACTION リソースは CICS バンドルで定義されていますが、アプリケーションと一緒にデプロイされません。プラットフォーム・プロジェクト、アプリケーション・プロジェクト、アプリケーション・バインディング・プロジェクト、および CICS バンドル・プロジェクトは、zFS プラットフォーム・ホーム・ディレクトリー内のサブディレクトリーにエクスポートされます。プラットフォームおよびアプリケーションに対してプラットフォーム定義 (PLATDEF) およびアプリケーション定義 (APPLDEF) が作成され、CICSplex SM データ・リポジトリに格納されます。プラットフォーム定義は、ターゲット CICS 領域を含む領域タイプでプラットフォームを作成するために、CICSplex にインストールされます。アプリケーション定義は、CICS 領域にアプリケーションのバージョン 1.0.0 用の CICS バンドルを作成するために、プラットフォームにインストールされます。CICS バンドルからの情報を使用して、アプリケーションの LIBRARY と PROGRAM の各リソースが CICS 領域に動的に生成されます。LIBRARY リソースは、アプリケーションのバージョン 1.0.0 用のプログラムを含む、z/OS 上のデータ・セットをポイントします。TRANSACTION リソースの定義を含む CICS バンドルもプラットフォームに追加され、CICS 領域に TRANSACTION リソースが動的に生成されます。

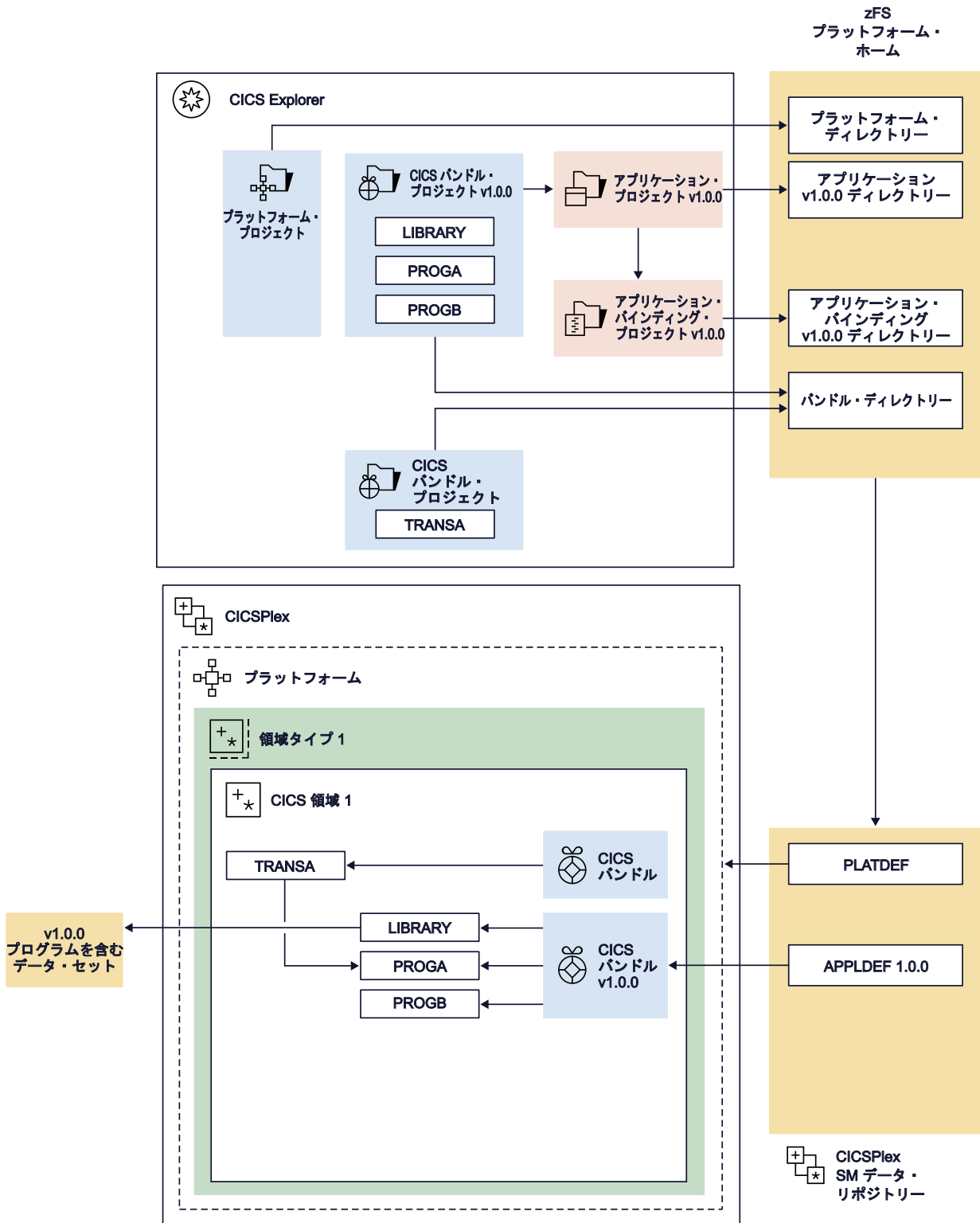


図 16. アプリケーションのバージョン 1.0.0 のサンプル・トポロジー

61 ページの図 17 は、アプリケーションのバージョン 1.0.1 では、アプリケーションの新しいバージョンのプログラムを含むデータ・セットをポイントするように、CICS バンドル内の **LIBRARY** リソース定義が更新されることを示しています。

す。アプリケーションの他のリソース定義は変更されません。CICS トランザクションのリソースを含む CICS バンドル・プロジェクトは個別にデプロイされるため、これも更新する必要はありません。

CICS Explorer を使用して、アプリケーションの新しいバージョンをパッケージ化します。新しいアプリケーション・バージョンには、アプリケーションのリソースを定義する CICS バンドル・プロジェクトのバージョン 1.0.1 が含まれています。アプリケーションおよびアプリケーション・バインディングの新しいバージョン用に新規サブディレクトリーが作成されますが、CICS バンドルは同じバンドル・サブディレクトリーに格納されます。アプリケーションのバージョン 1.0.1 に対してアプリケーション定義 (APPLDEF) が作成され、CICSplex SM データ・リポジトリに格納されます。APPLDEF リソースの名前は、古いバージョンの APPLDEF リソースに使用されたものと同じ名前ですが、バージョン番号は、アプリケーションの新しいバージョンと一致するように変更されます。CICS 領域にアプリケーションのバージョン 1.0.1 用の CICS バンドルおよびリソースを作成するために、新しいアプリケーション定義がプラットフォームにインストールされます。アプリケーションのバージョン 1.0.0 用の CICS バンドルとリソースは CICS 領域にインストールされたままですが、TRANSACTION リソース (更新されていません) はアプリケーションのバージョン 1.0.1 を自動的にポイントするようになります。

これで CICS 領域は、アプリケーションの新しいバージョンをホストするようになりました。アプリケーションの新しいバージョン用のリソースを含む CICS バンドルがインストールされると、アプリケーションの新しいバージョン専用の LIBRARY と PROGRAM の各リソースが CICS 領域に動的に作成されます。アプリケーションのプログラムの新しいバージョンは、z/OS® 上の異なる区分データ・セット (PDS) または拡張区分データ・セット (PDSE) に格納されます。アプリケーションの新しいバージョン用の更新された LIBRARY リソースは、新規データ・セットを参照します。アプリケーションを実行する CICS トランザクションは、更新プロセスには関与せず、変更されません。ユーザーがトランザクションを実行すると、CICS によって、自動的に、利用可能なアプリケーションのバージョンのうちで最上位のものがユーザーに提供されます。つまり、ここではアプリケーションのバージョン 1.0.1 のプログラムが実行されます。

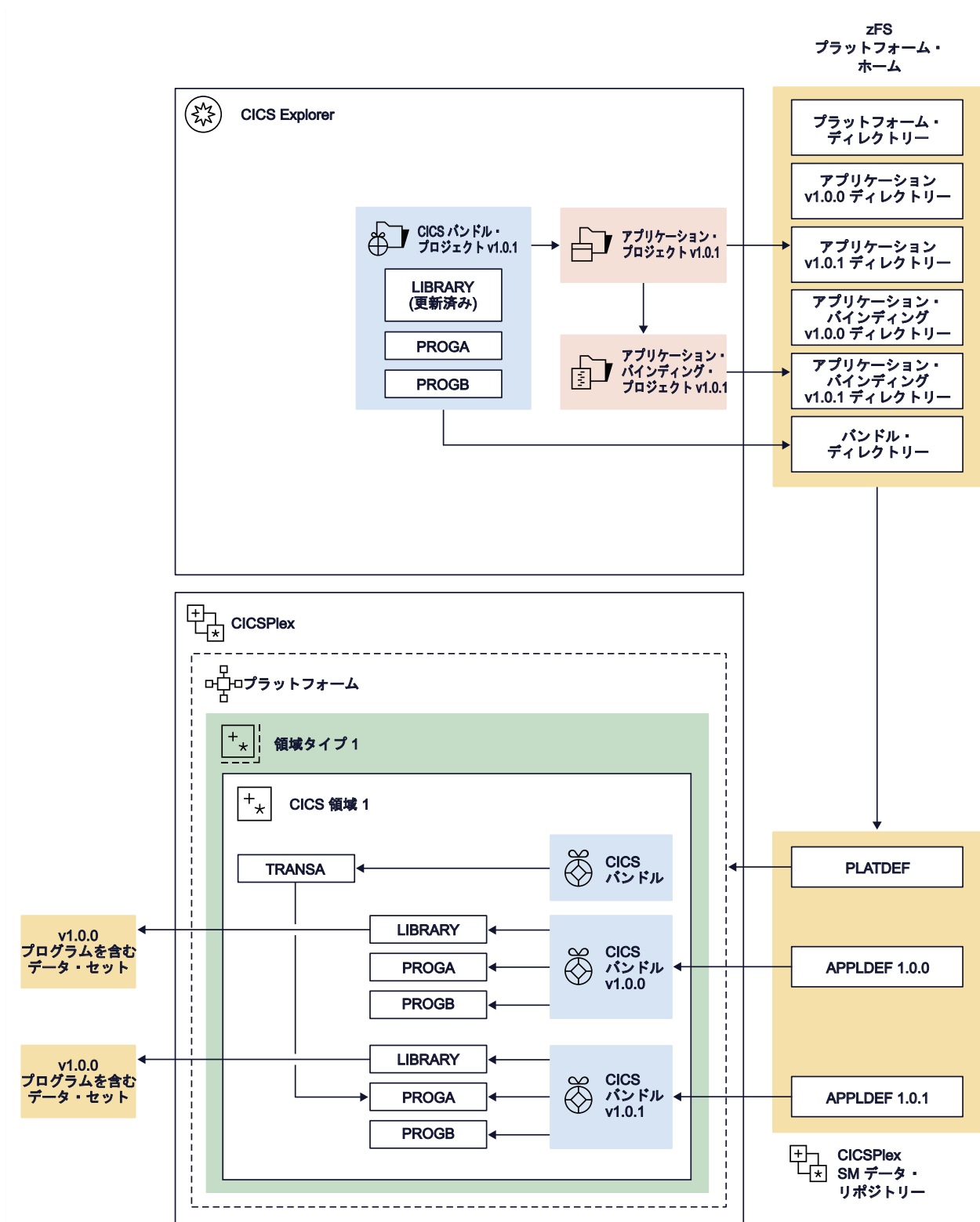


図 17. 複数バージョン対応アプリケーションのサンプル・トポロジー。すべてのユーザーに対し、アプリケーションの前のバージョンが別のバージョンで置き換えられる

例: アプリケーションの 2 つのバージョンを同時にホストする

企業が CICS アプリケーションを所有するシナリオを考えます。このアプリケーションは、ブラウザー・インターフェースを介して Web アプリケーションとしてユーザーに提示され、Liberty JVM サーバーでホストされるプレゼンテーション・ロジックを持ちます。この企業は、アプリケーションの新しいバージョンの導入を予定しています。アプリケーションの変更は大幅なので、ビジネスの中断を避けるために、この企業はすべてのユーザーが同時に新しいバージョンに移行することを望んでいません。

システム管理者は、プラットフォームの一部である同じ CICS 領域に、アプリケーションの両方のバージョンをデプロイして、使用可能にします。異なる URL を使用することによって、ユーザーは、アプリケーションの新しいバージョンまたは古いバージョンに誘導されます。

63 ページの図 18 に、アプリケーションのバージョン 1.0.0 での、CICS Explorer のプロジェクトを使用したプラットフォームの定義およびアプリケーションのパッケージ化を示します。アプリケーションには、エンタープライズ・バンドル・アーカイブ (EBA) に格納されている OSGi アプリケーション・プロジェクトとして Web アプリケーションをパッケージ化する CICS バンドル・プロジェクトが含まれています。アプリケーションの URIMAP リソースが INDEX という名前で CICS バンドル内に作成され、アプリケーション・バインディングとともにデプロイされます。Liberty JVM サーバーも、CICS バンドル・プロジェクトを使用して定義されます。プロジェクトは zFS プラットフォーム・ホーム・ディレクトリー内のサブディレクトリーにエクスポートされます。プラットフォームおよびアプリケーションに対してプラットフォーム定義 (PLATDEF) およびアプリケーション定義 (APPLDEF) が作成され、CICSplex SM データ・リポジトリーに格納されます。プラットフォーム定義は、ターゲット CICS 領域を含む領域タイプでプラットフォームを作成するために、CICSplex にインストールされます。アプリケーション定義は、CICS 領域にアプリケーションのバージョン 1.0.0 用の CICS バンドルを作成するために、プラットフォームにインストールされます。CICS バンドルからの情報を使用して、アプリケーションのバージョン 1.0.0 用の URIMAP リソースが、INDEX という名前で CICS 領域に動的に生成され、Web アプリケーション用のリソースが Liberty JVM サーバーにデプロイされます。

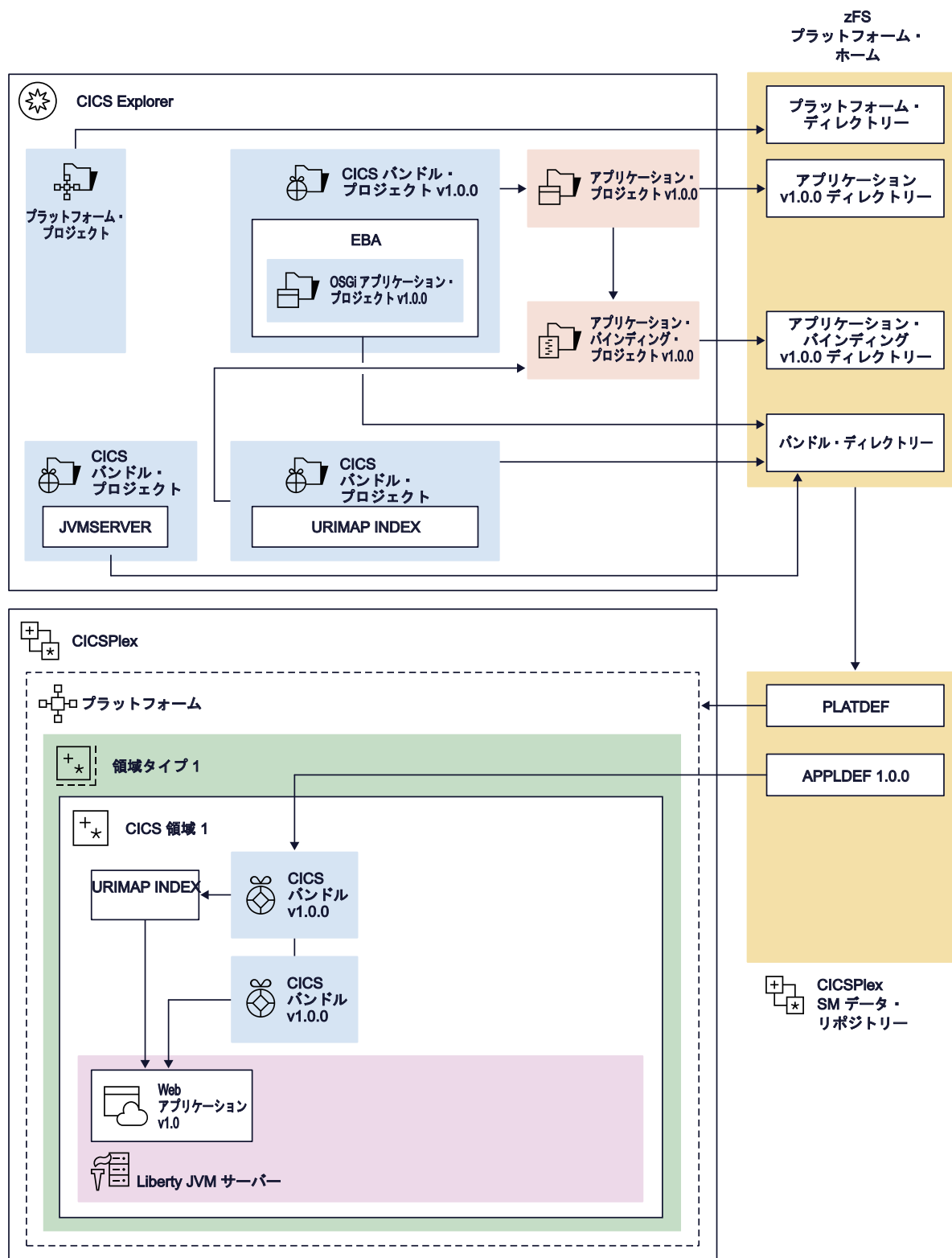


図 18. アプリケーションのバージョン 1.0.0 のサンプル・トポロジー

65 ページの図 19 に、アプリケーションのバージョン 1.1.0 での、CICS Explorer を使用したアプリケーションのパッケージ化を示します。新しいアプリケーション

ン・バージョンには、Web アプリケーションをパッケージ化する CICS バンドル・プロジェクトのバージョン 1.1.0 が含まれ、OSGi アプリケーション・プロジェクトもバージョンが 1.1.0 に変更されます。アプリケーションの新規バージョン用の新しい URIMAP リソースが CICS バンドル内に作成され、新しいバージョンのアプリケーション・バインディングとともにデプロイされます。プロジェクトは zFS プラットフォーム・ホーム・ディレクトリー内のサブディレクトリーにエクスポートされます。アプリケーションおよびアプリケーション・バインディングの新しいバージョン用に新規サブディレクトリーが作成されますが、CICS バンドルは同じ bundles サブディレクトリーに格納されます。アプリケーションのバージョン 1.1.0 に対してアプリケーション定義 (APPLDEF) が作成され、CICSplex SM データ・リポジトリに格納されます。APPLDEF リソースの名前は、古いバージョンの APPLDEF リソースに使用されたものと同じ名前ですが、バージョン番号は、アプリケーションの新しいバージョンと一致するように変更されます。CICS 領域に、アプリケーションのバージョン 1.0.0 用の CICS バンドルとリソースに加え、アプリケーションのバージョン 1.1.0 用の CICS バンドルとリソースを作成するために、新しいアプリケーション定義がプラットフォームにインストールされます。

CICS 領域は、アプリケーションの新しいバージョンのワークロードだけでなく、そのアプリケーションの古いバージョンのワークロードも処理します。アプリケーションのバージョン 1.1.0 用の URIMAP リソースが、INDEX11 という名前で CICS 領域に動的に生成され、Web アプリケーションのバージョン 1.1.0 用のリソースが Liberty JVM サーバーにデプロイされます。アプリケーションの元のバージョン用の CICS バンドルは CICS 領域にインストールされたままで、INDEX という名前の URIMAP リソースおよび Web アプリケーションのバージョン 1.0.0 用のリソースも引き続き使用可能です。したがって、アプリケーションの両方のバージョンが CICS 領域内に存在し、使用可能で、異なる URIMAP リソースで指定された異なる URL によってアクセスされます。

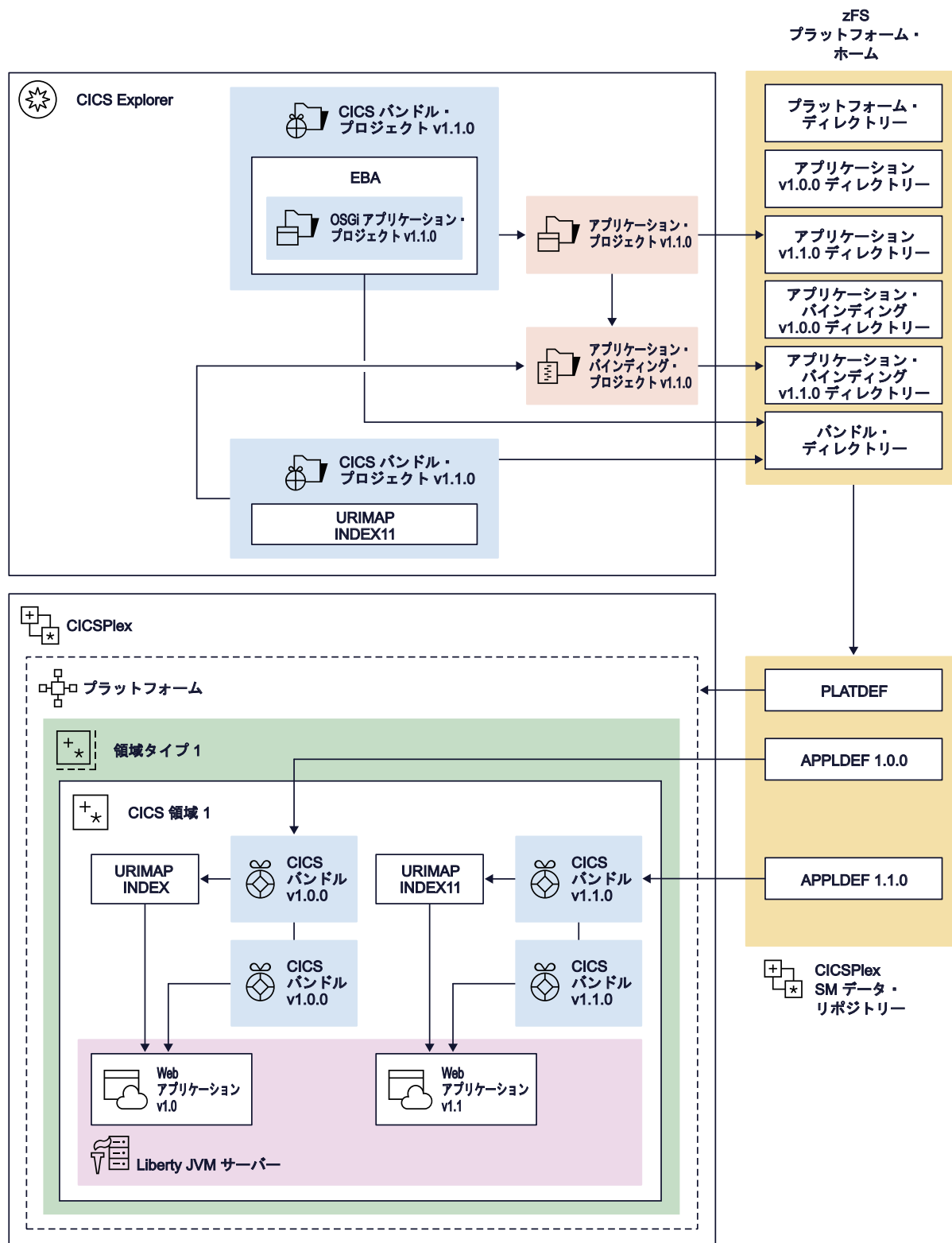


図 19. 複数バージョン対応アプリケーションのサンプル・トポロジー。アプリケーションのあるバージョンが別のバージョンと同時に実行される

アプリケーションへのバージョンの割り当て

CICS TS では、特定のリソースにバージョンが関連付けられています。例えば、クラウド使用可能化のコンテキスト内のアプリケーションには、バージョンと名前が必要です。CICS アプリケーションまたは CICS プラットフォームいずれかの一部としてインストールされる CICS バンドルにも、バージョンを指定する必要があります。CICS アプリケーション・バインディングにもバージョンがあります。これらのバージョンは、CICS 製品のバージョン (バージョン 5 など) とは異なります。このバージョンが必要な理由および使用方法について説明します。

CICS 環境での更新をデプロイおよび管理するために、CICS バンドル、アプリケーション・バンドル、およびアプリケーション・バインディングにバージョン管理ポリシーを適用する必要があります。アプリケーション・バンドルの既存のバージョンを使用して、アプリケーションの CICS バンドルの新しいバージョンをインストールすることはできません。また、アプリケーション・バインディングの既存のバージョンをアプリケーション・バンドルの新しいバージョンとともに使用することはできません。アプリケーションの CICS バンドルを更新した場合は、必ずアプリケーション・バンドルとアプリケーション・バインディングのバージョンを更新する必要があります。

クラウドを使用可能にするために、CICS では、セマンティック・バージョン管理 (技術資料 Semantic Versioning を参照) に基づいてシステムをバージョン管理することをお勧めしています。このバージョン管理システムは、Eclipse などの OSGi ベースのプロジェクトで広く使用され、CICS 用 Java クラス・ライブラリー (JCICS) API を備えた CICS V4.2 で初めて採用されました。このシステムを利用すると、API またはサービスの作成者が実装の変更の内容を明確に記述できるため、お客様は以前のバージョンとの互換性の問題を理解することができます。

CICS TS アプリケーション、CICS バンドル、または OSGi バンドルに使用されるバージョン属性は、`<major>.<minor>.<micro>` という形式になります (例: 2.3.1)。バージョンは、アプリケーション開発者など、対象のリソースの保守担当者が手動で更新する必要があります。バージョンは、新規アクティビティーの一部として行う最初の変更である必要があります。

バージョンの各部分の変更は、以下のように使用します。

マイクロ

例えば、1.0.0 から 1.0.1 です。外部機能の変更 (バグ修正など) は行われません。

マイナー

例えば、1.0.1 から 1.100.0 です。以前のバージョンまたは外部機能の変更と互換性があります (例えば、既存のクライアントは影響を受けません)。

マイナー・バージョンは、1 ずつではなく 100 ずつ増分することをお勧めします。この手法は、複数のバージョンのアプリケーションを同時に実行する場合に役立ちます。例えば、実動環境でアプリケーションのバージョン 1.0.0 を実行し、バージョン 1.1.0 を追加するシナリオを考えます。しばらくの間は、両方のバージョンが問題なく同時に実行されます。その後、バージョン 1.0.0 にバグが見つかり、アプリケーションの外部機能に小さい変更が必要になります。外部機能が変更されるため、マイナー・バージョン番号を増分することが理想的ですが、V1.1.0 は既に使用されています。アプリ

ケーションの新しいバージョンを追加したときにバージョン 1.100.0 を使用していれば、V1.0.0 でバグを修正して、アプリケーションの外部機能への変更を表すために、マイナー・バージョン番号を増分できます。この手法は、CICS TS で、JCICS API に組み込まれる OSGi バンドルをバージョン管理するために使用されています。

メジャー

例えば、1.100.0 から 2.0.0 です。変更は以前のバージョンと互換性がありません (例えば、ファイル・レコードの形式が異なるか、操作が削除されています)。

セマンティック・バージョン管理の原則に従ってアプリケーションのバージョンを変更するときは、新しいバージョンは、アプリケーションに組み込まれる CICS バンドルでの最も大きい変更を反映する必要があります。例えば、アプリケーションの 1 つの CICS バンドルをバージョン 1.0.1 からバージョン 1.0.2 に変更 (これはマイクロ・バージョン変更) し、アプリケーションの別の CICS バンドルをバージョン 1.2.0 からバージョン 1.3.0 に変更 (これはマイナー・バージョン変更) するとします。これら 2 つの CICS バンドルを含むアプリケーション・バンドルには、マイナー・バージョン変更を含める必要があります。つまり、アプリケーションの前のバージョンが 2.5.1 だった場合、バージョン 2.6.0 に変更する必要があります。

第 4 章 アプリケーションの設計

以下の基本概念を使用して、CICS アプリケーションの設計に役立ててください。

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、103 ページの『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) 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あります。

関連タスク:

 CICS プログラミング・ロードマップ

タスクの開始方法

CICS で開始される作業、つまりタスクは、非送信請求入力から、または自動タスク開始 (ATI) によって開始されます。

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 (314 ページの『RETURN IMMEDIATE』で説明)、および SEND PAGE コマンド (517 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 (377 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 (533 ページの『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。

基本機能

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、440 ページの『CICS プリンターの使用』で説明しています。

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)。

代替機能

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 は実行するトランザクションをどのように決定するのでしょうか？ 通常、同じ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、TRANSID オプションを使用して、次にどのトランザクション CICS を実行するかを決定します。

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状態で、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。72 ページの図 20 (このロジックの図) を参照してください。

「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「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。この機能について詳しくは、447 ページの『表示画面の印刷』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、460 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。348 ページの『3270 端末からの入力』の『アテンション・キー』では、アテンション ID について説明しています。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文字の前の先頭の文字 (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。フィールド分離文字は、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、LINK コマンドの例外条件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。

75 ページの図 21は、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、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、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 21. 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 22 と図 23 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 22. 表示ロジック

```

..
EXEC CICS ADDRESS COMMAREA
..
EXEC CICS READ UPDATE
..
EXEC CICS REWRITE
..
EXEC CICS RETURN..

```

図 23. 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、248 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を複数のトランザクションで並行処理できるようにするための手法です。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。

この手法を、完了までプログラムを実行する単スレッドと比較してください。単スレッドでは、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、再入可能性は、プログラムが呼び出されるたびに作業用ストレージの最新コピーを取得することによって確保されます。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。これらの属性は、PROGRAM リソース定義に設定されます。次の表は、プログラムで利用可能な CONCURRENCY および API の設定と、アプリケーション・プログラムが実行される TCB のタイプを示します。

表 12. PROGRAM CONCURRENCY および API の設定と、使用される TCB のタイプの組み合わせ

CONCURRENCY 属性	API 属性	CICS TCB
CONCURRENCY(QUASIRENT)	API(CICSAPI)	アプリケーション・プログラムは、常に CICS 準再入可能 (QR TCB) の下で実行されます。リソース・マネージャーでオープン TCB への切り替えが発生した場合、CICS は常に QR TCB に戻ってからアプリケーションに戻ります。
CONCURRENCY(QUASIRENT)	API(OPENAPI)	無効な組み合わせです。OPENAPI プログラムは QR TCB 上で実行できません。
CONCURRENCY(THREADSAFE)	API(CICSAPI)	アプリケーション・プログラムは、QR TCB またはオープン TCB 上で実行できます。リソース・マネージャーでオープン TCB への切り替えが発生した場合、CICS はアプリケーションに戻るときにオープン TCB 上から移動しません。QR TCB への切り替えが発生した場合、CICS はアプリケーションに戻るときに QR TCB 上から移動しません。
CONCURRENCY(THREADSAFE)	API(OPENAPI)	CONCURRENCY(REQUIRED) API(OPENAPI) と同じです。

表 12. PROGRAM CONCURRENCY および API の設定と、使用される TCB のタイプの組み合わせ (続き)

CONCURRENCY 属性	API 属性	CICS TCB
CONCURRENCY(REQUIRED)	API(CICSAPI)	アプリケーション・プログラムは、常にオープン TCB 上で実行されます。CICS サービスは TCB キーの一致を必要としないため、アプリケーションは常に L8 のオープン TCB 上で実行されます。OPENAPI リソース・マネージャーは L8 TCB を使用するため、呼び出されたときに TCB 切り替えは必要ありません。QR TCB への切り替えが発生した場合、CICS はアプリケーションに戻るときにオープン TCB に戻ります。
CONCURRENCY(REQUIRED)	API(OPENAPI)	アプリケーション・プログラムは、常にオープン TCB 上で実行されます。TCB のキーは、プログラムの実行キーと一致している必要があります。CICS は、EXECKEY(USER) が設定されるときは L9 TCB を使用し、EXECKEY(CICS) が設定されるときは L8 TCB を使用します。アプリケーションがユーザー・キーであり、OPENAPI リソース・マネージャーが呼び出される場合、L9 TCB から L8 TCB への切り替えが発生します。CICS は、L9 TCB に戻ってからアプリケーションに戻ります。QR TCB への切り替えが発生した場合、CICS はアプリケーションに戻るときにオープン TCB に戻ります。

準再入可能なアプリケーション・プログラム

準再入可能なプログラムとは、入り口点および各 **EXEC CICS** コマンドの前後で制御権がそれに渡される際に、首尾一貫した状態であるプログラムのことです。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。

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。

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。例えば、ユーザー・アプリケーション・プログラムが 11 のユーザー・タスクで使用されている場合は、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。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、355 ページの『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ます。CICS は、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・トランザクション環境 (OTE) のオープン TCB 上で同時に実行できます。

オープン・トランザクション環境へのアクセス

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は、オープン・トランザクション環境に自動的に組み込まれます。これらのアプリケーションは、スレッド・セーフであることから良好なパフォーマンスを得ることができま

す。 Db2 リソースにアクセスする CICS アプリケーションにより使用される CICS Db2 タスク関連ユーザー出口は、オープン API TRUE であるため、CICS Db2 アプリケーションがスレッド・セーフであることから良好なパフォーマンスを得ることができます。CICS Db2 アプリケーションのスレッド・セーフ・プログラミングについて詳しくは、Enabling CICS Db2 applications to use the open transaction environment (OTE) through threadsafe programmingを参照してください。

その他のユーザー・アプリケーション・プログラム、PLT プログラム、ユーザー置き換え可能モジュール、またはタスク関連ユーザー出口の場合、OPENAPI プログラムとして定義することによって、オープン・トランザクション環境を使用することを選択できます。この場合、必ずスレッド・セーフである必要があります。OPENAPI プログラムのスレッド・セーフ・プログラミングについて詳しくは、87 ページの『OPENAPI プログラム』を参照してください。

プログラムを CONCURRENCY(REQUIRED) として定義する場合、常にオープン TCB 上で実行されます。使用されるオープン TCB のタイプは、API 設定に応じて決定されます。CICSAPI プログラムの場合、CICS はプログラムの実行キーに関係なく、L8 オープン TCB を使用します。OPENAPI プログラムの場合、CICS は EXECKEY(USER) が設定されるときは L9 TCB を使用し、EXECKEY(CICS) が設定されるときは L8 TCB を使用します。REQUIRED はユーザー・アプリケーション・プログラム、PLT プログラム、およびユーザー置き換え可能モジュールに適用できます。グローバル・ユーザー出口プログラムおよびタスク関連ユーザー出口プログラムでは、CONCURRENCY(REQUIRED) を指定している場合、CICS は CONCURRENCY(THREADSAFE) が指定されているかのようにプログラムを処理します。

グローバル・ユーザー出口は、OPENAPI プログラムとして定義できませんが、グローバル・ユーザー出口に対して **ENABLE PROGRAM** コマンドで THREADSAFE オプションを使用する場合、スレッド・セーフ・プログラムとして使用可能になり、それを使用するアプリケーションと同じオープン TCB 上で実行できます。**ENABLE PROGRAM** コマンドが CONCURRENCY オプションまたは API オプションを指定しない場合、プログラム定義のオプションが使用されます。

シリアル化手法

オープン TCB 上で実行されているアプリケーションは、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ル化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。

ファイルや一時データ・キュー、一時記憶域キュー、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。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ル化します。

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その他のリソースでは、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。ユーザー・アプリケーション・プログラムのためのスレッド・セーフ・プログラミングについて詳しくは、アプリケーションのスレッド・セーフ化を参照してください。

TCB の切り替え

スレッド・セーフ・プログラムは、オープン TCB と QR TCB 間を切り替えることなく、これらのプログラムがオープン TCB 上に留まることによって、パフォーマンスを良好にしています。プログラムがスレッド・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場合、オープン TCB から QR TCB への TCB の切り替えが、次のような環境で発生します。

- プログラムがスレッド・セーフではない EXEC CICS コマンドを発行する場合、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。プログラムが OPENAPI または CONCURRENCY (REQUIRED) として定義されている場合、CICS はオープン TCB に切り替えて、アプリケーション・ロジックの実行を継続します。プログラムが OPENAPI または CONCURRENCY(REQUIRED) として定義されていない場合、QR TCB 上での実行が継続されます。CICS Db2 アプリケーションでは、プログラムが OPENAPI または CONCURRENCY(REQUIRED) として定義されておらず、それ以上の Db2 要求を行わない場合は、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、不利益にしかありません。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。
- 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場合、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した場合、状態は非スレッド・セーフ EXEC CICS コマンドの実行後と同じです。OPENAPI または CONCURRENCY(REQUIRED) プログラムがオープン TCB に切り替え、OPENAPI または CONCURRENCY(REQUIRED) として定義されていないプログラムは QR TCB 上での実行が継続されます。
- プログラムがスレッド・セーフ CICS コマンドを発行するか、Db2 要求を行う場合、グローバル・ユーザー出口がコマンドまたは要求の処理の一部として呼び出されることがあります。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するか、Db2 要求を完了します。

- 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。

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) または CONCURRENCY(REQUIRED) 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・セーフ標準に適合するように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) または CONCURRENCY(REQUIRED) 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) または CONCURRENCY(REQUIRED) 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。これを行うために、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) または CONCURRENCY(REQUIRED) が定義されている場合、処理はオープン TCB で継続できます。このシナリオではプログラム 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・セーフ標準に適合するようにコーディングされている必要があります。

アプリケーションのスレッド・セーフ化

アプリケーション・プログラムをスレッド・セーフにする場合、オープン・トランザクション環境を使用して、TCB の切り替えを回避し、良好なパフォーマンスを得ることができます。

始める前に

スレッド・セーフ・アプリケーション・プログラムを使用するには、システム初期設定パラメーター **FORCEQR** が YES に設定されていないことを確認します。

FORCEQR を使用すると、スレッド・セーフとして定義されたプログラムが強制的に QR TCB 上で実行されます。スレッド・セーフとして定義されているプログラムに関連する問題を調査して解決している間、一時的な手段として YES に設定する場合があります。

また、ファイル所有 CICS 領域のシステム初期設定パラメーター **FCQRONLY** についても、適切な設定を選択します。 **FCQRONLY** が YES に設定されている場合、CICS は強制的に CICS 領域のすべてのファイル制御要求を QR TCB 上で実行します。

- IPIC 接続を使用してファイル制御要求をリモート CICS TS 4.2 またはそれ以降の領域に機能シップする場合、これらの接続のパフォーマンスを改善するために、ファイル所有領域で、**FCQRONLY** を NO に設定します。
- MRO リンクまたは ISC over SNA 接続のみを使用している場合、またはファイル所有領域が CICS TS 4.2 より前のものである場合、ファイル所有領域で **FCQRONLY** を YES に設定します。

CICS 相互通信を使用して関数またはプログラムをリモート CICS システムで実行する要求を行っている場合、CICS システム間で TCP/IP 経由の IP 相互接続 (IPIC) を選択すると、スレッド・セーフ・アプリケーションへの最適なサポートが提供されます。IPIC 接続を利用すると、CICS はオープン TCB を使用してリモート CICS システムの要求を管理するミラー・プログラムを実行し、スループットが向上します。他の接続タイプを使用すると、CICS はミラー・プログラムの実行にオープン TCB を使用しません。 **EXEC CICS LINK** コマンドは、分散プログラム・リンク (DPL) に対して使用され、長期間実行されるミラーが使用されているリモート CICS 領域への IPIC 接続に対してはスレッド・セーフですが、他の接続タイプに対してはスレッド・セーフではありません。

このタスクについて

スレッド・セーフ・プログラムでは、プログラムがスレッド・セーフであるとはどういうことか、およびオープン TCB と QR TCB の間で TCB の切り替えが起こる事情について説明しています。

アプリケーション・プログラムをスレッド・セーフにしてオープン TCB 上に残すには、以下の手順を使用します。

手順

1. PROGRAM リソース定義で CONCURRENCY(THREADSAFE) を指定して、CICS に対してプログラムをスレッド・セーフとして定義します。

OPENAPI として定義されているプログラムの場合、CICS は CONCURRENCY(THREADSAFE) オプションを必要とします。スレッド・セーフとして定義されるコードのみが、オープン TCB の実行を許可されます。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 EXEC CICS コマンドが TCB の切り替えを使用して安全に処理されることを保

証できますが、プログラムがオープン TCB 上で実行できるようにするために、CICS はユーザーにアプリケーション・ロジックがスレッド・セーフであることを保証するよう求めます。

または、プログラムを CONCURRENCY(REQUIRED) として定義して、プログラムを最初からオープン TCB 上で実行できるようにすることも可能です。CONCURRENCY(REQUIRED) として定義されるプログラムは、常にオープン TCB 上で実行する必要があるため、スレッド・セーフ標準に適合するようコーディングする必要があります。使用されるオープン TCB のタイプは、API 設定によって異なります。

2. プログラムのロジック、つまり、**EXEC CICS** コマンド間のネイティブ言語コードがスレッド・セーフであることを確認します。

プログラムをスレッド・セーフとして CICS に対して定義するものの、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 は起こりうる結果からプログラムを保護することができません。プログラム・ロジックをスレッド・セーフにするには、共用リソースにアクセスするときに適切なシリアル化手法を使用して、これらのリソースへの同時アクセスを禁止する必要があります。**EXEC CICS** コマンドを使用して、ファイル、一時データ・キュー、一時記憶域キュー、Db2 のテーブルなどのリソースにアクセスする場合、CICS はスレッド・セーフな処理を保証しますが、共用ストレージなど、ユーザー・プログラムによって直接アクセスされるリソースの場合、ユーザー・プログラムがスレッド・セーフな処理を保証する必要があります。

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ことにより、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を確認します。

- ADDRESS CWA
- EXTRACT EXIT
- GETMAIN SHARED

ロード・モジュール・スキャナー・ユーティリティには、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、共用ストレージへのアクセスを可能にする CICS コマンドを識別することができます。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、そのすべてがグローバル・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。

ヒント: 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も組み込んでください。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には、次の手法を使用できます。

- 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、以下の手法のいずれかを使用して、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。
- 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。

3. 最適なパフォーマンスを実現するために、プログラムがスレッド・セーフ **EXEC CICS** コマンドのみを使用していることを確認します。

スレッド・セーフとして定義され、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** コマンドを含める場合、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。アプリケーションの結果は影響を受けませんが、パフォーマンスが影響を受ける場合があります。

スレッド・セーフなコマンドは、CICS API および SPI コマンド・トピックのコマンドの説明で、「このコマンドはスレッド・セーフです」という文で示されます。それらは、スレッド・セーフ・コマンドおよびスレッド・セーフ SPI コマンドにもリストされています。

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 DFHEIDNT が組み込まれており、これを使用すると、アプリケーションのスレッド・セーフではない CICS コマンドを特定することができます。

4. 任意のユーザー出口プログラムが、プログラムの使用する実行パスにある場合、最適なパフォーマンスを実現するには、それらがスレッド・セーフ標準に適合するようコーディングされ、スレッド・セーフとして CICS に対して定義されるようにします。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らの出口に該当します。また、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・セーフ標準に適合するようコーディングされ、スレッド・セーフとして CICS に対して定義されることを確認します。

スレッド・セーフ・ユーザー出口プログラムは、それを呼び出すスレッド・セーフ・アプリケーションと同じオープン TCB 上で使用でき、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。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・セーフではない場合、CICS はそれらを実行するために QR TCB に切り替えますが、これはアプリケーションのパフォーマンスに悪影響を与える可能性があります。

以下の重要なユーザー出口に注意してください。

- グローバル・ユーザー出口 XEIN および XEIOOUT が、**EXEC CICS** コマンドの前後に呼び出されます。
- グローバル・ユーザー出口 XPCFTCH が、CICS に対して定義されたプログラムが制御を受け取る前に呼び出されます。
- CICS Db2 要求の場合、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 要求に対する他の重要な出口には、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 DSNCEXT、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。
- IPIC 接続経由でリモート CICS 領域に対して CICS ファイル制御要求、一時データ要求、または一時記憶域要求を機能シップする場合、要求はスレッド・セーフであり、オープン TCB 上のリモート CICS 領域で実行することができます。最大限のパフォーマンスを実現するため、ファイル制御要求、一時データ要求、または一時記憶域要求のためにリモート CICS 領域で呼び出されるグローバル・ユーザー出口プログラムは、すべて、スレッド・セーフ・プログラムとして使用可能にする必要があります。
- a. ユーザー出口プログラムを CICS に対してスレッド・セーフとして定義するために、PROGRAM リソース定義で適切な属性を指定できます。
 - タスク関連ユーザー出口プログラムの場合は、OPENAPI および THREADSAFE を指定するか、THREADSAFE のみを指定します。

- グローバル・ユーザー出口プログラムの場合は、OPENAPI を使用することはできませんが、THREADSAFE を指定することができます。

グローバル・ユーザー出口プログラムまたはタスク関連ユーザー出口プログラムで CONCURRENCY(REQUIRED) を指定している場合、CICS は **CONCURRENCY(THREADSAFE)** が指定されているかのようにプログラムを処理します。

- b. ユーザー出口プログラムを CICS に対してスレッド・セーフとして定義するための代わりの方法として、**EXEC CICS ENABLE PROGRAM** コマンドを使用してこのプログラムを使用可能にするときに、適切なオプションを指定できます。
 - タスク関連ユーザー出口プログラムの場合は、OPENAPI または THREADSAFE を指定します。
 - グローバル・ユーザー出口プログラムの場合は、OPENAPI を使用することはできませんが、THREADSAFE を指定することができます。

OPENAPI オプションか THREADSAFE 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、このアクションにより CICS にプログラム論理がスレッド・セーフであることが示されるので、CICS は出口のプログラム定義の CONCURRENCY 設定を指定変更して、出口プログラムをスレッド・セーフとして処理します。

- c. 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 PROGRAM** コマンドにおいて THREADSAFE を指定します。 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。第 1 フェーズの PLT プログラムが CICS の初期化の早い段階で実行されるため、インストールされた PROGRAM リソース定義、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定義することはできません。 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。 ただし、**EXEC CICS ENABLE PROGRAM** コマンドの設定は、システムにより自動インストールされたプログラム定義の **CONCURRENCY(QUASIRENT)** 設定を指定変更します。

CONCURRENCY(REQUIRED) プログラム

アプリケーション・プログラムを CONCURRENCY(REQUIRED) として定義することは、プログラムの開始時から、そのプログラムが常にメイン CICS 準再入可能 TCB (QR TCB) の代わりに、オープン・タスク制御ブロック (オープン TCB) で実行されることを意味します。 **EXEC CICS** コマンドを処理するために CICS を QR TCB に切り替える必要がある場合、CICS は、オープン TCB に再度切り替えてから制御をアプリケーション・プログラムに戻します。

使用されるオープン TCB のタイプは、プログラムが使用する API に応じて決まります。

- プログラムが CICS のサポートする API (Db2、IMS™、および IBM MQ などの外部リソース・マネージャーへのアクセスを含む) のみを使用する場合、その

プログラムはプログラム属性 API(CICSAPI) で定義されている必要があります。この場合、CICS コマンドが TCB のキーに依存しないため、CICS はプログラムの実行キーに関係なく、常に L8 オープン TCB を使用します。

- プログラムが他の非 CICS API を使用する場合、そのプログラムはプログラム属性 API(OPENAPI) で定義されている必要があります。この場合、CICS はプログラムの実行キーに応じて L9 TCB または L8 TCB を使用します。これは、非 CICS API を正しく操作できるようにするためです。

グローバル・ユーザー出口は CONCURRENCY(REQUIRED) として定義することはできませんが、これらの出口を ENABLE PROGRAM コマンドで THREADSAFE オプションを使用して有効にする場合は、必要に応じてオープン TCB で実行することが可能です。

タスク関連ユーザー出口 (TRUE) は、CONCURRENCY(REQUIRED) として定義したり、使用可能にしたりできますが、その場合、TRUE をスレッド・セーフ標準に従って記述する必要があります。CONCURRENCY(REQUIRED) API(OPENAPI) として定義されている TRUE は常に L8 TCB で実行されます。

CONCURRENCY(REQUIRED) API(CICSAPI) として定義されている TRUE は、アプリケーション環境のニーズに応じて、T8、L8、または X8 TCB で実行することができます。CICS-Db2 TRUE は CONCURRENCY(REQUIRED) API(CICSAPI) として有効にすることができます。

(Db2 呼び出しと同じ TCB で実行できるというパフォーマンス上のメリットを得るため) THREADSAFE CICSAPI として定義されている既存のスレッド・セーフ CICS-Db2 アプリケーションは、REQUIRED CICSAPI として定義することにより、さらに拡張することができます。この定義は、プログラムをオープン TCB へ移動させるための最初の Db2 呼び出しを待機することなく、プログラムを即座に L8 オープン TCB で実行できることを意味します。アプリケーションが実行する非スレッド・セーフ CICS コマンドがある場合は、その数に応じて、さらにメリットが得られます。

OPENAPI プログラム

オープン・トランザクション環境 (OTE) とは、CICS アプリケーション・コードが、他のトランザクションからの干渉を受けることなく、CICS アドレス・スペース内で非 CICS サービスを使用できる環境のことです。PROGRAM リソース定義で OPENAPI 属性を使用することによって、ユーザー・アプリケーション・プログラム、PLT プログラム、ユーザー置き換え可能モジュール、またはタスク関連ユーザー出口 (TRUE) を OPENAPI プログラムとして定義し、OTE を使用できます。

属性 API(OPENAPI) を使用することによりプログラムを OPENAPI プログラムと定義することは、プログラムの開始時から、そのプログラムが常にメイン CICS 準再入可能 TCB (QR TCB) の代わりに、L8 または L9 モードのオープン・タスク制御ブロック (オープン TCB) で実行されることを意味します。グローバル・ユーザー出口は、OPENAPI プログラムとして定義することはできませんが、**ENABLE PROGRAM** コマンドで THREADSAFE オプションを使用して有効にする場合は、必要に応じてオープン TCB で実行することが可能です。

アプリケーションのワークロードを QR TCB から複数のオープン TCB に移動すると、特に CPU 集中型プログラムで、スループットが向上する可能性があります。

す。他の非 CICS API を使用できますが、非 CICS API を CICS 内で使用することは、完全にユーザーの責任になることに注意する必要があります。CICS 内の非 CICS API のテストは行われておらず、そのような API の使用は IBM Service によってサポートされていません。

OPENAPI プログラムは、オープン TCB 上で実行するにはスレッド・セーフである必要があります。スレッド・セーフなプログラムの要件は、次のとおりです。

1. プログラムは、CICS に CONCURRENCY(REQUIRED) として定義する必要があります。これはつまり、このプログラムはオープン TCB での実行が必須であることを意味します。
2. プログラムのロジック、つまり、EXEC CICS コマンド間のネイティブ言語コードがスレッド・セーフである必要があります。プログラムを CICS に対してオープン TCB で実行されるよう定義するが、スレッド・セーフではないアプリケーション・ロジックを含める場合、結果は予測不能であり、CICS は起こり得る結果を防ぐことができません。
3. 最適なパフォーマンスを実現するために、プログラムは、スレッド・セーフ EXEC CICS コマンドのみを使用する必要があります。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与える場合があります。
4. 最適なパフォーマンスを実現するには、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必要があります。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・セーフではない場合、CICS はそれらを実行するために QR TCB に切り替えますが、これはアプリケーションのパフォーマンスに悪影響を与える可能性があります。

78 ページの『スレッド・セーフ・プログラム』では、スレッド・セーフ・プログラムの要件について、より詳細に説明しています。

OPENAPI プログラムには、いくつかの追加の責任と制限があります。例えば、タスクの終了のために特別に獲得した非 CICS リソースはすべて解放する必要があります。特定の MVS システム・サービスを使用しないようにする必要があります。OPENAPI プログラムのスレッド・セーフの制約事項では、これらの要件について説明しています。

(アプリケーション・ロジックがスレッド・セーフと仮定した場合の) REQUIRED OPENAPI として定義する候補となるプログラムには、他の非 CICS API を自身の責任において使用するプログラムなどがあります。

OPENAPI プログラムの TCB

次の TCB は、OPENAPI プログラムに対して使用されます。

- L8 モード TCB は、Web サービス要求の処理、XML の構文解析、および CICS の Web サポートのための z/OS UNIX ファイルへのアクセスのために L8 TCB 上で実行されるいくつかの CICS プログラムを含む CICS キー OPENAPI アプリケーション・プログラムに対して使用されます。

- L9 モード TCB は、ユーザー・キー OPENAPI アプリケーション・プログラムに対して使用されます。

L8 モード TCB は、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 TRUE を経由して、プログラムがリソース・マネージャーにアクセスする必要がある場合にも使用されます。オープン API TRUE は、L8 モード TCB の下で制御され、サブタスク TCB を作成することなく非 CICS API を使用できます。

CICS では、L8 モード・オープン TCB および L9 モードのオープン TCB のプール内の L8 TCB および L9 TCB の数が自動的に制御されます。

OPENAPI プログラムを使用すると、通常のスレッド・セーフ・プログラムより多くの TCB の切り替えを引き起こす可能性があります。OPENAPI プログラムが、スレッド・セーフではない EXEC CICS コマンドまたはユーザー出口プログラムを使用して、QR TCB への切り替えを引き起こす場合、CICS はオープン TCB に切り替えて、アプリケーション・ロジックの実行を継続するため、余分なスイッチが存在します。TCB のキーが OPENAPI プログラムに対して適切であることが必要とされるため、追加の TCB の切り替えが発生する場合があります。OPENAPI TRUE は常に L8 TCB 上の CICS キーで実行されるため、例えば、ユーザー・キー OPENAPI プログラムが L9 TCB 上で実行されるものの Db2 呼び出しを行う場合、CICS は Db2 を呼び出すために L8 TCB に切り替えてから、L9 TCB に戻ってプログラムの実行を継続します。この切り替えのために、通常、CICS Db2 アプリケーションは OPENAPI プログラムではなく、(CICSAPI) スレッド・セーフ・プログラムとして定義されます。CICS キー CICS Db2 アプリケーションは、必要に応じて OPENAPI プログラムとして定義できます。

OPENAPI プログラムのスレッド・セーフの制約事項

OPENAPI プログラムは、QR TCB による制約からは自由ですが、CICS システム全体と、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。

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。このようなリソースには、以下が含まれる場合があります。
 - 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 を使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とも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、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なら、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) については、PROGRAM 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 について詳しくは、CICS 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保管するための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ります。

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

これらの領域は、すべて、ソースが動的ストレージ域 (DSA)、拡張動的ストレージ域 (EDSA)、または 2 GB 境界より上の動的ストレージ域 (GDSA) である主記憶域機構です。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。それぞれの記憶域機構は、アクセス可能性と存続期間が異なります。したがって、異なる一連のストレージ・ニーズに対応します。

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 の決定について詳しくは、TRANSACTION 属性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・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、358 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。

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。

GETMAIN コマンドの USERDATAKEY オプションおよび CICSDATAKEY オプションによってトランザクション・リソース定義の TASKDATAKEY オプションを指定変更する方法については、358 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 コマンドに伴うプロセッサ・オーバーヘッドのため、ユーザー・ストレージは小規模なストレージに使用しないでください。小規模なものについては、トランザクション作業域 (TWA) を使用するか、または大きい要求にまとめてください。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、長く保持することができますが、プロセッサ・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、118 ページの『RETURN コマンドでの COMMAREA の使用』を参照してください。

COMMAREA 内の情報は、関与している 2 つのプログラムでのみ使用可能です。ただし、後でトランザクションで呼び出される可能性のある他のプログラムでもデータを使用できるよう、これらのプログラムで明示的に設定している場合を除きます。

- 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ため、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。

どちらの場合も、制御を受け取るプログラムに領域のアドレスが渡されて、CICS コマンド・レベル・インターフェースはアドレス可能度をセットアップします。詳しくは、162 ページの『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 は受信側プログラムのアドレッシング・モードに適合する区域内で COMMAREA を作成することにより、受信側プログラムで COMMAREA を確実にアドレッシングできるようにします。受信側が AMODE(24) である場合、COMMAREA は 16 MB 境界より下で作成されます。受信側が AMODE(31) である場合、COMMAREA は 16 MB 境界より上、2 GB 境界より下で作成されます。

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 について詳しくは、358 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。128 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、120 ページの『チャネルによるプログラム間データ転送』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。

一時記憶域キュー

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、一時記憶域キューに保持されます。項目は、親タスクによって、または一時記憶域キューに割り当てられたシンボル名を使用することにより他のタスクによって取り出すことができます。

複数の項目が入っている一時記憶域キューは、小さいデータ・セットと考えることができます。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。項目は、順次に、または項目番号によって直接に、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。

一時記憶域キューは、最大 16 文字のシンボル名によって識別されます。名前の重複による競合を避けるため、命名規則を設定します。例えば、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。一時記憶域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、端末およびデータ・セット・レコードなどのリソース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。

タスク間でデータを受け渡す他のメソッドと比較して、一時記憶域キューは、プロセッサの使用が増える場合があります。一時記憶域キューに対する各対話に対して EXEC CICS コマンドを使用し、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。このため、主一時記憶域でのプロセッサの使用は、CWA または TCTUA より増えます。補助記憶装置を使用すると、通常、データ・セットの I/O もあります。共用された一時記憶域プールは、一時記憶域サーバーを必要とし、アプリケーションはデータを取得するためにカップリング・ファシリティーにアクセスする必要があります。

ただし、一時記憶域キューは、データの受け渡しを行う他の方法と比べて、いくつかの利点があります。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し、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。このため、一時記憶域キューは、比較的大容量のデータや、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。また、一時記憶域キューでは、リソース保護も可能です。

一時記憶域キューは、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、ストレージ内にそのまま残っています。アプリケーションは、**DELETEQ TS** コマンドを使用して、一時記憶域キューをその有用な期間が終了したときに削除することができます。アプリケーションが常に一時記憶域キューを削除できるわけではない場合は、自動削除の設定を検討してください。最近アクセスされていない場合に、主ストレージまたは補助ストレージのリカバリー不能な一時記憶域キューを CICS が自動的に削除するようにできます。TSMODEL リソース定義の有効期限のインターバルが、自動削除を制御します。

一時記憶域キューの場所の選択

CICS 領域の一時記憶域は、z/OS カップリング・ファシリティの主ストレージ、補助ストレージ、または共用一時記憶域プール内に配置することができます。さまざまな一時記憶域の場所と各場所の一時記憶域キューで利用可能な機能の概要については、「パフォーマンスの改善」の「CICS 一時記憶域: 概要」を参照してください。

アプリケーションは、データの最初の項目を一時記憶域キューに書き込むために、**WRITEQ TS** コマンドを使用します。コマンドは、一時記憶域キューのシンボル名を指定します。この名前がインストールされた TSMODEL リソース定義に一致する場合、CICS は TSMODEL リソース定義によって指定された一時記憶域の場所に一時記憶域キューを作成します。一致する TSMODEL リソース定義がない場合、CICS はアプリケーションが **WRITEQ TS** コマンドで指定する一時記憶域の場所を使用します。デフォルトの場所は、補助一時記憶域です。

一時記憶域キューの配置場所を選択する場合、次の要因を考慮してください。

使用の存続時間と頻度

- 主一時記憶域は、短期間に必要である、またはアクセス頻度が高い一時記憶域キューにより適しています。
- 補助一時記憶域および共用一時記憶域プールは、存続期間が比較的長い、またはアクセス頻度が低い一時記憶域キューにより適しています。キューが主ストレージ、または補助ストレージやカップリング・ファシリティ・ストレージのいずれに行くかを決定するために、存続期間に 1 秒というカットオフ・ポイントを使用できます。

リカバリー

- 主ストレージの一時記憶域キューは、リカバリー可能として定義できません。
- 補助ストレージの一時記憶域キューは、リカバリー可能として定義できます。
- CICS のリカバリーは、共用一時記憶域プールの一時記憶域キューでは利用できません。ただし、カップリング・ファシリティは CICS の再始動によって影響を受けません。このため、共用一時記憶域プールの一時記憶域キューは、持続していると考えられます。

自動削除

有効期限のインターバルを対応する一時記憶域モデルに追加することによって、適格な一時記憶域キューを、必要なくなったときに CICS が自動的に削除するよう指定できます。

- 主ストレージの一時記憶域キューに対して有効期限の間隔を設定できます。
- 補助一時記憶域のリカバリー不能キューに対して有効期限の間隔を設定できます。 リカバリー可能キューは、自動的に削除できません。
- 共用一時記憶域プール内の一時記憶域キューは、自動的に削除できます

ストレージ・タイプ

主一時記憶域は、CICS 領域内の 64 ビット・ストレージです。リカバリー可能な一時記憶域を必要としない場合は、アプリケーションで主一時記憶域を使用するように指定することができます。この結果、31 ビット・ストレージのスペースへの圧力が少なくなり、データをディスクに書き込む I/O アクティビティーが減少します。

一時記憶域キューのロックおよび待機

CICS の一時記憶域ドメインは複数の要求を同時に処理できますが、同じ一時記憶域キューに対する要求はシリアル化されます。 キューは各要求期間中はロックされます。

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域キューに書き込み、または削除を行うことはできません。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。

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。 このタスクは、**HANDLE CONDITION NOSPACE** コマンドか、**RESP** または **NOHANDLE** オプションを指定した **WRITEQ TS** コマンドのいずれかを使用して、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。この状態は、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する必要があります。

区画内一時データ

区画内一時データは、単一データ・セットに、CICS が主ストレージに維持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。

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、112 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。一時記憶域キューと違って、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、一時記憶域キューの項目は、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せんが、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。

- 一時記憶域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 または GETMAIN64 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 または FREEMAIN64 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、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。符号付きハーフワード 2 進数値の使用によって、長さの理論的な上限が 32 KB に制限されます。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・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、リファレンス: アプリケーション開発を参照してください。可能であれば、CICS コマンド・オプションで指定する長さは、24 KB を超えないようにしてください。詳しくは、CICS コマンドの **LENGTH** オプションを参照してください。

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。

EXEC CICS LINK コマンドを使用して **COMMAREA** にデータを渡す場合、**COMMAREA** で渡すデータの長さに一致する **LENGTH** 値を指定するようにしてください。**LENGTH** には 0 (ゼロ) を指定しないでください。ゼロを指定すると、結果の振る舞いが予測不能で、**EXEC CICS LINK** コマンドが失敗する場合があるためです。**COMMAREA** を使用してデータを渡す場合、リンクされているプログラムは、タスクの **EIB** の **EIBCALEN** フィールドが、プログラムの予期する内容に一致するか検証しなければなりません。不一致があると、記憶保護違反またはシステム障害になる場合があります。詳しくは、165 ページの『**COMMAREA**』を参照してください。

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を示されます。

WAIT EXTERNAL コマンドおよびいくつかの **QUERY SECURITY** コマンドには、リソースの状況または定義を提供するオプションがあります。CICS は、これらのオプションに関連する値を提供します。このことにより、CICS 値データ域という名前が付いています。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-value data areas (CVDAs)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 212 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・サイズが限界を 64 KB 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ます。

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 24 KB より低く設定されることがあります。

データ・セットの作成については、Defining data setsを参照してください。ファイルのリソース定義については、FILE リソース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 32,000 バイト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・サイズを満たす限り、この限度が原因で、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ほとんどありません。

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

以下の手法を使用して、アプリケーションをエラー・フリーにするのに役立ててください。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点です。各システムの有効範囲は、まだ十分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

これらのストレージのツールおよび技法を使用して、アプリケーション・プログラムのエラーを最小化します。

このタスクについて

- 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どうかは、CICS システム初期設定パラメーターで選択できます。この機能の詳細については、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

ここでは、アプリケーションに適用される一般的な規則をいくつか紹介します。

このタスクについて

- テストは、実動 CICS システムで実施しないでください。稼働中のデータベースに影響を与えることなくエラーを切り分けられるテスト・システムを使用してください。
- 可能であれば、アプリケーション開発者以外がテストを実施してください。
- 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。 1042 ページの『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。 927 ページの『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・テストを使用して、シングルユーザー環境では検出されない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の優れたツールです。

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。

- 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。複数領域操作 (Multiregion operation) 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に対して最終テストを実施して、エラーがないか調べてください。

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。

問題の解決について詳しくは、Troubleshooting and support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、端末を持たないトランザクションで使用されるリソースを、無許可の使用から保護できます。

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 コマンド・サマリーを参照してください。

パフォーマンスの設計

アプリケーション・プログラムの設計を変更して、パフォーマンスと効率を改善することができます。

- 『プログラム・サイズ』
- 104 ページの『仮想記憶域』
- 108 ページの『リソースの排他制御』
- 109 ページの『操作のコントロール』
- 110 ページの『オペレーティング・システム待機』
- 110 ページの『NOSUSPEND オプション』
- 112 ページの『効果的な順次データ・セットのアクセス』
- 113 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、69 ページの『第 4 章 アプリケーションの設計』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

注意深く設計することによって、使用される仮想記憶域の量を最小に抑え、アプリケーションのオーバーヘッドを削減できます。

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

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。

このタスクについて

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:

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。

このタスクについて

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。
- 数箇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してください。
- 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。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まいません。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、前述の示唆の中には、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:

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。

このタスクについて

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:

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。

このタスクについて

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 (ページング技法) 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。参照されないページのコードをページインしなくても済むようにするため、システムは、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、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。さまざまな方法を採用して、リソースの競合の遅延を削減できます。

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。

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: リカバリー不能データ・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、302 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。868 ページの『アプリケーション・プログラムにおけるリソースのロック (エンキュー)』に、リソース保護の範囲に関する詳細情報があります。

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

多数の操作手法を使用して、CICS システムのパフォーマンスと効率に影響を与えることができます。

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ますようになります。実行する準備ができていない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 について詳しくは、CICS で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。

オペレーティング・システムの待機的主要原因には、次のものがあります。

- 区画外一時データ・セット (112 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。言語の制約事項のガイダンス情報については、653 ページの『第 8 章 COBOL アプリケーションの開発』、593 ページの『第 7 章 C および C++ アプリケーションの開発』、および 693 ページの『第 10 章 PL/I アプリケーションの開発』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 およびアセンブラ言語マクロ。

NOSUSPEND オプション

特定のコマンドのデフォルトの処置では、必要なリソースが使用可能になるまでアプリケーションが中断されます。NOSUSPEND オプションを使用すると、そのような待機を禁止して、アプリケーション・プログラム内の次の命令に戻るようになります。

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。これらの条件が発生する可能性があるコマンドは、以下のとおりです。

- ALLOCATE
- ENQ
- GETMAIN
- WRITE JOURNALNAME
- WRITE JOURNALNUM
- READQ TD
- WRITEQ

NOSUSPEND オプション (ALLOCATE コマンドの場合は NOQUEUE オプションともいう) を使用して、そのような待機を禁止し、即座に、アプリケーション・プログラム内でコマンドの次にある命令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、COBOL、PL/I、およびアセンブラ言語アプリケーション (ただし、AMODE(64) アセンブラ言語アプリケーションを除く) でのみサポートされています。サポートされている他のすべての高水準言語では使用できません。

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、以下の 3 つの状態のいずれかを示します。

- 現在、ラベルが指定されている。
HANDLE CONDITION condition(label)
- 条件が無視される。
IGNORE CONDITION
- 現在、ラベルが指定されていない。
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. 条件に対する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せん。
 - 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。それぞれが異なるパフォーマンスの特性を持っています。

一時記憶域および区画内一時データ・キュー (既に、95 ページの『一時記憶域キュー』および 97 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もので、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 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の必要性和、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。

非同期要求のための設計

アプリケーションを設計する場合は、データ・タイプ、連続するトランザクションやプログラム間でのデータの受け渡し、リカバリー、パフォーマンス、セキュリティ、および保全性を考慮する必要があります。アプリケーション・プログラマーは、自分がプログラマーとしてどのような側面を処理する必要があり、CICS でどの側面が処理されるかについての知識を必要とします。

プログラム間でのデータの受け渡し

親タスクが子タスクを実行する場合、チャンネルは **EXEC CICS RUN TRANSID** コマンドで指定できます。チャンネル上のコンテナは、すべてコピーされて子に渡されます(これにより、子が処理を実行中にチャンネル・データが誤って更新されることがなくなります)。チャンネルが存在しない場合は、作成されます。

親タスクが子タスクから確実に応答を受け取れるようにするには、親タスクから子タスクに渡すデータがない場合でも、必ず **EXEC CICS RUN TRANSID** コマンドに **CHANNEL** オプションを指定してください。

親タスクが **EXEC CICS FETCH CHILD CHANNEL** を使用して子タスクから応答を受け取ると、親タスクは CICS が指定した新しいチャンネルを取得します。親は、この新規チャンネルを使用して、子から返されたデータを受け取ります。

注: 非同期に開始された子タスクは、端末に関連付けられていないため、**EXEC CICS RETURN TRANSID** コマンドを使用して制御を返すことはできません。

親タスクと子タスクの保護

子タスクは親タスクと同じユーザー ID の下で実行されます。

親と子の接続

子タスクはローカルで実行する必要があります。子タスクは、リモート・サービスを呼び出す (例えば、Web サービスを呼び出したり、別の CICS 領域への分散プロ

グラム・リンクを実行したりする) ことができますが、最初はその親タスクに対してローカルでなければなりません。子は、これをその子とリンク先プログラムに伝えることができます。返された結果や子のデータを受け取ることができるのは、親のみです。

リカバリーのための設計

親と子は、それぞれ別個のタスクです。子と子の間または親と子の間のいずれでも、CICS による作業単位の調整は行われません。

トランザクション間のデータの共用

CICS には、トランザクション間でデータを共用するためのいくつかの機能があります。共通作業域 (CWA)、TCTTE ユーザー域 (TCTUA)、COMMAREA、表示画面、またはチャンネルとコンテナを使用することができます。

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について詳しくは、175 ページの『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』
- 118 ページの『TCTTE ユーザー域 (TCTUA) の使用』
- 118 ページの『RETURN コマンドでの COMMAREA の使用』
- 119 ページの『RETURN コマンドにおけるチャンネルの使用』
- 119 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におけるあらゆるプログラムは、共用使用と同じ規則に従わなくてはなりません。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。

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。詳細については、『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。

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 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してください。

CWA より CICS 保護を優先するには、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) を指定します。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、358 ページの『ストレージ制御』を参照してください。

116 ページの図 24 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。

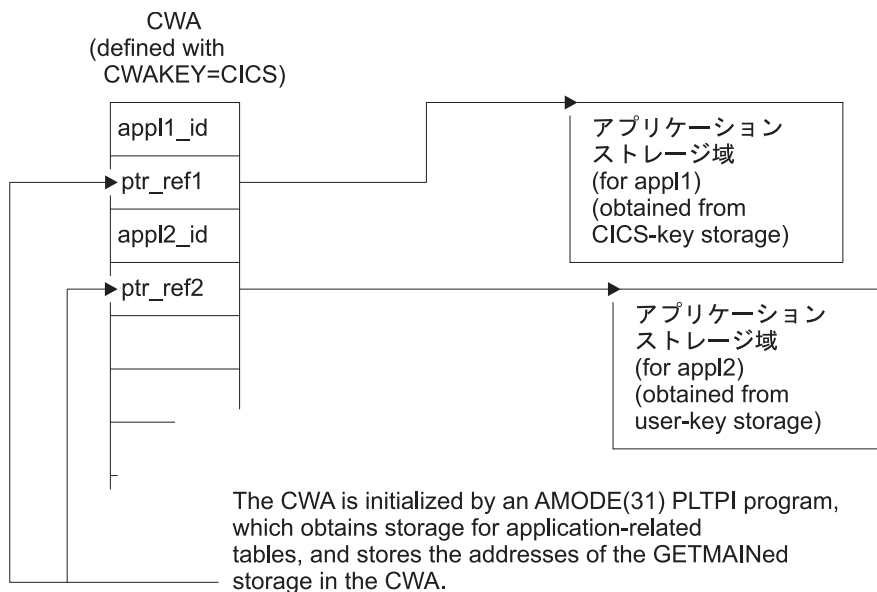


図 24. CICS キー・ストレージにおける CWA の使用例： この図は、ユーザー・キーまたは CICS キー・ストレージにおいて、アプリケーション・プログラムが使用するために確保されているストレージを参照するための、CWA の使用方法を示しています。CWA 自体は、CICS キー・ストレージに入れられているため、保護されています。

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、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 24 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。

117 ページの図 25 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 25. 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、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性を、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 境界より上と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・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 インターフェイスが、

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 共用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 (93 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、SNA 用 z/OS Communications Server の合計最大データ長は 32KB です。この制限は、Communications Server 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、以下のとおりです。

- プロセッサのオーバーヘッドは大きくない (XCTL コマンドで COMMAREA を使用すると同等で、主一時記憶域を使用するのとはほぼ同じです)。
- リカバリー可能ではない。
- リソース保護はない。
- 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。128 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、120 ページの『チャネルによるプログラム間データ転送』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できます。

例えば、ユーザーのせいでデータ入力中に発生したエラーは、データを処理するトランザクションによって (強調表示またはメッセージを使用して) 強調表示されます。次のトランザクション ID は、(訂正後の入力を処理するように) そのトランザクション自身のものを指すように設定され、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)。

チャネルによるプログラム間データ転送

チャネルは、COMMAREA に適用される 32 KB 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手法です。

このセクションには、次のものが含まれています。

- 121 ページの『チャネル: クイック・スタート』
- 125 ページの『チャネルの使用: いくつかの典型的なシナリオ』
- 130 ページの『チャネルの作成』
- 131 ページの『現行チャネル』
- 136 ページの『チャネルの有効範囲』
- 141 ページの『プログラムに渡されたコンテナの検出』
- 141 ページの『リンクから返されたコンテナの検出』
- 147 ページの『CICS の読み取り専用コンテナ』
- 144 ページの『チャネルの設計: 最良事例』
- 146 ページの『チャネルの構成および使用: 例』
- 148 ページの『チャネルと BTS アクティビティ』
- 147 ページの『JCICS からのチャネルの使用』
- 150 ページの『チャネルを使用した動的ルーティング』
- 150 ページの『データ変換』
- 128 ページの『チャネルの利点』
- 158 ページの『COMMAREA からチャネルへのマイグレーション』
- 142 ページの『チャネルとコンテナの削除とそのストレージの解放』

チャンネル: クイック・スタート

チャンネルおよびコンテナの簡単な紹介。

チャンネルおよびコンテナ

コンテナとは、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。プログラムは、任意の数のコンテナを相互に受け渡すことができます。コンテナは、チャンネルと呼ばれる集合にグループ化されます。チャンネルはパラメーター・リストに類似しています。

名前付きコンテナを作成してチャンネルに割り当てるために、プログラムは **EXEC CICS PUT CONTAINER**(コンテナ名) **CHANNEL**(チャンネル名) コマンドを使用します。次に、**EXEC CICS LINK**、**XCTL**、**START**、**RUN TRANSID** または **RETURN** の各コマンドの **CHANNEL(channel-name)** オプションを使用して、チャンネルとそのコンテナを 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**(コンテナ名) コマンドを使用して、渡されたコンテナ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ンネルに属する名前付きコンテナを読み取ります。

2 番目のプログラムが **EXEC CICS LINK** コマンドにより呼び出された場合、コンテナを呼び出し側プログラムに返すこともできます。この処理は、新しいコンテナを作成するか、または既存のコンテナを再利用することにより実行できます。

チャンネルとコンテナは、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されるか、または (**EXEC CICS FETCH CHANNEL** コマンドの場合) そのチャンネルをフェッチするプログラムでのみ認識されます。これらのプログラムが終了すると、CICS は自動的にコンテナとそのストレージを破棄します。その前に、プログラムでチャンネルとコンテナを削除するコマンドを発行することもできます。

AMODE (64) プログラムでは、**EXEC CICS PUT64 CONTAINER** コマンドおよび **EXEC CICS GET64 CONTAINER** コマンドを使用して、同様の方法でチャンネルおよびコンテナを介して 64 ビット・ストレージ内のデータを転送することができます。これらコマンドは、非 Language Environment® (LE) AMODE(64) アセンブリ言語アプリケーション・プログラムのみで使用するためのものです。また、CICS ビジネス・トランザクション・サービス (BTS) コンテナはサポートされていません。

チャンネル・コンテナはリカバリー可能ではありません。RETURN TRANSID CHANNEL() で始まる疑似会話型トランザクションは、再始動できません。リカバリー可能なコンテナを使用する必要がある場合は、CICS Business Transaction Services (BTS) コンテナを使用します。

z/OS で実行されているが、CICS の外側で実行されているプログラムは、連絡域を使用する代わりに、外部 CICS インターフェース (EXCI) のチャンネルとコンテナを使用して CICS プログラムにデータを渡すことができます。サポートされるコマンドは次のとおりです。**EXEC CICS DELETE CHANNEL**、**EXEC CICS DELETE CONTAINER**、**EXEC CICS GET CONTAINER**、**EXEC CICS MOVE CONTAINER**、および **EXEC CICS PUT CONTAINER**。EXCI では、**EXEC CICS PUT64 CONTAINER** コマンドと **EXEC CICS GET64 CONTAINER** コマンドはサポートされません。

EXCI ジョブで使用されるチャンネルとコンテナは、ジョブの終了時に解放されます。ただし、チャンネルが不要になった時点で、ジョブで **EXEC CICS DELETE CHANNEL** コマンドを発行して、チャンネルとそのコンテナのセットを削除することが適切なプログラミング手法といえます。そうしないと、同じジョブ内でそれ以降に EXCI が使用された場合、チャンネルとそのコンテナにアクセスします。

トランザクション・チャンネル **DFHTRANSACTION**

チャンネルは通常、リンク・レベルが変更されると、有効範囲から外れます。そのため、トランザクション内のすべてのプログラムで使用できなくなることがあります。チャンネルを **DFHTRANSACTION** という名前で作成する場合、そのチャンネルは、リンク・レベルが変更されても、有効範囲から外れません。そのため、トランザクション内のすべてのプログラム (API が有効な出口点を含む) で使用できます。ただし、CICS TS 5.2 より前のバージョンでは、トランザクション・チャンネルを CICS 領域に渡すことはできません。

トランザクション・チャンネルは、MRO および IPIC リンク・タイプでのみ使用できます。

トランザクション・チャンネルは、SNA over SNA リンク・タイプではサポートされません。

DFHTRANSACTION は、チャンネル名を受け入れるすべての API コマンドで使用できます。

トランザクション・チャンネルを削除することはできません。したがって、データの使用をクリーンアップする場合は、チャンネルを削除するのではなく、チャンネルからコンテナを削除してください。

トランザクション・チャンネルは、DPL 要求にシップされます。トランザクション・チャンネルは他の機能シップされた要求にはシップされないため、ターゲット領域上の出口で使用することはできません。

分散プログラム・リンク (DPL) 呼び出しをリモート・プログラムに対して行うトランザクション・プログラムが存在する場合、**DFHTRANSACTION** チャンネルを使用するには、DPL スタック内のフロントエンド・プログラムは、空のチャンネルであってもチャンネルを作成するプログラムでなければなりません。

チャンネルとコンテナを使用する EXCI ジョブでは、トランザクション・チャンネル **DFHTRANSACTION** を使用できます。 **EXEC CICS LINK** コマンドまたは呼び出しレベル **DPL_REQUEST** で指定されたチャンネルとそのコンテナ・セットに加えて、トランザクション・チャンネルとそのコンテナ・データが CICS にシップされます。トランザクション・チャンネルは、**EXCI EXEC CICS LINK** コマンドまたは呼び出しレベル **DPL_REQUEST** の **COMMAREAS** にはシップされていません。

EXCI クライアントでトランザクション・チャンネルを使用する場合は、CICS サーバー・タスクではなく、EXCI クライアントでチャンネルを作成する必要があります。トランザクション・チャンネルには、EXCI ジョブの存続期間があります。これは、CICS サーバー上で、トランザクション・チャンネルが別の CICS システムからの DPL 要求によってシップされた場合と同様に動作します。

EXCI ジョブでトランザクション・チャンネルが CICS TS 5.2 より前のバージョンの CICS 領域に渡される場合、CICS プログラムは異常終了コード AXGA で異常終了します。

基本的な例

チャンネルを作成して 2 番目のプログラムに渡すプログラムの簡単な例

124 ページの図 26 に、以下を行う COBOL プログラム CLIENT1 を示します。

1. PUT CONTAINER(コンテナ名) CHANNEL(チャンネル名) コマンドを使用して、inqcustrec と呼ばれるチャンネルを作成し、2 つのコンテナ 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(チャンネル名) コマンドを使用して、プログラム SERVER1 にリンクし、inqcustrec チャンネルを渡します。
3. GET CONTAINER(コンテナ名) CHANNEL(チャンネル名) 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 チャンネルの custrec コンテナにあります。

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を表しています。チャンネルおよびコンテナは (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、これらの行を両方のプログラムで使用されるコピーブックに組み込むと、使用されるコンテナの名前が記録されるため、プログラムの保守が容易になります。

推奨

チャン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、使用するコンテナの名前を記録して、そのコンテナ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、使用するチャンネルおよびコンテナを指定します。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 26. 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

125 ページの図 27 に、CLIENT1 によってリンクされた 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 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。チャンネルが明示的に指定されていない場合、現行チャンネル (つまり、プログラムを起動したチャンネル) 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 27. データが渡されたチャン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、124 ページの図 26 で示すように、プログラム CLIENT1 からリンクされています。

チャンネルの使用: いくつかの典型的なシナリオ

チャンネルおよびコンテナは、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。このシナリオでは、チャンネルの使用法のいくつかの例を示します。

1 つのチャンネルに 1 つのプログラム

この例は、単一チャンネルを使用した独立型プログラムを示しています。

図 28 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 28. 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)

この例は、単一チャネルから呼び出される関連したプログラム (コンポーネント) のセットを示します。

図 29 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。

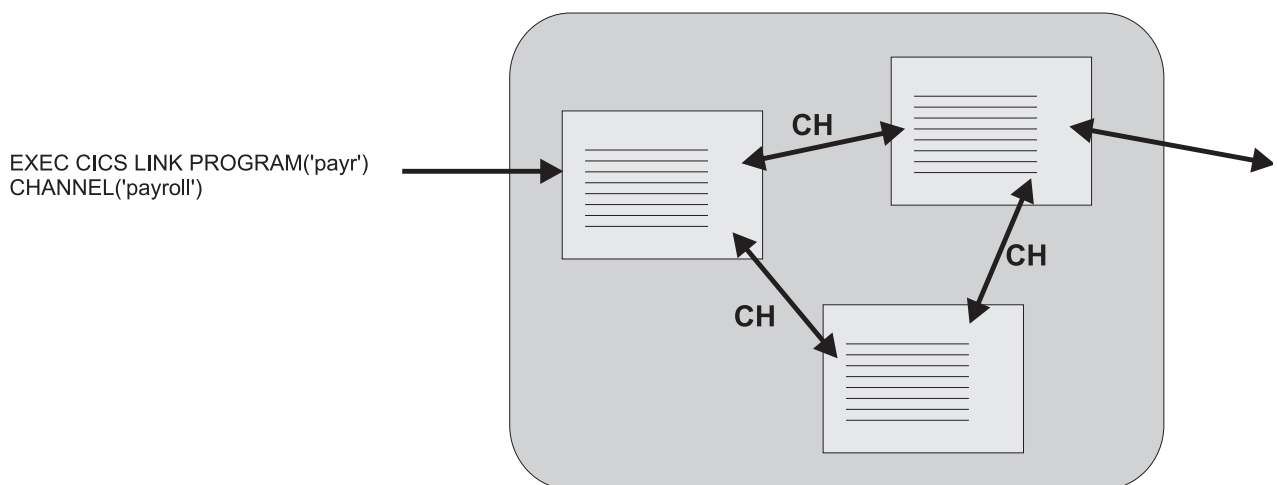


図 29. 『コンポーネント』 - 単一外部チャネルを経由して起動される関連プログラムのセット : 『CH』 は、コンポーネント内の複数プログラムが、それらのプログラム間でチャネルを受け渡すことができることを示します。

複数のチャンネルに 1 つのコンポーネント

この例は、2 つの代替チャンネルから呼び出すことができる関連したプログラム (コンポーネント) のセットを示します。

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 つの代替外部チャン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ンネルを判別し、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 番目のチャンネルの異なるコンテナ・セットから構成される、3 番目のチャンネルを処理するようにします。次に、3 番目のチャン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。

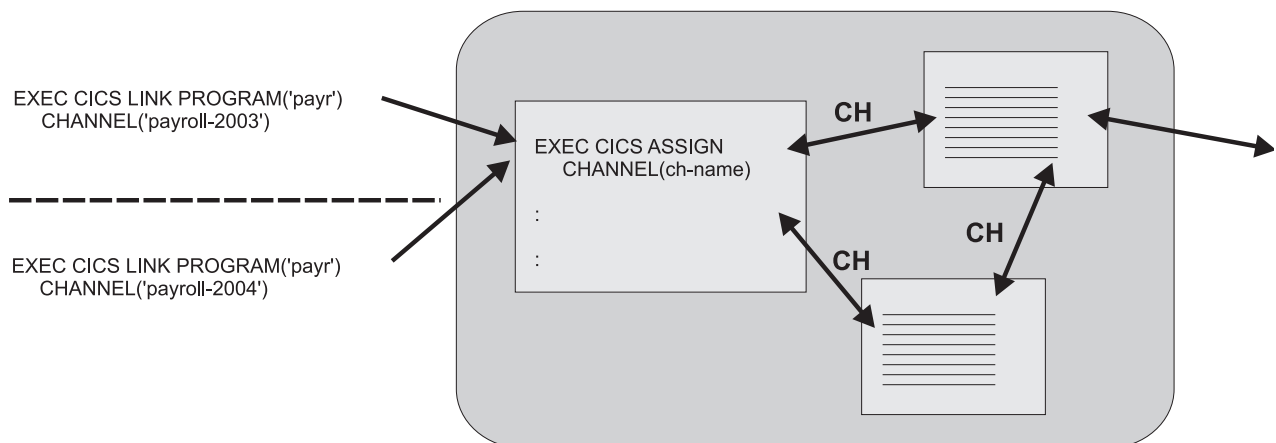


図 30. 同一コンポーネントへの複数の外部チャンネル：『CH』は、コンポーネント内の複数プログラムが、それらのプログラム間でチャンネルを受け渡すことができることを示します。

複数の対話式コンポーネント

この例は、複数のコンポーネントがチャンネルを経由して対話する方法を示します。

128 ページの図 31 に、「人的要因」コンポーネントと「給与計算」コンポーネントを示します。これらの各コンポーネントには、呼び出しに使用できるチャンネルがあります。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。

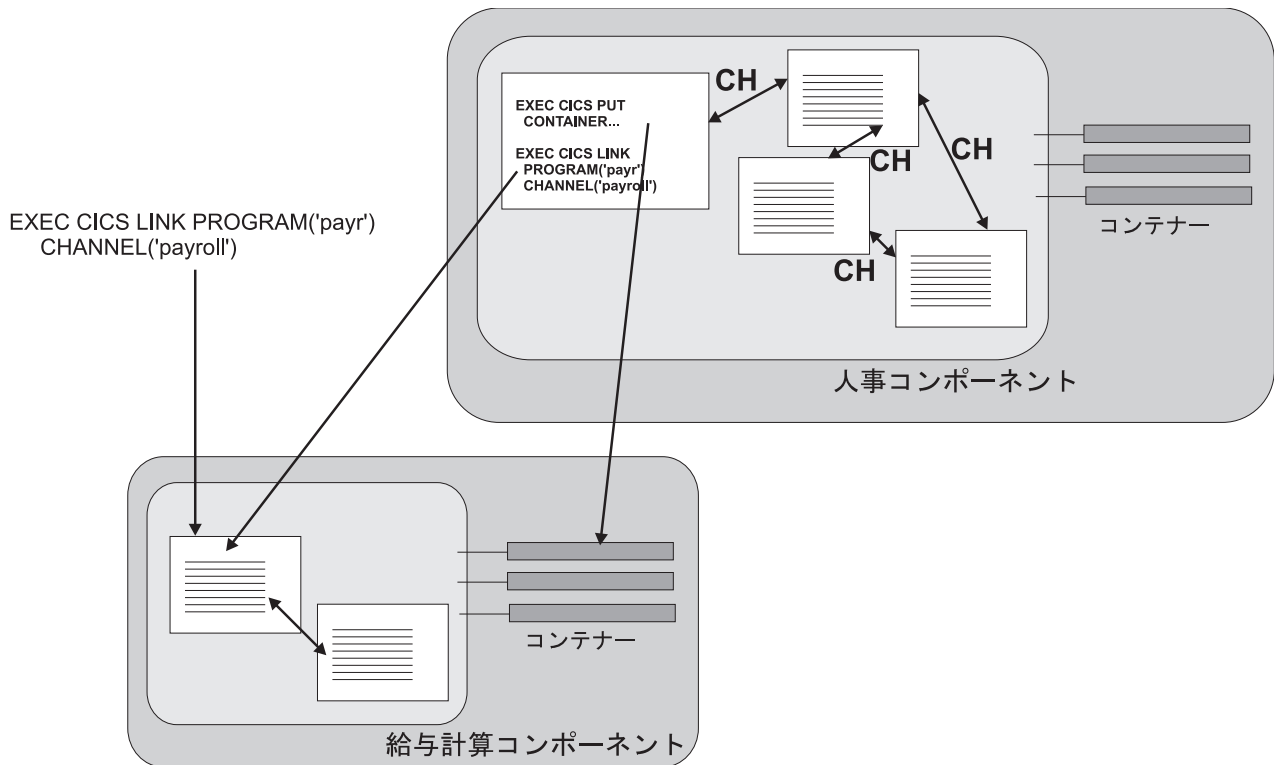


図 31. チャンネルを経由して対話する複数のコンポーネント

チャンネルの利点

CICS プログラムでチャンネル・モデルおよびコンテナ・モデルを使用してデータを交換する場合は、連絡域 (COMMAREA) を使用する場合に比べ、いくつかの利点があります。

- チャンネルに追加できるコンテナの数に制限はなく、個々のコンテナのサイズにも、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に注意してください。

- チャンネルには複数のコンテナを含めることができるので、構造化された方法でデータを渡すことができます。
- 返されるデータの正確なサイズを知るために、チャンネルを使用するプログラムは必要ありません。
- チャンネルは 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。チャンネルは、**LINK**、**START**、**XCTL**、**RUN TRANSID** および **RETURN** の各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。
- リンク・レベルが変更されても、トランザクション・チャンネル **DFHTRANSACTION** は有効範囲に入ったままになります。そのため、これを使用して、トランザクション内のどのプログラム間でもデータを交換することができます。

- CICS がサポートしているどの言語で書かれた CICS アプリケーション・プログラムでもチャンネルを使用できます。例えば、ある CICS 領域上の Java クライアント・プログラムが、チャンネルを使用して、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。
- 複数のチャン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。例えば、以下のことができます。
 - 一緒に呼び出されたチャンネルの動的な検出。
 - チャンネル内のコンテナのカウンタおよび参照。
 - 渡されたチャンネルに応じた処理の変更。
- 1 つ以上のチャン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ン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) 新規チャン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 (およびそのストレージ) は自動的に破棄されます。プログラムは、個別のコンテナ、または 1 つのチャンネル全体とそれに属するすべてのコンテナを、それらが有効範囲から外れる前に削除するコマンドを発行することもできます。
- チャンネル・アプリケーションで使用されるデータ変換モデルは、シンプルであり、アプリケーション開発者が簡易な API コマンドを使用して制御することができます。
- CICS Business Transaction Services (BTS) に関して経験のあるプログラマーは、非 BTS アプリケーションでも容易にコンテナを使用できます。
- コンテナを使用するプログラムは、チャンネルと 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)。

チャンネルが、あらゆるケースで最良のソリューションとなるわけではありません。アプリケーションを設計する際は、チャンネルを使用することによって以下の 1 つまたは 2 つの影響があることを考慮に入れておく必要があります。

- チャンネルがリモート・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 ではなく ISC によって接続されている場合に当てはまります。
- コンテナ・データが複数の場所で保持されることがあるため、チャンネルは、同じ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ります。COMMAREA はポインターによってアクセスされるが、コンテナ内のデータはプログラム間でコピーされる。

チャネルの作成

いくつかの API コマンドのいずれかでチャネルを指名することによって、チャネルを作成できます。 現行プログラムのスコープ内にチャネルが存在しない場合は、CICS によって作成されます。

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。

```
LINK PROGRAM CHANNEL
MOVE CONTAINER CHANNEL TOCHANNEL
PUT CONTAINER CHANNEL
PUT64 CONTAINER
RETURN TRANSID CHANNEL
START TRANSID CHANNEL
XCTL PROGRAM CHANNEL
WEB RECEIVE TOCHANNEL
WEB CONVERSE TOCHANNEL
RUN TRANSID CHANNEL
```

チャネルを作成し、そのチャネルにデータのコンテナ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。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を追加します。その後続くコマンドは、チャネルにさらにコンテナを追加します。そのコンテナが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。

AMODE (64) プログラムでは、**EXEC CICS PUT64 CONTAINER** コマンドを使用して、同様の方法でチャネルを作成して使用することができます。このコマンドは、言語環境 (LE) プログラム以外の AMODE(64) アセンブラー言語アプリケーション・プログラム専用です。

注: 新しいチャネルはリンク・レベルが変更されるまで有効範囲に入ったままになります。ただし、DFHTRANSACTION という名前のトランザクション・チャネルを除きます。リンク・レベルが変更されても、トランザクション・チャネルは有効範囲から外れません。これは、そのトランザクションで常にアクセス可能です。チャネルの有効範囲について詳しくは、136 ページの『チャネルの有効範囲』を参照してください。

CICS が API の一部として定義するチャネルを除き、チャネルに *DFH* という文字で始まる名前は付けられません。これらのチャネルは、その API によって規定されているとおりにしか使用できません。

チャネルにコンテナを追加するもう 1 つの方法は、追加するコンテナ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、現行チャンネルであることを意味する。
2. ソース・チャンネルはプログラムのスコープ内にある必要がある。
3. ターゲットのチャンネルが、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・コンテナが存在しない場合は、エラーが発生します。
5. ターゲットのコンテナ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。
6. チャンネルが作成されると、それは有効範囲から外れるまで存在する。それに関連付けられているストレージも、コンテナまたはセット・ストレージのどちらであっても、チャンネルが有効範囲から外れるまで存在します。

チャン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ン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、空のチャン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)
- EXEC CICS RUN TRANSID CHANNEL(channel-name)

現行チャンネル

プログラムの現行チャンネルとは、そのプログラムの呼び出しに使用されたチャンネルがあれば、そのチャンネルを指します。次の例は、プログラム間で現行チャンネルとそのコンテナがどのように渡されるのかを示します。

プログラムはその他のチャン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ン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ンネルの例

この例は、プログラムが現行チャンネルとそのコンテナを、EXEC CICS LINK コマンドを使用して別のプログラムに渡す方法を示します。

以下の図は、プログラムの現行チャン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、端末ユーザーなどによって開始される最上位のプログラムです。このプログラムは、プログラムからは始動されず、現行チャンネルも持っていません。

B、C、D、および E は、それぞれ第 2 レベル、第 3 レベル、第 4 レベル、および第 5 レベルのプログラムです。

プログラム B の現行チャンネルは、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、チャンネル X にコンテナを 1 つ追加し、別のコンテナを削除するという変更を加えます。

プログラム C の現行チャンネルも、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 からプログラム D に現行チャンネルが渡されていないため、プログラム D には現行チャンネルがありません。

プログラム E の現行チャンネルは、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。

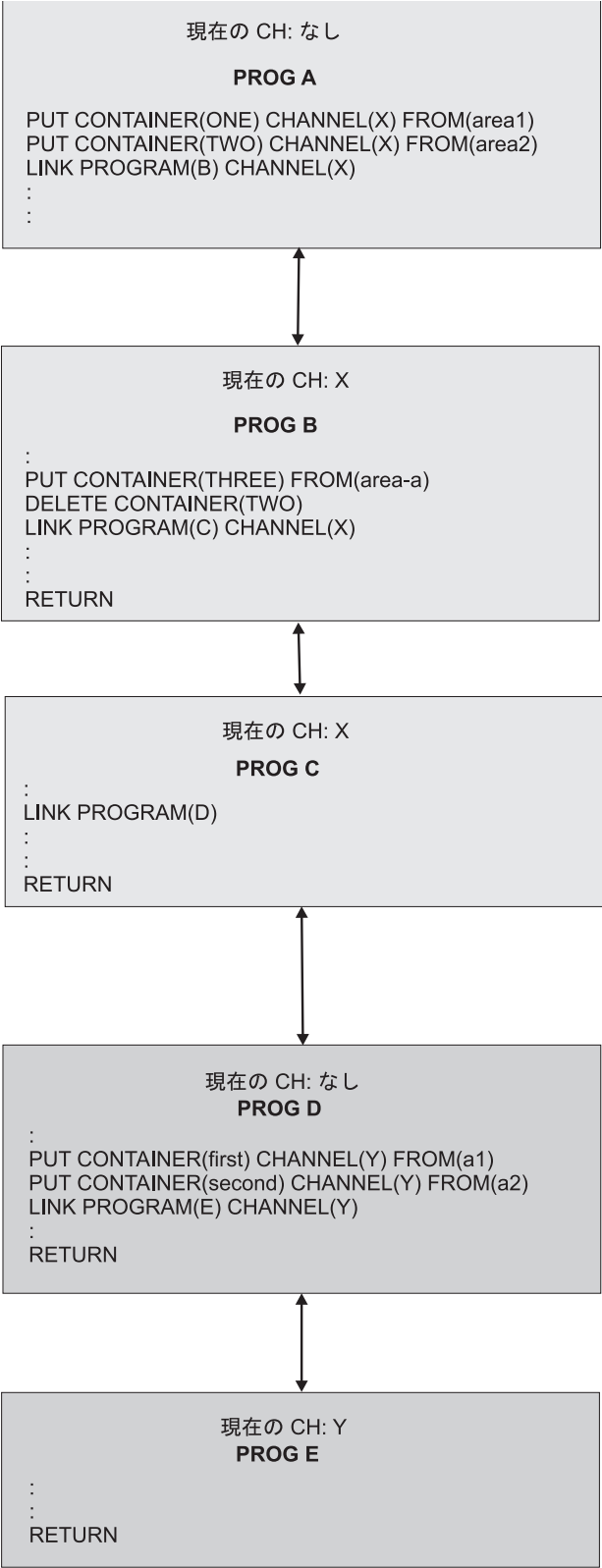


図 32. 現行チャンネル: LINK コマンドを使用した例

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ンネル (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ンネル - LINK コマンドを使用した例

プログラム	現行チャンネル	コマンドの発行	コメント
A	なし	<pre> ・ 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) ・ </pre>	<p>プログラム A はチャンネル X を作成し、プログラム B に渡します。</p> <p>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 チャンネルは変更される場合があります。ご注意ください。プログラム A によって作成されたときと同じコンテナ・セットが X チャンネルに含まれているとは限りません。(コンテナ TWO は削除され、コンテナ THREE はプログラム B によって追加されています。)</p>
B	X	<pre> ・ EXEC CICS PUT CONTAINER(THREE) FROM(area-a) EXEC CICS DELETE CONTAINER(TWO) EXEC CICS LINK PROGRAM(C) CHANNEL(X).. EXEC CICS RETURN </pre>	<p>プログラム B は、コンテナの追加および削除によってチャンネル X (そのプログラムの現行チャンネル) に変更を加え、変更済みのチャンネルをプログラム C に渡します。</p> <p>プログラム B は、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。プログラムの現行チャンネルが暗黙指定されます。</p>
C	X	<pre> ・ EXEC CICS LINK PROGRAM(D).. EXEC CICS RETURN </pre>	<p>プログラム C はプログラム D とリンクしますが、チャンネルは渡しません。</p>
D	なし	<pre> ・ 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 </pre>	<p>プログラム D はチャンネル Y を新規作成し、プログラム E に渡します。</p>
E	Y	<pre> ・ RETURN ・ </pre>	<p>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。</p>

XCTL コマンドを使用した現行チャンネルの例

この例は、プログラムが現行チャンネルとそのコンテナを、EXEC CICS XCTL コマンドを使用して別のプログラムに渡す方法を示します。

135 ページの図 33 に、4 つの対話式プログラムを示します。A1 は、端末ユーザーなどによって開始される最上位のプログラムです。このプログラムは、プログラムからは始動されず、現行チャン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 の現行チャンネルは、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 から現行チャンネルが渡されていないため、B2 には現行チャンネルはありません。

B3 の現行チャンネルは、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、チャンネル Y およびそのコンテナは、CICS によって削除されます。

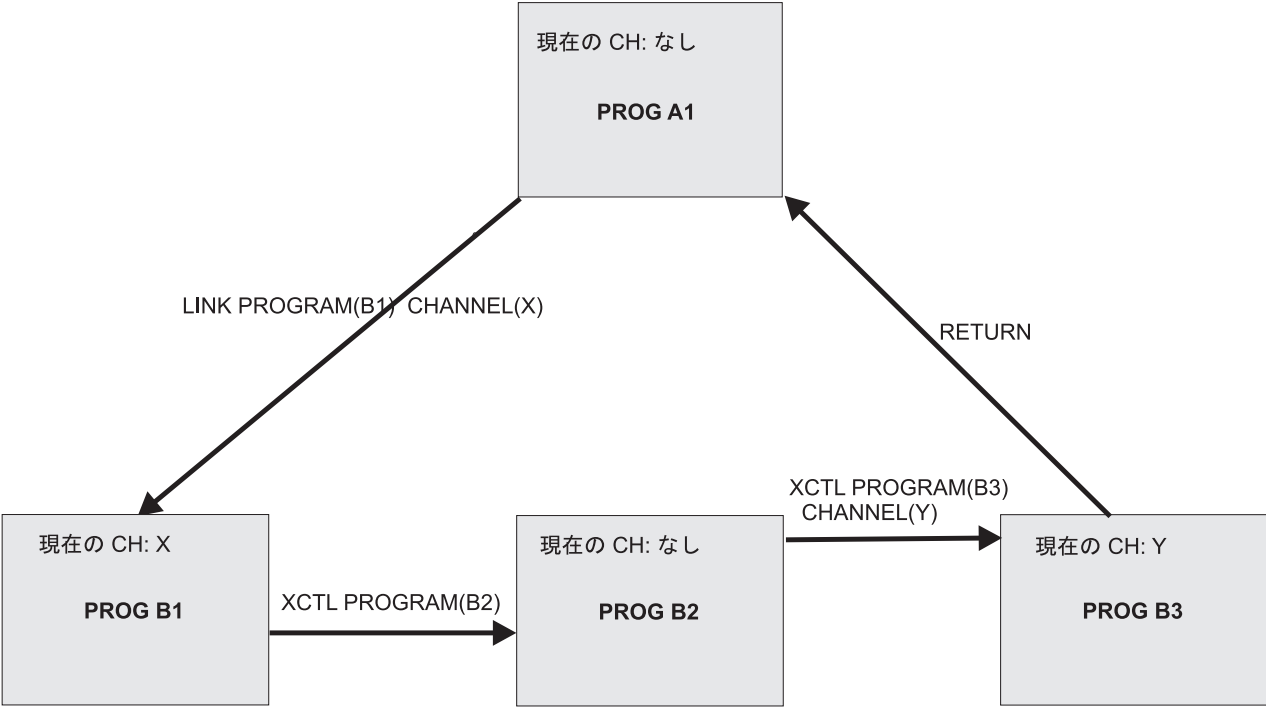


図 33. XCTL コマンドを使用した現行チャンネルの例

以下のテーブルには、図 33 で示された 4 つのプログラムそれぞれの現行チャンネル (もしあれば) の名前がリストされています。

表 14. 対話式プログラムの現行チャンネルの例

プログラム	現行チャンネル	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 .

現行チャンネル: START、RUN TRANSID および RETURN コマンド
LINK コマンドおよび **XCTL** コマンドと同様に、**START**、**RUN TRANSID**、**RETURN** の各コマンドでチャンネルを渡すことができます。

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) がチャンネルに渡され、そのチャンネルがそのプログラムの現行チャンネルになります。

注: チャンネルを指定した EXEC CICS START 要求の場合はいずれも、INTERVAL、AT、FOR、UNTIL がサポートされていないので、これらのオプションを指定することでその要求を据え置くことはできません。この動作は、JCICS ではなく CICS 変換プログラムによってモニターされます。これらのコマンドを使用して EXEC CICS START 要求を据え置こうとすると、変換プログラムのエラーになります。

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。
- 論理レベルで最高位のプログラム、すなわち、CICS に制御を返すプログラムが発行した場合。

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ンネルに渡され、そのチャンネルがそのプログラムの現行チャンネルになります。

チャンネルの有効範囲

チャンネルの有効範囲は、チャンネルにアクセスできるコードです。この例は、図の各チャンネルの有効範囲を示しています。

この有効範囲は、チャンネルおよびコンテナ保管の存続時間を定義するため、重要です。詳しくは、142 ページの『チャンネルとコンテナの削除とそのストレージの解放』を参照してください。LINK コマンドについて詳しくは、131 ページの『LINK コマンドを使用した現行チャンネルの例』を参照してください。

LINK コマンドを使用した有効範囲の例

X チャンネルの有効範囲はプログラム A、B、および C です。Y チャンネルの有効範囲はプログラム D と E です。

これらのチャンネルはいずれも、トランザクション・チャンネル **DFHTRANSACTION** ではありません。**DFHTRANSACTION** の有効範囲はトランザクション全体です。

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 チャンネルが変更されました。プログラム A によって作成されたときと同じコンテナ・セットが X チャンネルに含まれているとは限りません。

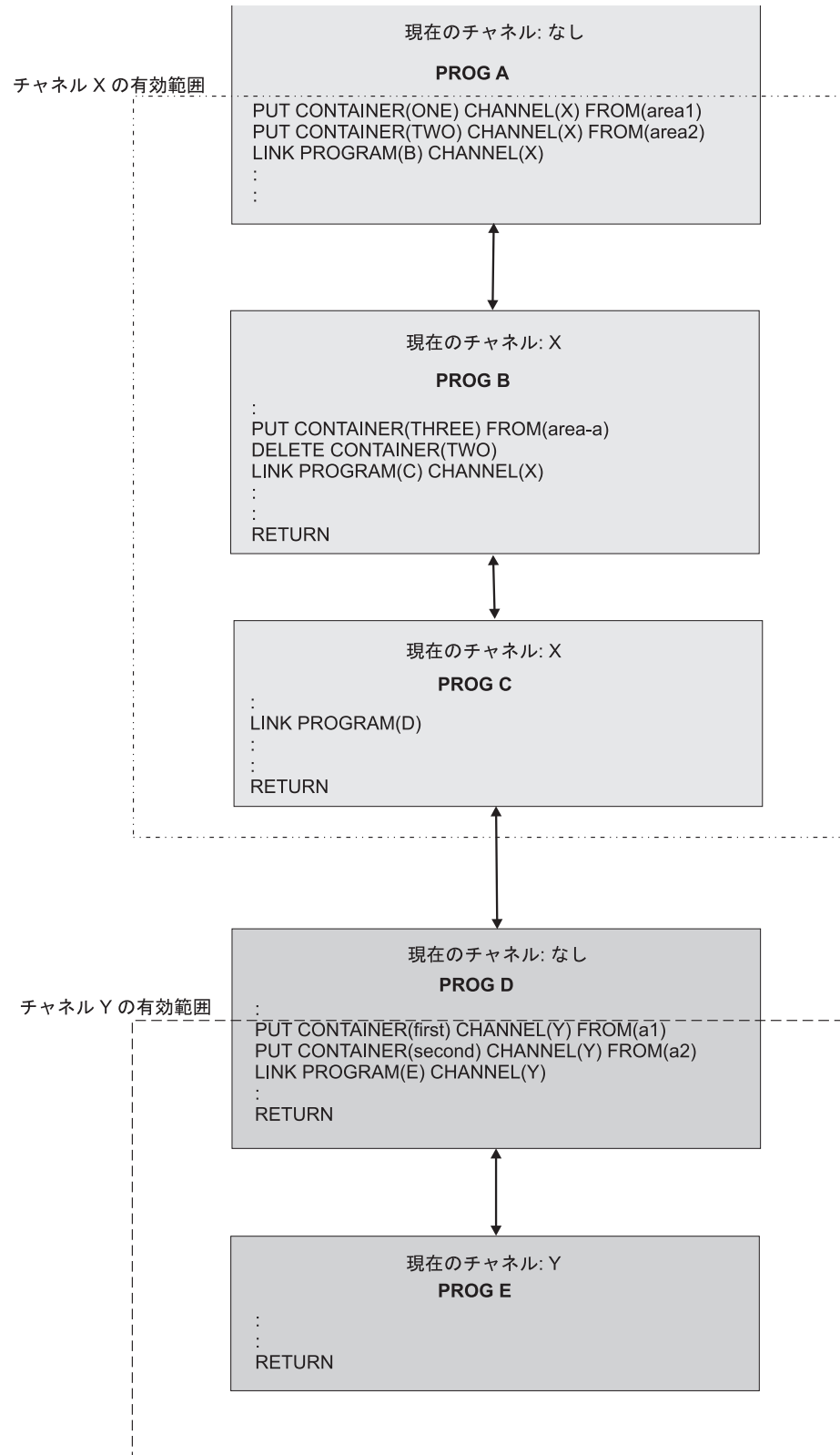


図 34. チャンネルの有効範囲 - LINK コマンドの例

次の表に、前述した 5 つの各プログラムの現行チャンネル (存在する場合) について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 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	適用外
B	X	A、B、C
C	X	A、B、C
D	なし	適用外
E	Y	D、E

トランザクション・チャネル DFHTRANSACTION

チャネルは、リンク・レベルが変更されると、LINK コマンドで渡されるまで有効範囲から外れます。これにより、チャネルはトランザクション内のすべてのプログラムで使用できなくなります。チャネルが有効範囲から外れないようにするために、予約名 DFHTRANSACTION でチャネルを指定して PUT CONTAINER コマンドを発行し、トランザクション・チャネルを作成できます。

トランザクション・チャネルは、CHANNEL キーワードをサポートするすべてのコマンドで使用できます。ただし、DELETE CHANNEL コマンドは、CICS によりトランザクション・チャネルを削除するタイミングが制御されるため除外されます。データを破棄するタイミングは、チャネル自体を削除するのではなく、トランザクション・チャネルからコンテナを削除することで制御できます。

CICS トランザクションごとに使用できるトランザクション・チャネルは 1 つのみです。このトランザクション・チャネルは、ローカル領域内のトランザクションの一部として実行中のすべてのプログラム (API が有効な出口点を含む) で使用できます。

トランザクション・チャネルは、以下の仕様を満たしている限り、LINK コマンドに指定された commarea またはチャネルとともにリモート領域に渡されます。

- 2 つのシステム間のリンクが MRO または IPIC である。ISC over SNA を使用するシステムはサポートされません。
- ターゲット領域が CICS TS 5.2 以上である。これにより、トランザクション・チャネルがサポートされます。
- 要求が DPL 要求 (つまり、機能シッパされた LINK PROGRAM 要求) である。トランザクション・チャネルは、他の機能シッパされた要求 (機能シッパされたファイル制御要求など) ではリモート領域に渡されません。

トランザクションで、トランザクション・チャネルを複数領域環境で使用する必要がある場合は、トランザクション・チャネルを使用する必要がある最初の領域でこのトランザクション・チャネルを作成する必要があります。例えば、領域 A の DPL から領域 B にわたって使用する場合は、トランザクション・チャネルを領域 A で作成する必要があります。

LINK コマンドおよび XCTL コマンドを使用した有効範囲の例

この例は、各チャネルの有効範囲を示すために、『XCTL コマンドを使用した現行チャネルの例』の図に追加して示されるものです。

XCTL コマンドについて詳しくは、134 ページの『XCTL コマンドを使用した現行チャンネルの例』を参照してください。

140 ページの図 35に、以前に説明した 4 つの同一の対話式プログラムと、プログラム B1 から EXEC CICS LINK コマンドにより呼び出された第 3 レベルのプログラム C1 を示します。

X チャンネルの有効範囲は、A1 と B1 に制限されます。

Y チャンネルの有効範囲は、B2 と B3 です。

Z チャンネルの有効範囲は、B1 と C1 です。

これらのチャンネルはいずれも、トランザクション全体が有効範囲となるトランザクション・チャンネル DFHTRANSACTION ではありません。

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 チャンネルはプログラム 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・セットが X チャンネルに含まれているとは限りません。

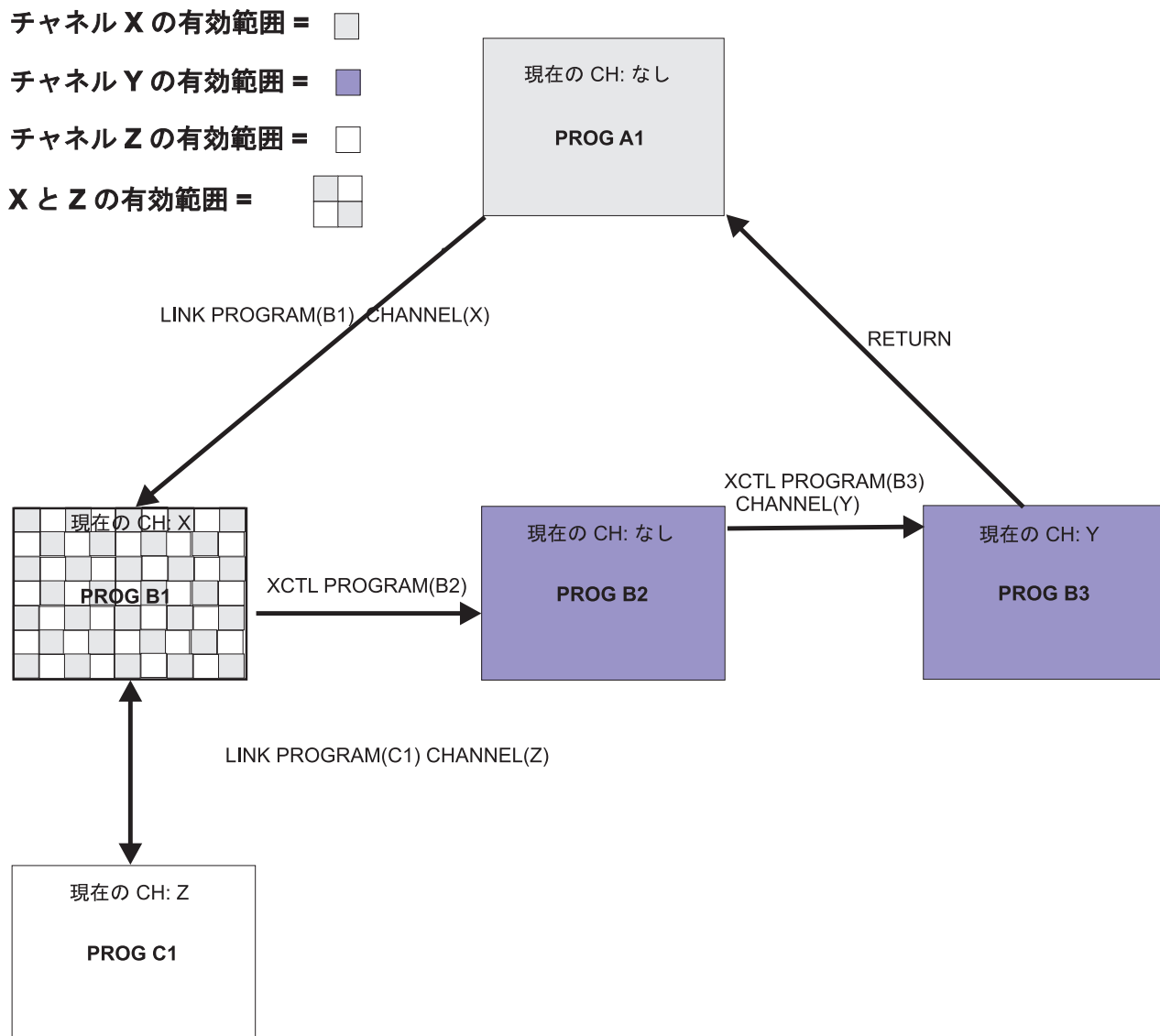


図 35. チャンネルの有効範囲 - LINK コマンドと XCTL コマンドを示す例

以下のテーブルに、図 35 で示した 5 つの各プログラムの現行チャンネル (ある場合) の名前と有効範囲をリストします。

表 16. チャンネルの有効範囲 - LINK と XCTL コマンドの例

プログラム	現行チャンネル	チャンネルの有効範囲
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の検出

プログラムが呼び出されたとき、チャンネルの名前、およびチャンネルに渡されたコンテナの名前を判別できます。

通常、チャンネルを交換するプログラムは、そのチャン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、チャンネルの名前、およびチャンネル内のコンテナ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ンネルを処理するように作成されている場合、それらのチャン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ンネル（つまり、起動に使用したチャンネル）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（現行チャンネルがない場合、コマンドはブランクを返します）。

プログラムは、EXEC CICS QUERY CHANNEL コマンドを発行することで、現行チャンネル内のコンテナの数をカウントできます。このコマンドにはどのチャンネルに対しても使用できるので、プログラムはまず現行チャンネルの名前を検出して、それをコマンドで指定する必要があります。プログラムは、ブラウズにより現行チャンネル内のコンテナの名前を取得することもできます。通常、これらの操作は必要ありません。複数のチャンネルを処理するよう作成されたプログラムは、それらの各チャンネル内のコンテナ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。

現行チャンネル内のコンテナ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、現行チャンネルの名前、および必要に応じてチャンネル内のコンテナの数と名前を取得すると、渡されたデータの種類の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の検出

LINK コマンドの後で、プログラムはリンク先のプログラムによって返されたコンテナの名前を検出できます。

プログラムはチャンネルを作成し、そのチャンネルを EXEC CICS LINK PROGRAM(*program-name*) CHANNEL(*channel-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、結果を同じチャンネル（現行チャンネル）に返します。返されるとすぐに、最初のプログラムは返されたチャンネルの名前を認識しますが、チャンネル内のコンテナの数と名前は必ずしも認識するわけではありません。（返されたチャンネルには、渡されたチャンネルと同じコンテナが含まれるか、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 EXEC CICS QUERY CHANNEL コマンドを発行して現行チャンネルの名前を指定することによって、チャンネル内のコンテナの数をカウントで

きます。最初のプログラムは、ブラウザによりコンテナ名を検出することもできます。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*).

チャネルとコンテナの削除とそのストレージの解放

コンテナがラージ・オブジェクトである場合もあるので、CICS 領域がストレージ不足にならないように、それらのオブジェクトのストレージを正しく管理することが重要です。以前にアクセスしていたコンテナやチャネルをアプリケーションが使用できなくなると、CICS はそれらのコンテナやチャネルを自動的に削除します。個別のコンテナまたはチャネル全体を削除するコマンドを、アプリケーションから発行することもできます。

アプリケーションが初めてコマンドで名前を指定してチャネルを作成し、そこにコンテナを追加すると、CICS はコンテナに保持されるデータ用に 64 ビット・ストレージ (2 GB より上) を使用します。チャネルとコンテナに 64 ビット・ストレージを使用すると、CICS 領域に適用される z/OS **MEMLIMIT** パラメーターに選択する値に影響します。また、64 ビット・ストレージを使用する他の CICS 機能も許可する必要があります。詳しくは、「パフォーマンスの改善」の『**MEMLIMIT** の見積もり、確認、および設定』を参照してください。

アプリケーション・プログラムが **SET** オプションを指定した **GET CONTAINER** コマンドを発行すると、CICS はそのプログラムのリソース定義の **DATALLOCATION** 設定に従って、24 ビット・ストレージ (16 MB より下) または 31 ビット・ストレージ (16 MB より上で 2 GB より下) のいずれかのタスク・ストレージにコンテナのコピーを作成します。アプリケーション・プログラムは、コンテナの内容を処理してから **PUT CONTAINER** コマンドを使用してその内容を元の 64 ビット・ストレージ (2 GB より上) にコピーすることができます。非 Language Environment (LE) AMODE(64) アセンブリ言語アプリケーション・プログラムは、**PUT64 CONTAINER** コマンドと **GET64 CONTAINER** コマンドを使用して、64 ビット・ストレージ内のチャネルとコンテナを直接処理できます。

大きなコンテナ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に注意してください。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に入れるのではなく、複数のコンテナに分割する。コンテナを作成する際、その操作によってトランザクションに割り振られる 64 ビット・ストレージの合計が **MEMLIMIT** 値の 5% を超えることにならないかどうかを CICS は検査します。その状態では、CICS は警告メッセージ DFHPG0400 を発行し、コンテナを作成しません。

コンテナの削除とそのストレージの解放

CICS では、チャネルがリンク・スタック内のすべてのプログラムの有効範囲から外れると、チャネル上にあるすべてのコンテナの 64 ビット・ストレージ (2 GB より上) が自動的に解放されます。チャネルの有効範囲は、チャネルにアクセスできるコードで、チャネルを作成したプログラムとチャネルが渡されたプログラムを含みます。例外はトランザクション・チャネル DFHTRANSACTION で、これはトラ

ンザクション内のすべてのプログラムで使用可能であり、リンク・レベルが変更されても有効範囲から外れません。チャンネルの有効範囲について詳しくは、136 ページの『チャンネルの有効範囲』を参照してください。

チャンネルが有効範囲から外れる前にコンテナを削除する場合は、アプリケーション・プログラムで **DELETE CONTAINER** コマンドを発行して、指定したコンテナを明示的に削除します。このコマンドを発行すると、チャンネル内に含まれるすべてのデータが破棄され、コンテナに使用されていた 64 ビット・ストレージが解放されます。チャンネルを所有するアプリケーション・プログラムは、**DELETE CHANNEL** コマンドを発行して、チャンネル内にあるすべてのコンテナを削除することもできます。

プログラムが **GET CONTAINER** コマンドを発行し、CICS がコンテナのコピーを作成してそのプログラムで使えるようにすると、以下のいずれかが発生するまで、CICS はコンテナのそのコピーを 24 ビット・ストレージまたは 31 ビット・ストレージで維持します。

- 同じチャンネル内の同じコンテナに対して、SET オプションが指定された後続の **GET CONTAINER** コマンドまたは **GET64 CONTAINER** コマンドが、このストレージにアクセスできるプログラムによって発行された。
- コンテナが **DELETE CONTAINER** コマンドによって削除された。
- コンテナが **MOVE CONTAINER** コマンドによって移動された。
- チャンネルがプログラムの有効範囲外に出た。
- このチャンネルと、その中にあるコンテナが **DELETE CHANNEL** コマンドによって削除されます。

これらのイベントのいずれかが発生すると、24 ビット・ストレージまたは 31 ビット・ストレージ内のコンテナのコピーは削除されます。

チャンネルの削除とそのストレージの解放

CICS は、チャンネルが有効範囲から外れると、そのチャンネルおよびそこにあるすべてのコンテナの 64 ビット・ストレージ (2 GB より上) を自動的に解放します。24 ビット・ストレージまたは 31 ビット・ストレージにまだ存在している、そのチャンネル内のコンテナのすべてのコピーも、その時点で削除されます。

チャンネルが有効範囲から外れる前にチャンネル全体を削除する場合は、アプリケーション・プログラムで **DELETE CHANNEL** コマンドを発行します。このコマンドは、一回の操作でチャンネルに残っているすべてのコンテナを削除します。そのチャンネルとその残りのコンテナに使用されているすべての 64 ビット・ストレージは解放され、チャンネルのコンテナのコピーで 24 ビット・ストレージまたは 31 ビット・ストレージにまだ存在するものもすべて削除されます。

DELETE CHANNEL コマンドを発行するアプリケーション・プログラムは、そのチャンネルを所有するプログラムでなければなりません。チャンネルを所有するプログラムは、以下のいずれかのコマンドでそのチャンネルを指定して作成したプログラムです。

- **LINK PROGRAM CHANNEL**
- **MOVE CONTAINER CHANNEL TOCHANNEL**
- **PUT CONTAINER CHANNEL**

- **PUT64 CONTAINER**
- **RETURN TRANSID CHANNEL**
- **START TRANSID CHANNEL**
- **XCTL PROGRAM CHANNEL**
- **WEB RECEIVE TOCHANNEL**
- **WEB CONVERSE TOCHANNEL**

アプリケーション・プログラムは、以下のチャンネルは削除できません。

- アプリケーション・プログラムの現行チャンネル。つまり、プログラムを呼び出すために使用されたチャンネル。
- そのアプリケーション・プログラムが作成しなかったチャンネル。
- 読み取り専用チャンネル。
- トランザクション・チャンネル DFHTRANSACTION。

チャンネルの設計: 最良事例

コンテナは、プログラム間で情報を渡すために使用されます。これらのコンテナは、チャンネルと呼ばれるセットにグループ化されます。最良事例のセットにならってチャンネルを設計することをお勧めします。

このタスクについて

DPL 呼び出しの終了時、サーバー・プログラムによって変更されていない入力コンテナはクライアントに返されません。内容がサーバー・プログラムによって変更された入力コンテナおよびサーバー・プログラムによって作成されたコンテナは返されます。したがって、DPL について最良のパフォーマンスを実現するには、次のベスト・プラクティスを使用します。

- データの入力と出力で別々のコンテナを使用する。
- 出力コンテナは、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を使用する。
- 構造がオプションの場合は、別のコンテナにする。
- エラー情報には専用のコンテナを使用する。

STARTBROWSE コマンドまたは **GETNEXT** コマンドを使用している場合は、コンテナが返される順序が未定義であり、変更される可能性があることに注意してください。アプリケーションは、返されるコンテナの順序に依存しないようにする必要があります。そのように記述された既存のアプリケーションがある場合は、アプリケーションのアップグレードのアドバイスを参照してください。

以下のチャンネルの設計に関する一般的なヒントでは、最適な DPL パフォーマンスを達成するための推奨事項を示すとともに、さらに詳しく掘り下げます。

- データの入力と出力で別々のコンテナを使用する。これには、以下のような利点があります。
 - 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の流れが小さくなるため、DPL 呼び出しでチャン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を使用する。これには、以下のような利点があります。
 - 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コンテナが戻されないため、CICS 領域間の伝送効率が向上する。
- 各構造体ごとに別々のコンテナを使用する。これには、以下のような利点があります。
 - 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をチャンネル間で移動することによって、チャン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、別のコンテナにする。これにより、コンテナ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を使用する。これには、以下のような利点があります。
 - エラー情報の識別が容易になる。
 - 以下の理由で、効率が向上する。
 - 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。
 - チャンネル内のコンテナの表示を開始するより、**GET CONTAINER**(*known-error-container-name*) コマンドまたは **GET64 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、code page1 の文字データと code page2 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を使用するのではなく、タイプごとに別々のコンテナを使用する。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に入れるのではなく、複数のコンテナに分割する。

チャンネルがリモート・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当てはまります。

重要: 大きなコンテナ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に注意してください。

- チャンネルとコンテナは、2 GB 境界より下のストレージと、2 GB 境界より上の 64 ビット・ストレージを使用する。ストレージを使用および解放する方法に

については、142 ページの『チャンネルとコンテナの削除とそのストレージの解放』を参照してください。チャンネルとコンテナに 64 ビット・ストレージを使用すると、CICS 領域に適用される z/OS **MEMLIMIT** パラメーターに選択する値に影響します。また、64 ビット・ストレージを使用する他の CICS 機能についても考慮する必要があります。詳しくは、「パフォーマンスの改善」の『MEMLIMIT の見積もり、確認、および設定』を参照してください。

チャンネルの構成および使用: 例

この例では、クライアント・プログラムがチャンネルを構成して、サーバー・プログラムに渡し、サーバーの出力を取得します。サーバー・プログラムは、チャンネルのコンテナからデータを取得し、クライアントに出力を返します。

図 36 に、以下を行う CICS クライアント・プログラムを示します。

1. EXEC CICS PUT CONTAINER コマンドを使用して、コンテナのセットを構成 (およびコンテナのセットにデータを追加) します。すべてのコンテナが、同じ名前のチャンネル 『 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 チャンネルを渡します。
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 チャンネルの status コンテナ内に入っていると認識されます。

```
* create the employee container on the payroll-2004
チャンネル (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 36. クライアント・プログラムがチャンネルを構成して、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

147 ページの図 37 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・プログラムでは以下の処理が行われます。

1. サーバー・プログラムを起動するチャンネルを照会します。
2. EXEC CICS GET CONTAINER コマンドを実行して、payroll-2004 チャンネルの employee および wage の各コンテナ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 チャンネルの status コンテナ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 37. 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

CICS の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。これらのコンテナーには、CICS によって読み取り専用のマークが付けられ、その結果、CICS がユーザー・プログラムから戻るときに必要なデータをユーザー・プログラムで変更できなくなることがあります。

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。 **PUT CONTAINER**、**PUT64 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すると、INVREQ 条件が発生します。プログラムで **EXEC CICS DELETE CHANNEL** コマンドを使用して読み取り専用チャネルを削除することはできません。

JCICS からのチャネルの使用

CICS には、CICS Java プログラムがチャネルの受け渡しに使用できる JCICS クラスが用意されています。

JCICS でのチャネルの使用については、チャネルとコンテナーの例を参照してください。

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。

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が使用できます。さらに、コンテナ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 図 38 に示されています。

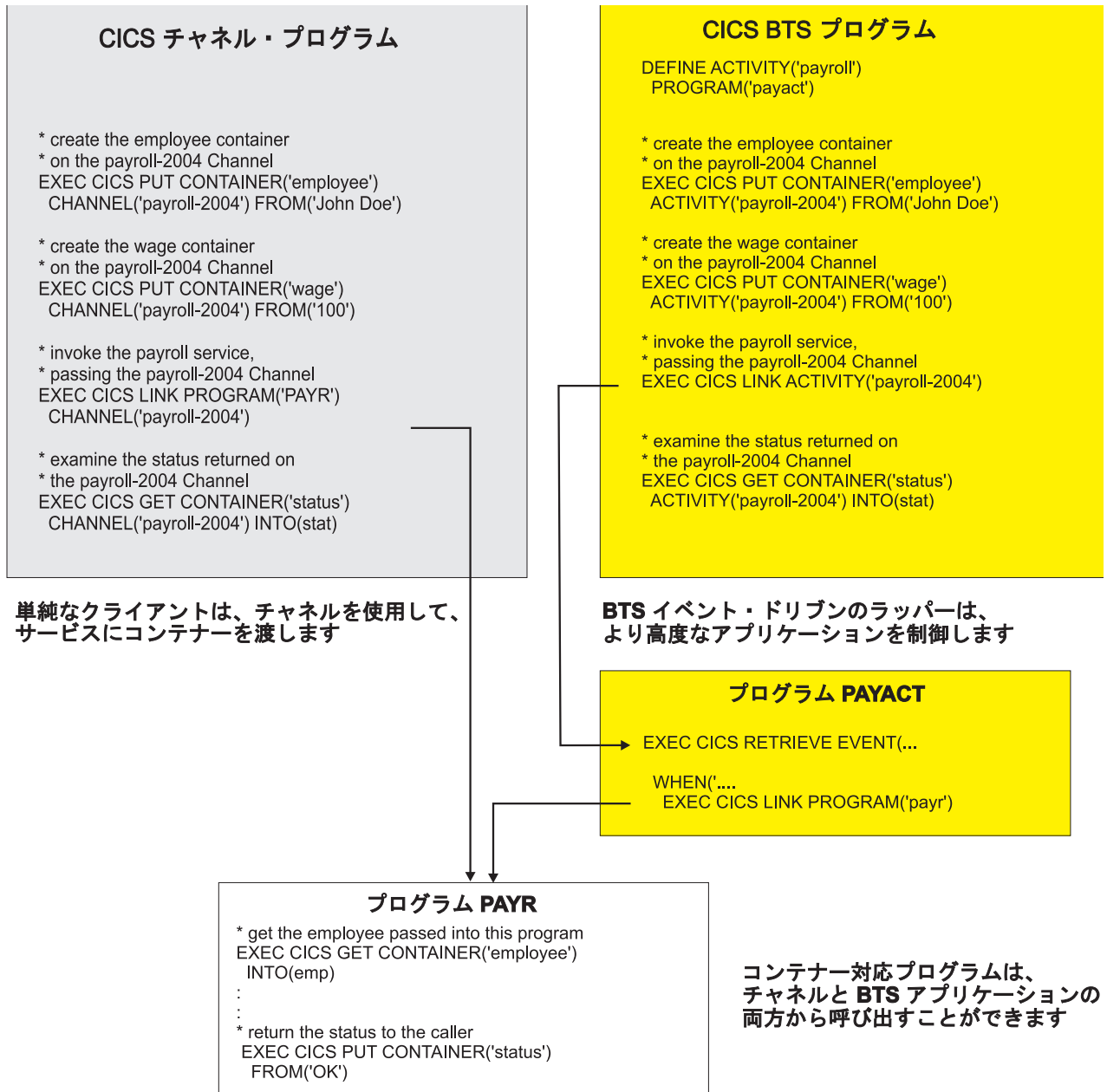


図 38. チャネルと BTS アクティビティー

コンテキスト

コンテナ・コマンドを実行するプログラムを、変更せずに、チャンネル・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。

チャンネルおよび 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・コマンドでは、それらのコマンドをチャンネルまたは 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・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 (チャンネル固有)
- PROCESS (BTS 固有)

GET CONTAINER

- ACQACTIVITY (BTS 固有)
- ACQPROCESS (BTS 固有)
- ACTIVITY (BTS 固有)
- CHANNEL (チャンネル固有)
- INTOCCSID (チャンネル固有)
- PROCESS (BTS 固有)

MOVE CONTAINER

- FROMACTIVITY (BTS 固有)
- CHANNEL (チャンネル固有)
- FROMPROCESS (BTS 固有)
- TOACTIVITY (BTS 固有)
- TOCHANNEL (チャンネル固有)
- TOPROCESS (BTS 固有)

PUT CONTAINER

- ACQACTIVITY (BTS 固有)
- ACQPROCESS (BTS 固有)
- ACTIVITY (BTS 固有)
- CHANNEL (チャンネル固有)
- DATATYPE (チャンネル固有)
- FROMCCSID (チャンネル固有)
- PROCESS (BTS 固有)

コンテナ・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 (チャンネル、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. チャンネル: プログラムが現行チャン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ンネルを持たず、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・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。

チャン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、チャンネルを受け渡して、動的にルーティングできます。

したがって、以下のタイプのチャンネル関連の要求を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ンネル名の 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、前述の 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ンネルがない場合、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ン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。

チャンネルを使用するアプリケーションは、チャンネル内で 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 ではなくチャン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。

データ変換

チャンネルおよびコンテナーを使用するアプリケーション・プログラムは、データをコード・ページ間で頻繁に変換する必要があります。

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 がサポートされている変換を参照してください。

チャンネルを使用したコード・ページ変換の準備

CICS は、z/OS 変換サービスを使用して文字変換をサポートします。

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は、z/OS による Unicode データ変換を参照してください。

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。

これらのサービスによってサポートされている変換については、「z/OS Unicode Services ユーザーズ・ガイドおよび解説書」で該当の付録を参照してください。これらの変換には、EBCDIC、ASCII、Unicode などの広い範囲の文字エンコード方式間での変換機能が含まれます。

注:

1. IBM MQ トランSPORTなどで使用される 037 と 500 の間の変換は、CICS と IBM MQ で使用される文字エンコード方式のわずかな違いのために発生する EBCDIC から EBCDIC への変換です。
2. 各コード・ページ内のすべてのポイントが他のコード・ページ内に直接対応するものがあるわけではありません (例えば、EBCDIC 文字 NL など)。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。特定の状況でどのような変換がより適切かということについては、プログラミング・コミュニティでも見解が分かれています。

CICS は、現在、z/OS 変換サービスを使用することにより、これらの文字変換をサポートしています。以前のリリースの CICS では一連のテーブルを使用して変換が実行されていたため、それらのテーブルを使用する変換も引き続きサポートされています。それらのテーブルでサポートされていない一対の CCSID 間での変換を CICS で実行する必要がある場合には、z/OS 変換サービスが使用されます。

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ます、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 では、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されます。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。

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、コンテナへのデータの書き込み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・アプリケーションで使用されるデータ変換モデルはシンプルです。チャネル・コンテナ内のデータは、アプリケーション・プログラマーの管理下で 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、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 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。 コンテナ内のデータは、(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・ページを識別するための 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。

データ変換には以下の API コマンドを使用することができます。

- EXEC CICS PUT CONTAINER [CHANNEL] [DATATYPE] [FROMCCSID
| FROMCODEPAGE]
- EXEC CICS GET CONTAINER [CHANNEL] [INTOCCSID |
INTOCODEPAGE]

非 Language Environment (LE) AMODE(64) アセンブラ言語アプリケーション・プログラムの場合、以下の API コマンドも 64 ビット・ストレージ内のデータの変換に使用することができます。

- EXEC CICS PUT64 CONTAINER [CHANNEL] [DATATYPE]
[FROMCCSID | FROMCODEPAGE]
- EXEC CICS GET64 CONTAINER [CHANNEL] [INTOCCSID |
INTOCODEPAGE]

CICS が自動的に文字データを変換する方法:

PUT CONTAINER コマンドまたは PUT64 CONTAINER コマンドの DATATYPE(DFHVALUE(CHAR)) オプションを使用して、コンテナが変換に適切な文字データを保持するように指定することができます。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、GET CONTAINER コマンドまたは GET64 CONTAINER コマンドはデータを自動的に変換します。

このタスクについて

以下の手順に、GET CONTAINER コマンドおよび PUT CONTAINER コマンドを使用して自動的に文字データが変換されるようにする方法を示します。

AMODE (64) プログラムでは、PUT64 CONTAINER コマンドおよび GET64 CONTAINER コマンドを使用して、同様の方法で 64 ビット・ストレージ内のデータを変換することができます。これらコマンドは、非 Language Environment (LE) AMODE(64) アセンブラー言語アプリケーション・プログラムのみで使用するものです。また、CICS ビジネス・トランザクション・サービス (BTS) コンテナはサポートされていません。

手順

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。以下に例を示します。

```
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 に返されます。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でプラットフォームが異なる場合は、データ変換が自動的に行われます。

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でプラットフォームが異なる場合は、データ変換が自動的に行われます。

コンテナを使用したコード・ページ変換:

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に、コンテナ API を使用することができます。

このタスクについて

以下の例は、データを code page1 から code page2 に変換します。

```
EXEC CICS PUT CONTAINER(
  temp
) DATATYPE(DFHVALUE(CHAR))
FROMCCSID(
  code_page1
) FROM(
  input-data
)
EXEC CICS GET CONTAINER(
  temp
) INTOCCSID(
  code_page2
)
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 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るために、変換の終了後は、変換されたデータをコピーし、コンテナを削除する。
- チャンネルのシップを避けるために、一時チャンネルを使用する。一時記憶域およびチャンネルの使用について詳しくは、192 ページの『一時記憶域を使用する場合の類縁性の回避』を参照してください。
- 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・ページとそのチャンネルのコード・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 アプリケーションを使用して、ソケットや IBM MQ メッセージ・キューからの UTF-8 または UTF-16 メッセージの取得、UTF-8 フォーマットのコンテナへのメッセージの配置、同じチャンネルの他のコンテナへの EBCDIC データ構造の配置、またはバック・エンドの 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) 呼び出しを行い、チャンネルを渡すことができます。

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、チャンネルのメッセージ・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、以下を実行できます。

```
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。

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、チャネル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。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に対して「最良事例」を実施する必要があります。詳しくは、144 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 (128 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。

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 (128 ページの『チャネルの利点』を参照してください)。

ユーザー作成の動的または分散ルーティング・プログラムでは、所有するアプリケーションへのチャネルとコンテナのインプリメント計画の有無にかかわらず作業が必要です。ワークロード管理のために CICSplex SM 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 DYRLEVEL、DYRTYPE、および DYRVER 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。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

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、この表に示されている指示を変更します。

このタスクについて

以下の指示では、`structure` は定義されたデータ構造の名前です。EXEC CICS GET CONTAINER および PUT CONTAINER コマンドは、コンテナの識別に

16 文字のフィールドを使用します。ここで `structure-name` として示しているように、使用しているデータ構造と同じ名前をコンテナに付ける規則が役立ちます。

表 17. 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)
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。チャンネルの例では、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

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、この表に示されている指示を変更します。

このタスクについて

以下の指示では、`structure` は定義されたデータ構造の名前です。EXEC CICS GET CONTAINER および PUT CONTAINER コマンドは、コンテナの識別に 16 文字のフィールドを使用します。ここで `structure-name` として示しているように、使用しているデータ構造と同じ名前をコンテナに付ける規則が役立ちます。

表 18. 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

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、この表に示されている指示を変更します。

このタスクについて

以下の指示では、**structure** は定義されたデータ構造の名前です。EXEC CICS GET CONTAINER および PUT CONTAINER コマンドは、コンテナの識別に 16 文字のフィールドを使用します。ここで **structure-name** として示しているように、使用しているデータ構造と同じ名前をコンテナに付ける規則が役立ちます。

表 19. 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)
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

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、この表に示されている指示を変更します。

このタスクについて

以下の指示では、**structure** は定義されたデータ構造の名前です。EXEC CICS GET CONTAINER および PUT CONTAINER コマンドは、コンテナの識別に 16 文字のフィールドを使用します。ここで **structure-name** として示しているように、使用しているデータ構造と同じ名前をコンテナに付ける規則が役立ちます。

表 20. 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を注意してください。

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

チャンネルを利用できなかった古いリリースの CICS では、一部のアプリケーションで一時記憶域キュー (TS キュー) を使用して 32 KB 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 TS キューから複数の読み書きが行われます。

このタスクについて

チャン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、以下の点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・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 またはチャン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。COMMAREA の代わりにチャンネルを使用するように、これらのコマンドをマイグレーションすることができます。

LINK または START コマンドがチャンネルではなく 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 ではなくチャンネルを使用する場合、ルーティング・プログラムに、DFHDYPDS の DYRCHANL フィールドでチャン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ンネルの名前であるため、DYRCHANL フィールドを使用して、チャンネルのコンテナ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・プログラムにチャンネルを使用した場合と同じ種類の機能を与えるために、チャンネルを使用するアプリケーションは、チャンネル内で DFHROUTE という名前の特殊なコンテナを作成できます。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 コンテナのアドレスが指定され、内容を確認および変更できます。

COMMAREA ではなくチャン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。

動的ルーティングと分散ルーティングの概要については、CICS 動的ルーティングの紹介を参照してください。動的ルーティング・プログラムまたは分散ルーティング・プログラムの作成については、動的ルーティング・プログラムの書き込みを参照してください。

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は、JCICS を使用した Java の開発および JCICS Javadoc の資料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 ファウンデーション・クラスの使用法を参照してください。

CICS API によるプログラム制御

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は、以下のようにして使用することができます。

- 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できます (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できます (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 の詳細については、245 ページの『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できます (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 の詳細については、 245 ページの『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 に制御権を返すことができます (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 の詳細については、 245 ページの『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できます (LOAD コマンド)。

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について詳しくは、CICS Interdependency Analyzer for z/OS の概要を参照してください。また、トランザクションの類縁性について詳しくは、175 ページの『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できます (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 で稼働しているアプリケーション・プログラムは、さまざまな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。図 39 に、論理レベルの概念を示します。

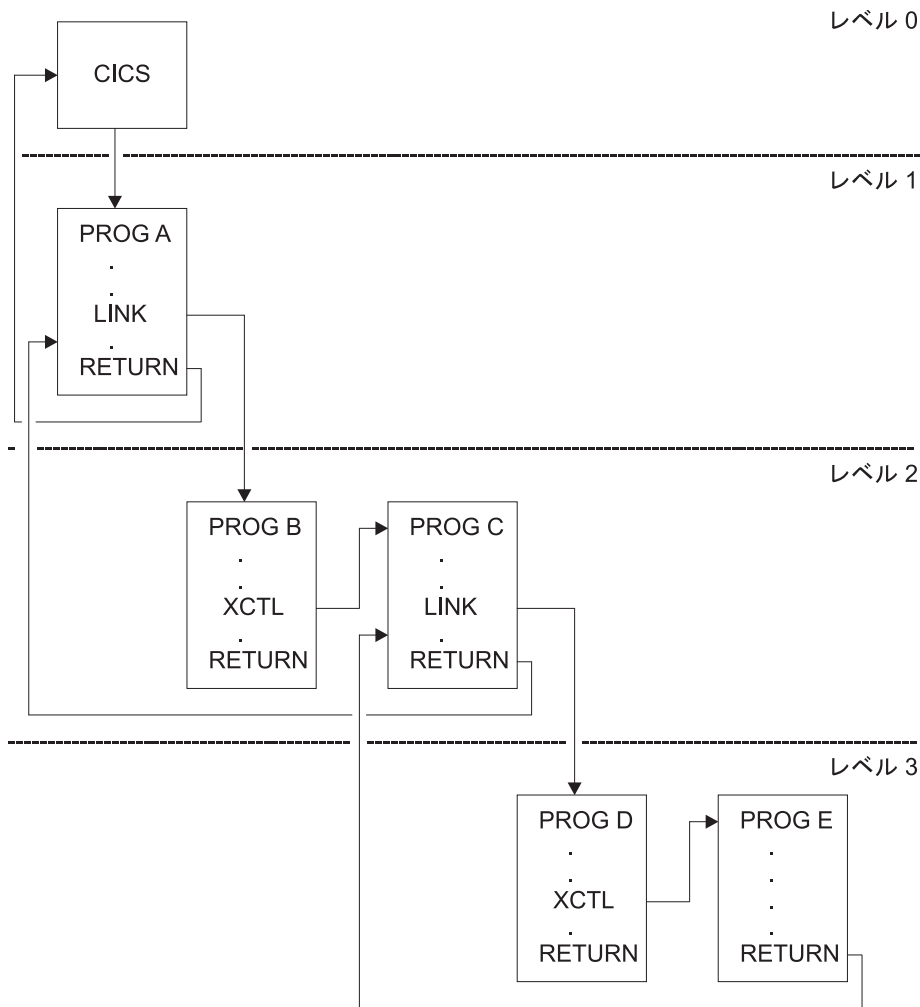


図 39. アプリケーション・プログラムの論理レベル

他のプログラムへのデータの受け渡し

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、それらのコマンドの COMMAREA、CHANNEL、または INPUTMSG オプションを指定することにより、他のプログラムにデータを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

COMMAREA オプションは、プログラムまたはトランザクションへのデータの受け渡しが行われる、データ域 (連絡域) の名前を指定します。これは、LINK、XCTL、および RETURN コマンドのオプションです。

LINK コマンドおよび XCTL コマンドの COMMAREA オプションは、呼び出されているプログラムへのデータの受け渡しが行われるデータ域の名前を指定します。

RETURN コマンドの COMMAREA オプションは、TRANSID オプションで識別されるトランザクションへのデータの受け渡しが行われる、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。

受信側 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。

アセンブラ言語プログラムでは、データ域を DSECT マッピングにしなければなりません。このデータ域のアドレッシングに使用されるレジスターは、DFHEISTG DSECT によってマップされる DFHEICAP (連絡域ポインター) からロードする必要があります。COMMAREA は、64 ビット・ストレージに含めることはできません。

受信側のデータ域の長さは元の連絡域と同じでも、短くてもかまいませんが、長くならないようにする必要があります。データの最初の部分のみについてアクセス権限が必要である場合、受信側のデータ域は短くてもかまいません。受信側のデータ域が元の連絡域よりも長い場合、トランザクションで、渡された領域外のデータの読み取りが試行される可能性があります。また、領域外のデータが上書きされ、その結果、CICS が異常終了する可能性もあります。

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、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。EIBCALEN の値がプログラムの DSECT の値と一致するようにし、トランザクションでその領域内のデータがアクセスされるようにしてください。

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 は送信 トランザクションとともに送信され、受信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) ヘ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 コマンドの発行前にテストする必要があります。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。128 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、120 ページの『チャネルによるプログラム間データ転送』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 オプションを使用して呼び出すことによって、リンクされたプログラムが元の端末入力は継続的に使用可能になるようにします。

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。INPUTMSG の図については、168 ページの図 40 を参照してください。

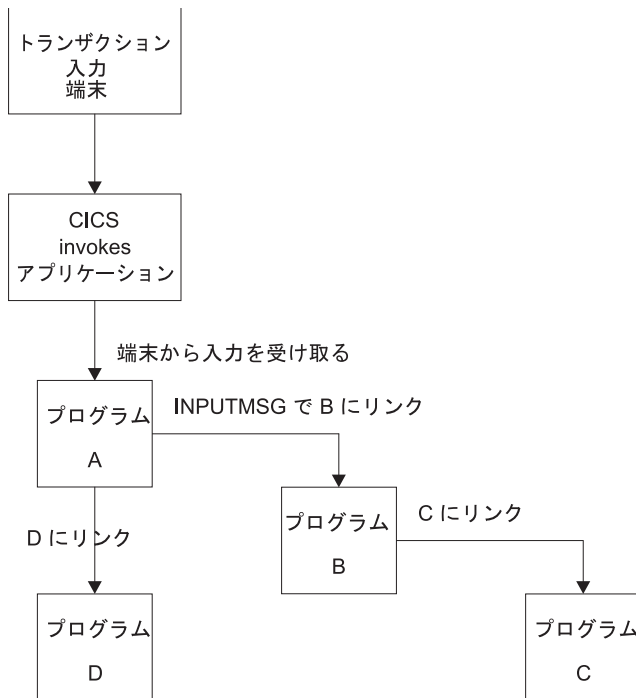


図 40. リンクされたチェーンでの INPUTMSG の使用

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。以下に例を示します。

```
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ます。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入力データに指定変更するので、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。ユーザー置換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、動的ルーティング・プログラムの書き込み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。
- 連絡域内で AMODE(64) プログラムに渡されるアドレスの長さは、64 ビットまたは 31 ビットにすることができます。使用する前に、すべての 31 ビット・アドレスを 64 ビット・アドレスに変更する必要があります。
- AMODE(24) プログラムにデータとして渡されるアドレスは、呼び出し元プログラムが正しくアクセスするためには、16 MB 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。ただし、16 MB より上だが 2 GB より下 (16 MB 境界より上) にある連絡域は、AMODE(24) 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、16 MB 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 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。

COMMAREA は、64 ビット・ストレージに含めることはできません。

LINK を使用したデータの受け渡し

LINK コマンドを使用してデータをリンク先のプログラムに渡す方法を以下の例で示します。XCTL コマンドも同様の方法でコーディングされます。

このタスクについて

図 41から 172 ページの図 44は、COBOL、C、C++、PL/I、およびアセンブラ言語で LINK コマンドをどのように使用するかを示しています。

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ンネルを使用する LINK コマンドの例については、120 ページの『チャンネルによるプログラム間データ転送』を参照してください。

LINK コマンドに指定する LENGTH 値が COMMAREA で渡されるデータの長さに一致していることを確認してください。LENGTH には 0 (ゼロ) を指定しないでください。ゼロを指定すると、結果の振る舞いが予測不能で、EXEC CICS LINK コマンドが失敗する場合があります。

COMMAREA を使用してデータを渡す場合、リンクされているプログラムは、タスクの EIB の EIBCALEN フィールドが、プログラムの予期する内容に一致するか検証しなければなりません。不一致があると、記憶保護違反またはシステム障害になる場合があります。詳しくは、165 ページの『COMMAREA』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 41. LINK コマンド: COBOL の場合

以下の例では、COMMAREA に文字ストリングが含まれています。構造体を含んでいる COMMAREA の例は、172 ページの『RETURN を使用したデータの受け渡し』を参照してください。

```
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 42. LINK コマンド: C の場合

```
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 43. LINK コマンド: PL/I の場合

```

    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 44. LINK コマンド: アセンブラー言語の場合

RETURN を使用したデータの受け渡し

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以下の例で示します。

このタスクについて

図 173 ページの図 45から 174 ページの図 48は、COBOL、C、C++、PL/I、およびアセンブラー言語で RETURN コマンドをどのように使用するかを示しています。

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ンネルを使用する RETURN コマンドの例については、120 ページの『チャンネルによるプログラム間データ転送』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 45. RETURN コマンド: COBOL の場合

```

    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 46. RETURN コマンド: C の場合

```

    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 47. RETURN コマンド: PL/I の場合

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 48. RETURN コマンド: アセンブラー言語の場合

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、その間に類縁性を作成したと言えます。

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 アプリケーションの場合も同様に有効です。Java アプリケーションの開発に関するガイダンスについては、JCICS を使用した Java の開発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、および CICS Business Transaction Services (BTS) のアクティビティーはすべて、動的にルーティングできます。

動的 ルーティング・プログラムを使用すると、以下を動的にルーティングできます。

- 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・プログラムを使用すると、以下を動的にルーティングできます。

- 適格な BTS のプロセスおよびアクティビティー。
- 端末関連でない、適格な EXEC CICS START 要求

動的ルーティングと分散ルーティングの概要について詳しくは、CICS 動的ルーティングの紹介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。(ただし厳密に言えば、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です。) 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。
- 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

トランザクション間の類縁性、およびトランザクションとシステム間の類縁性の 2 つのタイプの類縁性が、動的ルーティングに影響を与えます。トランザクション間の類縁性は、共通リソースを共用している、または処理を調整しているトランザクションのセットの間で発生します。トランザクションとシステム間の類縁性は、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その CICS 領域の特性を問い合わせたり、変更したりします。

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、次のような環境で発生す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。

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

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

イ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

トランザクション間の類縁性に関して、動的または分散ルーティング環境でアプリケーション・プログラムによって使用されるプログラミング手法は、安全な手法、安全ではない手法、または疑いのある手法として分類できます。

- 安全な手法は、トランザクション間の類縁性の原因となりません。
- 安全ではない手法は、本質的にトランザクション間の類縁性の原因となります。
- 疑いのある手法は、実装の仕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。

安全な手法

安全なプログラミング手法は、類縁性の必要を回避します。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 の説明は、182 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 ENQMODEL リソース 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、コンテナの使用。コンテナ・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、179 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

安全ではないプログラミング手法は、類縁性の作成を必要とする場合がある手法です。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、危険をとまないます。

詳しくは、188 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 の説明は、182 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 ENQMODEL リソース 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、184 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域キューの使用
- 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、191 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ため、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、データを受け渡す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。これについては、180 ページの図 49 を参照してください。

COMMAREA の一般的な特性は次のとおりです。

- プロセッサ・オーバーヘッドが低い。
- リカバリ可能ではない。
- 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、24 KB (1 KB = 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もあるためです。)
- 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。

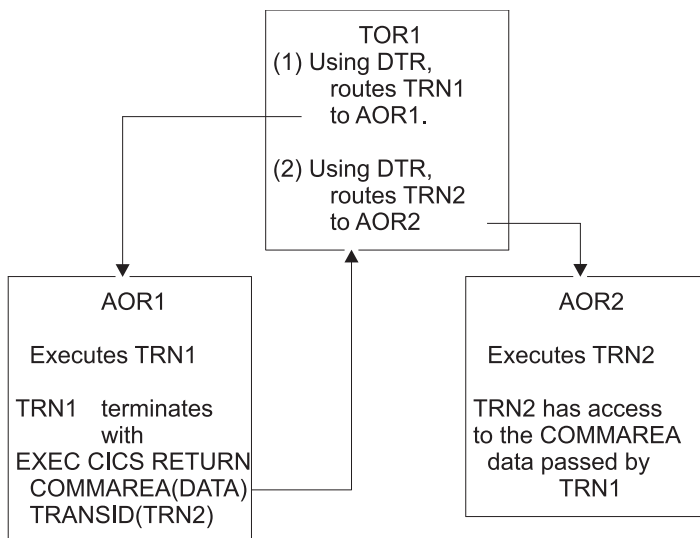


図 49. 動的トランザクション・ルーティング環境における疑似会話型の COMMAREA の使用

疑似会話型トランザクションで使用される COMMAREA は、図 49 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 属性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。これは、182 ページの図 50 に示されています。TCTUA の他の一般的な特性は次のとおりです。

- プロセッサ・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す。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。

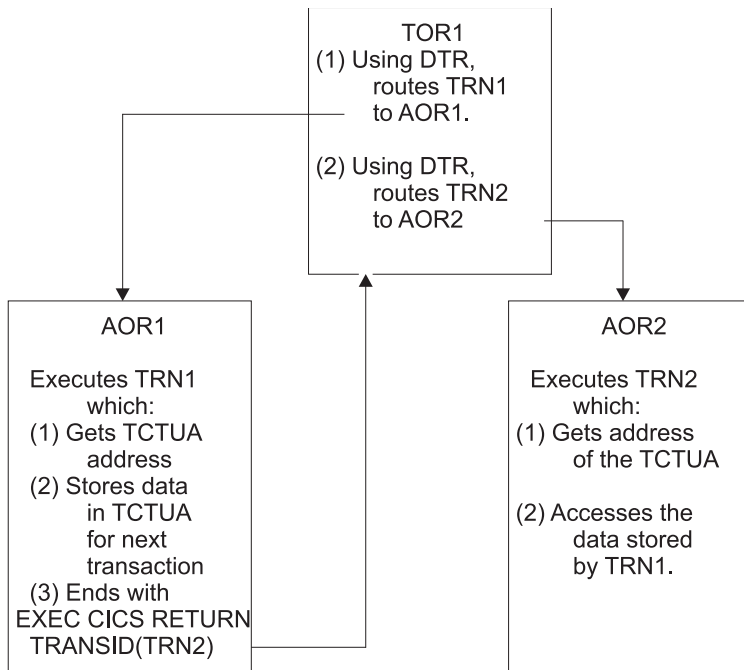


図 50. 動的ルーティング環境における疑似会話型の TCTUA の使用

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 は ENQMODEL リソースを使用する際に z/OS グローバル・リソース逐次化を利用して、複数のアプリケーションが使用するリソースをシスプレックス全体に渡って保護します。

シスプレックス・エンキューは、トランザクション間の類縁性の重要な原因を除去し、シスプレックス全体にわたるタスクのシングル・スレッド化および同期化を可能にします。また、共用一時記憶域キューに対する同時更新の逐次化を可能にし、異なる別の 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ません。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。

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、z/OS グローバル・リソースの逐次化によって管理されます。グローバル・リソースの逐次化について詳しくは、z/OS MVS 計画: グローバル・リソース逐次化を参照してください。CICS 用の z/OS グローバル・リソースの逐次化の構成については、グローバル CICS エンキューおよびデキュー: パフォーマンスの向上を参照してください。

EXEC CICS ENQ コマンドと **EXEC CICS DEQ** コマンドがリソースに対して発行されると、CICS は一致するインストール済みの ENQMODEL 定義があるかどうかを検査します。ENQMODEL リソースの ENQSCOPE 属性は、同じエンキュー有効範囲を共用する領域セットを定義します。ENQSCOPE 属性をブランク (デフォルト値) のままにしておくと、CICS は一致する ENQ 要求と DEQ 要求すべてを発行元の CICS 領域のローカル要求として処理します。ENQSCOPE が非ブランクの場合には、CICS は ENQ または DEQ をシスプレックス全体にわたるものとして扱い、z/OS グローバル・リソースの逐次化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。シスプレックス全体にわたるエンキュー機能またはデキュー機能を使用する必要がある CICS 領域すべてにおいて、必要な ENQMODEL リソースを定義し、インストールする必要があります。ENQMODEL リソースについて詳しくは、ENQMODEL リソース を参照してください。

適切な ENQMODEL リソースを定義すると、アプリケーションはシスプレックス・エンキューを使用します。アプリケーション・プログラムを変更する必要はありません。リソース名が動的に構成されるために事前に分からないアプリケーションの場合、エンキュー EXEC インターフェース・プログラム出口 XNQREQ と XNQREQC を使用して、リソース名の先頭に、該当する ENQMODEL リソース定義と一致する文字を指定できます。これらのユーザー出口について詳しくは、エンキュー EXEC インターフェース・プログラム出口 XNQREQ および XNQREQCを参照してください。

BTS コンテナ

コンテナは BTS アクティビティによって所有され、これを使用して、BTS アクティビティ間で、または同じアクティビティの異なる起動間で、データを受け渡すことができます。

コンテナはアクティビティ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。アクティビティは GET および PUT コンテナを使用して、コンテナの内容を更新します。CICS は、BTS アクティビティに関連した情報 (コンテナも含む) 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。

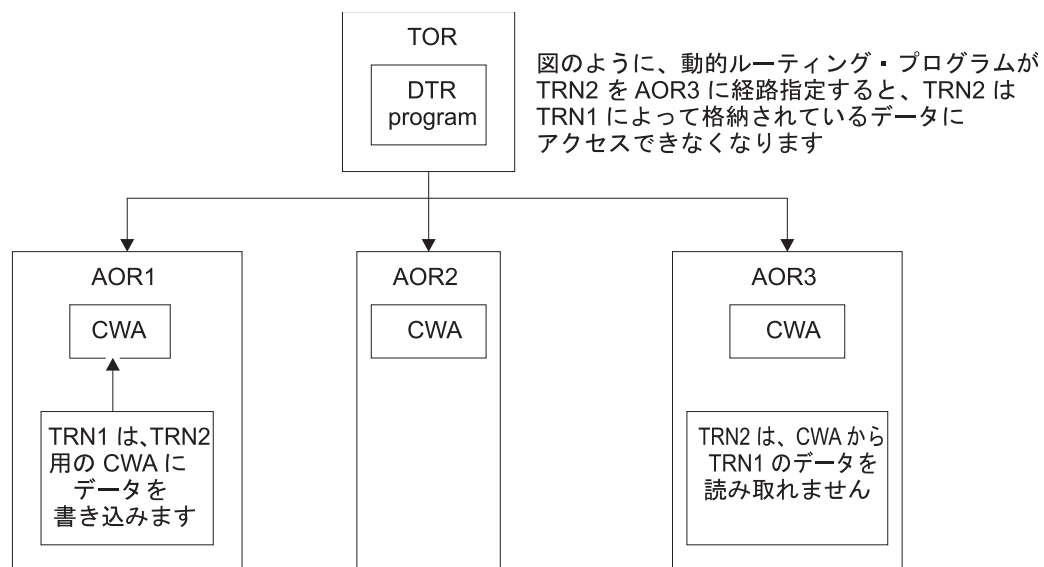
一般に危険なプログラミング手法を次のセクションで説明します。

- 『共通作業域の使用』
- 185 ページの『GETMAIN SHARED ストレージの使用』
- 186 ページの『LOAD PROGRAM HOLD コマンドの使用』
- 188 ページの『タスク存続期間ストレージの共用』
- 189 ページの『WAIT EVENT コマンドの使用』
- 190 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、185 ページの図 51 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。



CWA

図 51. 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。

186 ページの図 52 に、共用ストレージの使用法を示します。

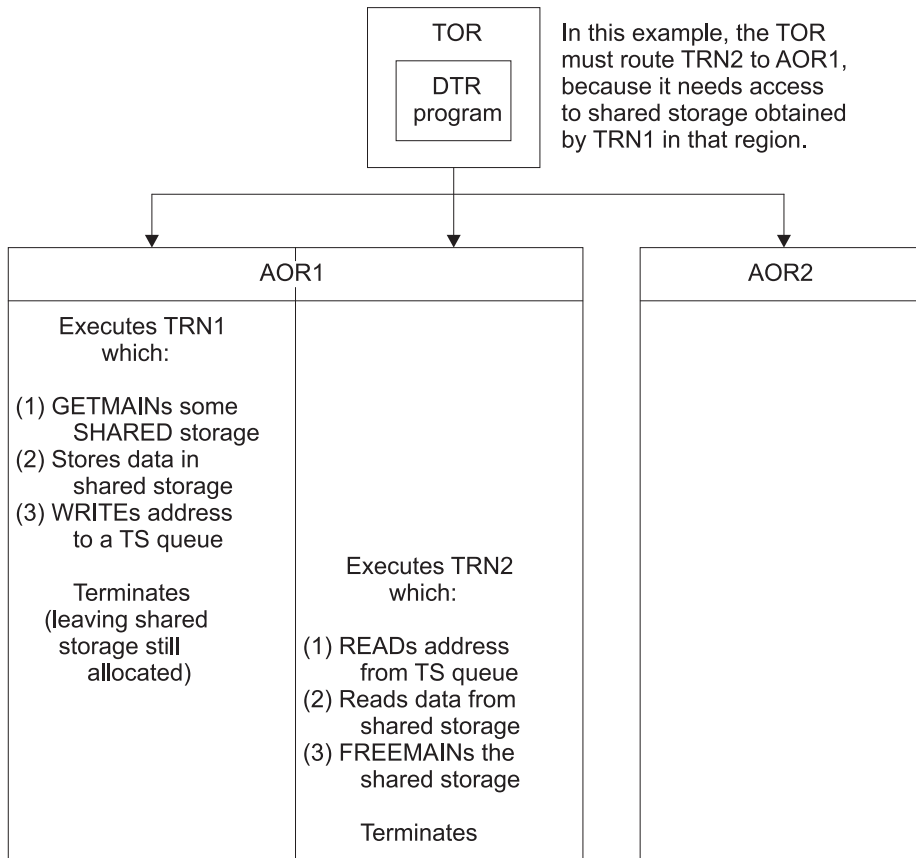


図 52. 共用ストレージの使用により生じるトランザクション間の類縁性を示す図： 動的トランザクション・ルーティング・プログラムは、この類縁性を認識し、TRN2 を TRN1 と同じターゲット領域に確実にルーティングする必要があります。

図 52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る必要があります (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、次の場合に、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。

一時記憶域キューを使用して、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、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 (186 ページの図 52、および図 53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。

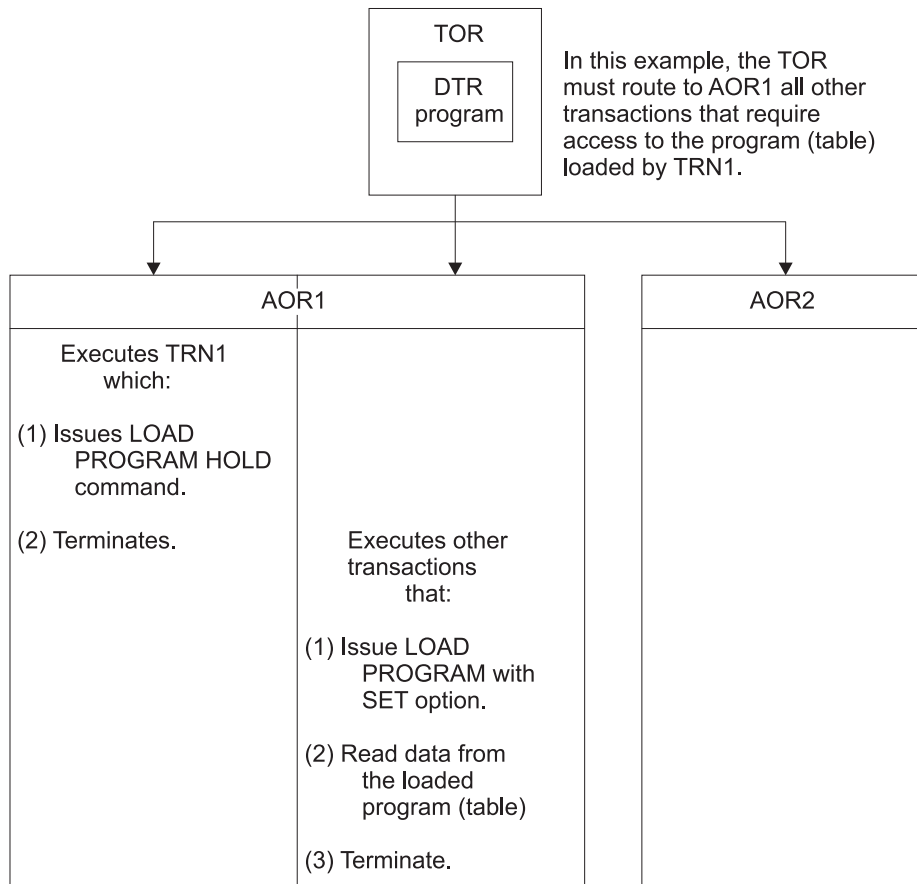


図 53. LOAD PROGRAM HOLD の使用により生じるトランザクション間の類縁性を示す図： 動的ルーティング・プログラムは、この類縁性を認識し、TRN2 を TRN1 と同じターゲット領域に確実にルーティングしなければなりません。

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、別のタスクと共用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、一時記憶域キューを使用して 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 54 の例を参照してください)。

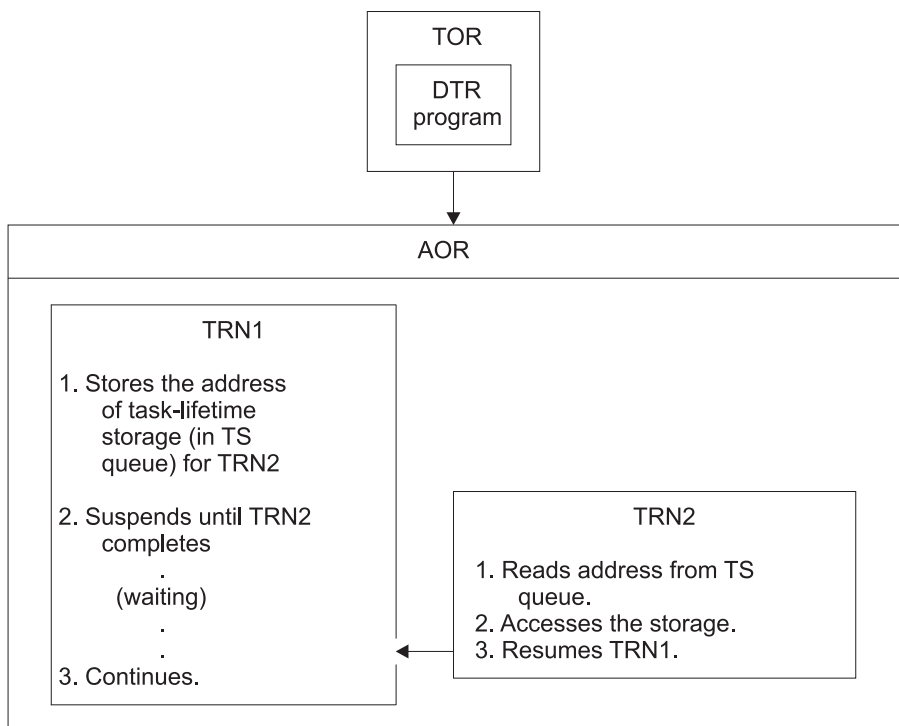


図 54. タスク存続期間ストレージの使用により生じるトランザクション間の類縁性を示す図： TRN2 は、TRN1 と同じターゲット領域で実行されなければなりません。また、TRN2 がタスク存続期間ストレージの使用を完了するまで TRN1 は終了してはなりません。

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 21 を参照してください。

表 21. 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 21. 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、188 ページの表 21 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。一時記憶域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。

190 ページの図 55 はこの同期手法を説明しています。

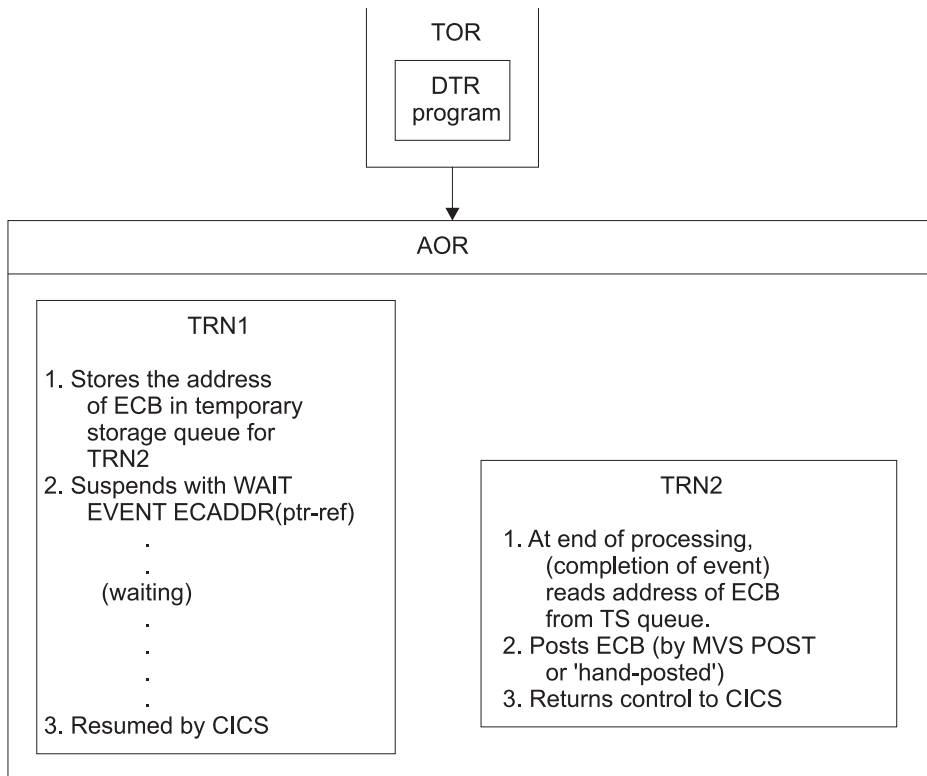


図 55. *WAIT EXTERNAL* コマンドの使用により生じるトランザクション間の類縁性を示す図： TRN2 は、TRN1 と同じターゲット領域で実行されなければなりません。

図 55 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、同じ領域で実行されている CICS タスクに対してのみ動作します。

ENQ および DEQ コマンドは、別の領域に存在するタスクが共用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 の説明は、182 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 ENQMODEL リソース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 56に説明されています。

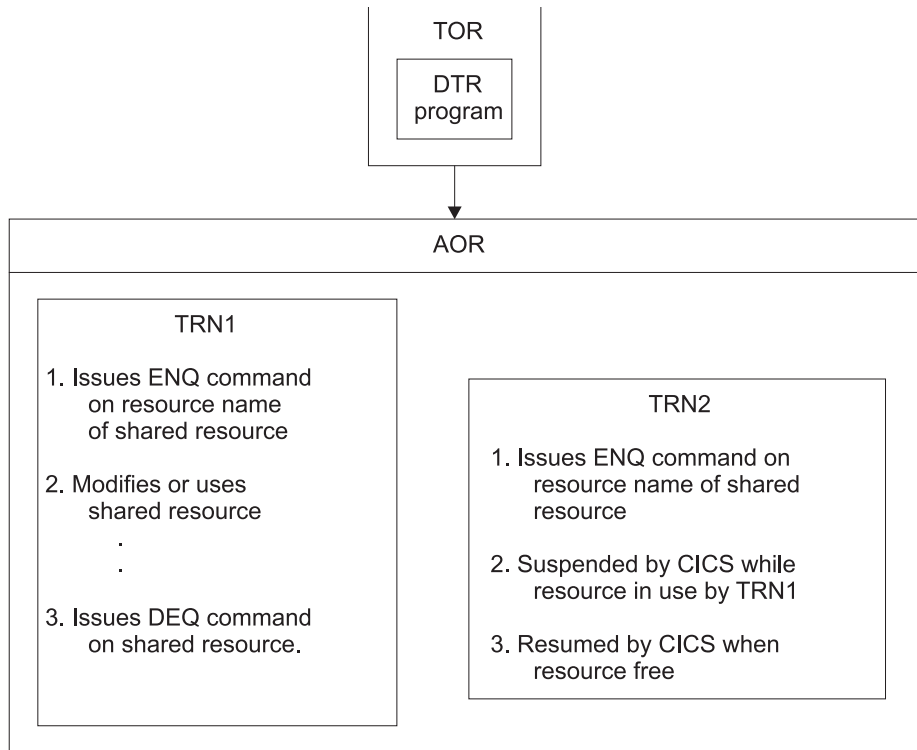


図 56. ENQ/DEQ コマンドの使用により生じるトランザクション間の類縁性を示す図： TRN2 は、TRN1 と同じターゲット領域で実行されなければなりません。

図 56 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。さまざまな方法を採用して、不要な類縁性を回避できます。

- 192 ページの『一時記憶域を使用する場合の類縁性の回避』
- 195 ページの『一時データの使用』
- 196 ページの『RETRIEVE WAIT および START コマンドを使用する場合の類縁性の回避』
- 198 ページの『START および CANCEL REQID コマンドを使用する場合の類縁性の回避』
- 200 ページの『DELAY および CANCEL REQID コマンドを使用する場合の類縁性の回避』

- 202 ページの『POST および CANCEL REQID コマンドを使用する場合の類縁性の回避』

一時記憶域を使用する場合の類縁性の回避

CICS アプリケーション・プログラムは一般的に、一時記憶域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。

一時記憶域キューは、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。一時記憶域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用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 プログラムヘデータを渡す推奨される方法は、チャンネルまたは COMMAREA を介する方法ですが、DPL アプリケーションにおいて、一時記憶域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る必要があります。

時には、一時記憶域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一時記憶域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、一時記憶域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。トランザクション間の類縁性を回避しながら、一時記憶域キューをグローバルにアクセスできるようにするために、キュー所有領域 (QOR) 内のリモート・キューを使用するか、またはカップリング・ファシリティの一時記憶域データ共用プールに存在する共用一時記憶域キューを使用できます。

キュー所有領域内のリモート一時記憶域キュー

一時記憶域要求をキュー所有領域 (QOR) に機能シップするには、以下のようにします。

- アプリケーション所有領域で、REMOTESYSTEM および REMOTEPREFIX (または XREMOTEPFX) 属性を使用して、TSMODEL リソースを作成し、一時記憶域キューをリモートとして定義します。
- キュー所有領域で、対応する TSMODEL リソースを作成して、一時記憶域キューの特性を定義します。

- 重複を回避したい場合は、ローカル領域とリモート領域の両方にインストールできる TSMODEL リソースを作成して、これらのリソースをアプリケーション所有領域とキュー所有領域との間で共用します。

一時記憶域キューの名前が動的に生成される場合、指定された端末に対して固有のキュー名を生成するプロセスに従う必要があります。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。この形式の汎用キュー名は、リモート・キューとして定義できます。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はできず、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る必要があります。

一時記憶域要求に対してグローバル・ユーザー出口を使用し、ローカル・キュー名からリモート・キュー名に一時記憶域キュー名を変更することはできません。

図 57 は、リモート・キュー所有領域の使用を示しています。

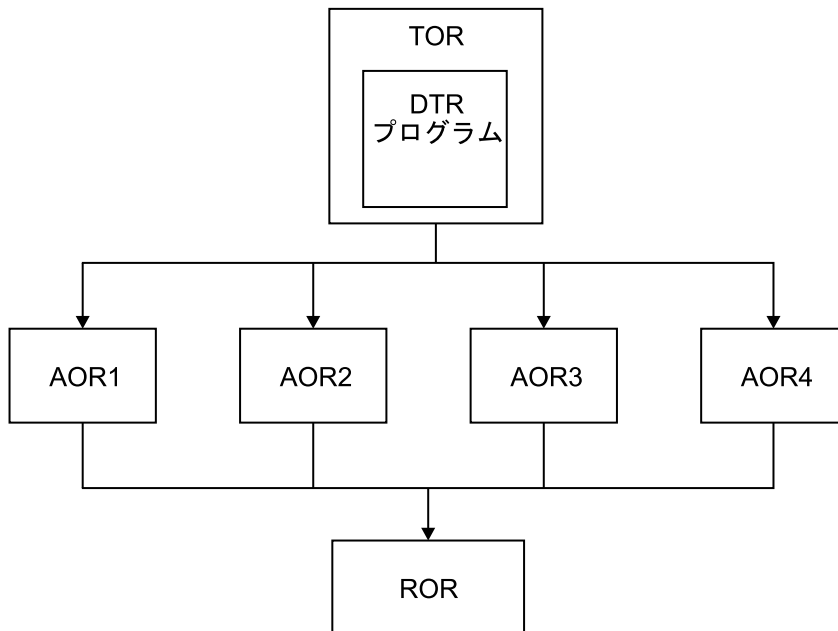


図 57. 一時記憶域に関連したトランザクション間の類縁性を回避するための、リモート・キューの使用：この例は、結合されたファイル所有領域およびキュー所有領域 (リソース所有領域、つまり ROR) を示しています。別々の領域を使用することができますが、ファイル制御および一時記憶域管理により個々に管理されているデータをリカバリーするときに「未確定」の窓が発生する可能性があるので、リカバリー操作時には特別な配慮が必要です。

共用一時記憶域キュー

カップリング・ファシリティの一時記憶域データ共用プール内に存在する共用一時記憶域キューを使用するには、アプリケーション所有領域で、POOLNAME および PREFIX (または XPREFIX) 属性を使用する TSMODEL リソースを作成し、一時記憶域要求を共用一時記憶域キューの名前付きプールにマップします。

一時記憶域サーバーを、カップリング・ファシリティで定義される一時記憶域キューの各プールをサポートするようにセットアップする必要があります。詳しくは、Setting up and running a temporary storage serverを参照してください。

TSMODEL リソース定義は、一時記憶域データ共用プール内に存在するキューに対する明示的な SYSID を指定するアプリケーションをサポートしないことに注意してください。アプリケーションは、代わりに、QUEUE または QNAME オプションを使用する必要があります。アプリケーションが共用キュー・プールに明示的な SYSID を使用するには、一時記憶域テーブル (TST) のサポートが必要です。

図 58は、カップリング・ファシリティの共用一時記憶域キューの同じプールを使用している 4 つの AOR を示しています。一時記憶域サーバーは、常にキュー・サーバー領域プログラムの DFHXQMN によって管理されます。

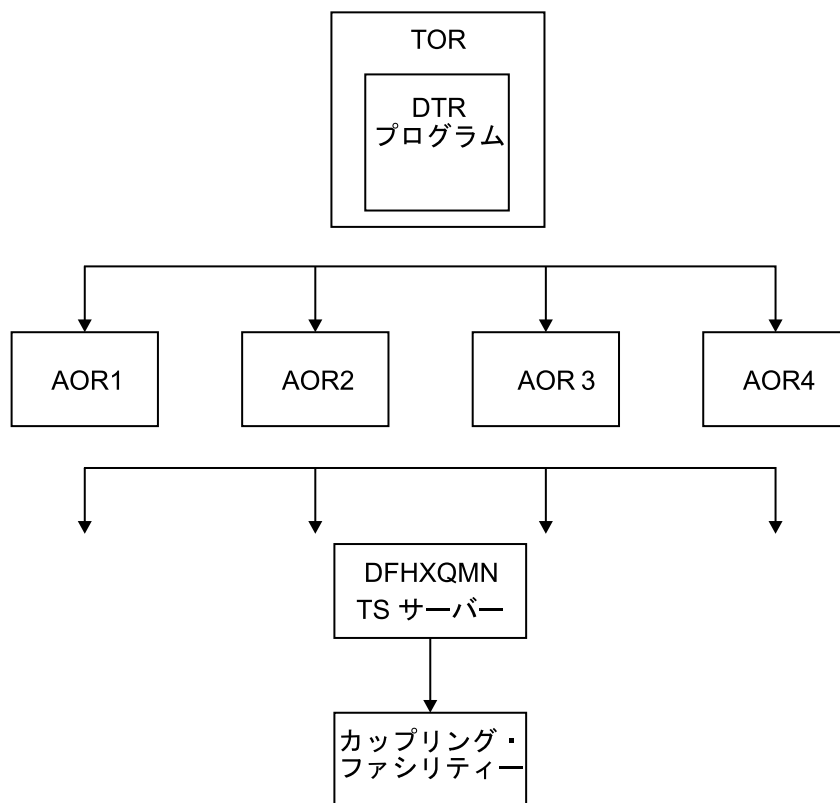


図 58. 一時記憶域データ共用サーバーの使用例

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。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ります。

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・プログラムは、一般的に、一時データ・キュー (TD) を使用します。共用する必要がある 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。

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。

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に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。

196 ページの図 59 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。

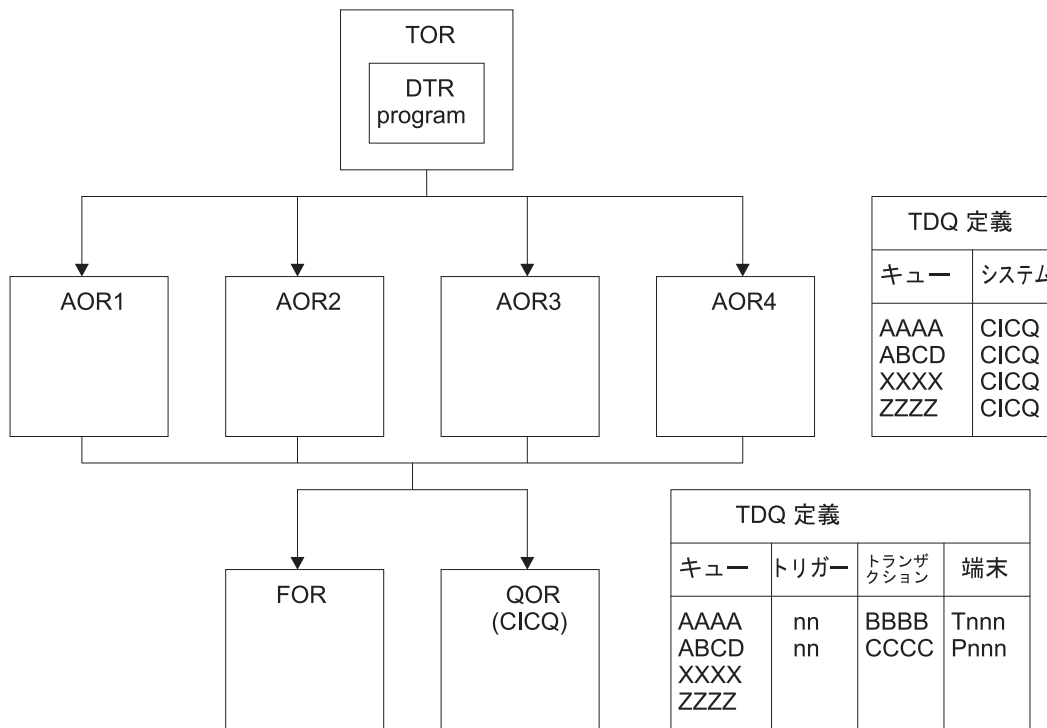


図 59. 一時データに関連したトランザクション間の類縁性を回避するための、リモート・キューの使用：ターゲット領域にインストールされた一時データ・キュー定義は、QOR (CICQ) に所有されたキューとして定義されています。QOR にインストールされた一時データ・キュー定義はすべてローカルで、トリガー・レベルを持つものもいくつかあります。

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もなりません (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 コマンドを使用する場合の類縁性の回避

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。この目的のために、RETRIEVE WAIT および START コマンドを使用します。

197 ページの図 60で説明する例では、TRN1 は非同期タスク TRN2 から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する必要があります。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。1 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。

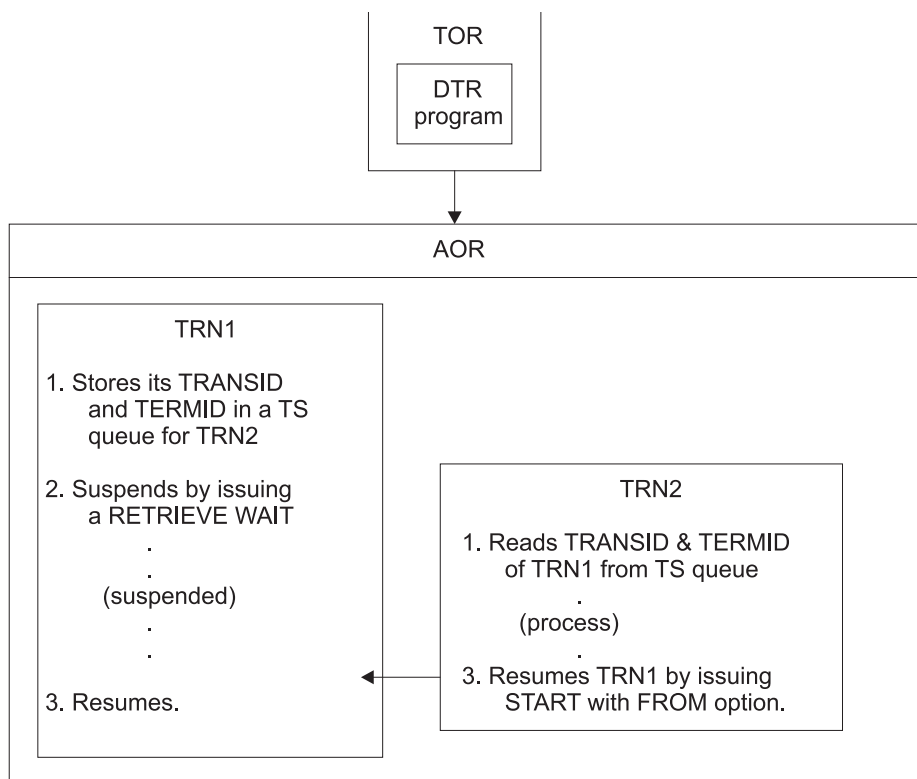


図 60. RETRIEVE WAIT および START コマンドを使用するタスク同期化の図

図 60 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 コマンドを使用する場合の類縁性の回避

CICS START コマンドは、別のトランザクションを開始するためにトランザクションが使用します。別のトランザクションが、CANCEL コマンドを使用して、START コマンドに関連付けられた REQID を指定し、このコマンドを取り消すことができます。

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。

一時記憶域キューの使用は、タスク間で REQID を受け渡す 1 つの方法です。

注: CANCEL コマンドは、指定されたインターバルが満了していない場合にのみ有効です。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。

199 ページの図 61 に、このプログラミング手法を示します。

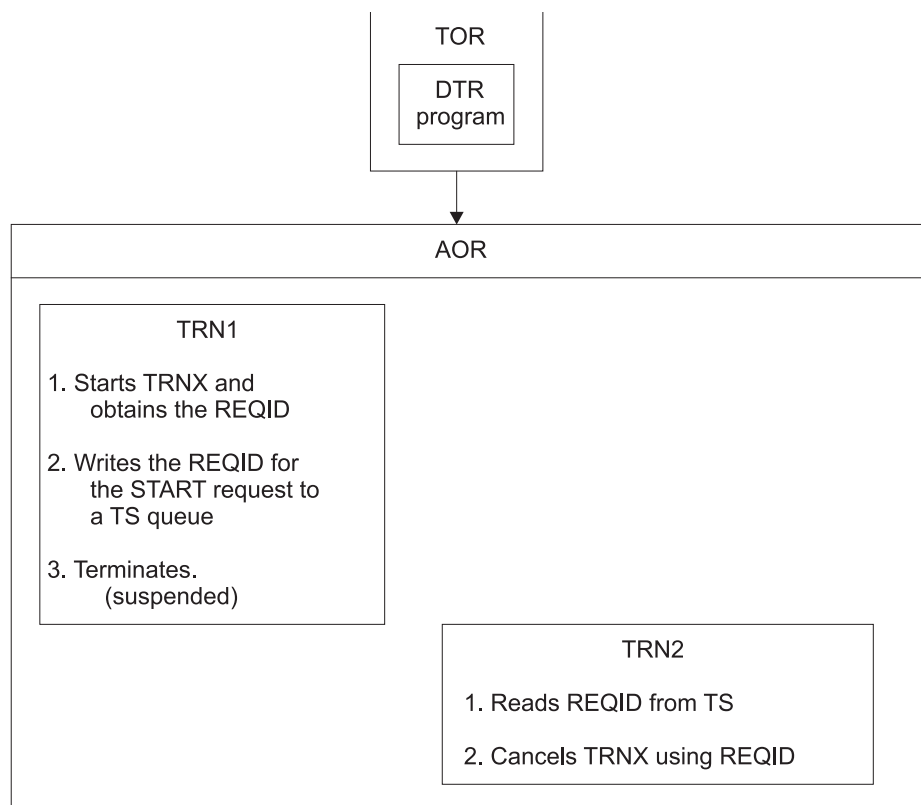


図 61. START および CANCEL REQID コマンドの使用を説明する図

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 コマンドを使用する場合の類縁性の回避

CICS DELAY コマンドは、タスクの中断に使用されます。別のタスクが、CANCEL コマンドを使用して、DELAY コマンドに関連付けられた REQID を指定し、このコマンドを取り消すことができます。

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、あるタスクが DELAY コマンドにより中断された別のタスクを再開できます。

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 コマンドに関連付けられた REQID を指定する必要があります。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。以下に例を示します。

```
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。以下に例を示します。

```
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、201 ページの図 62 で説明しています。

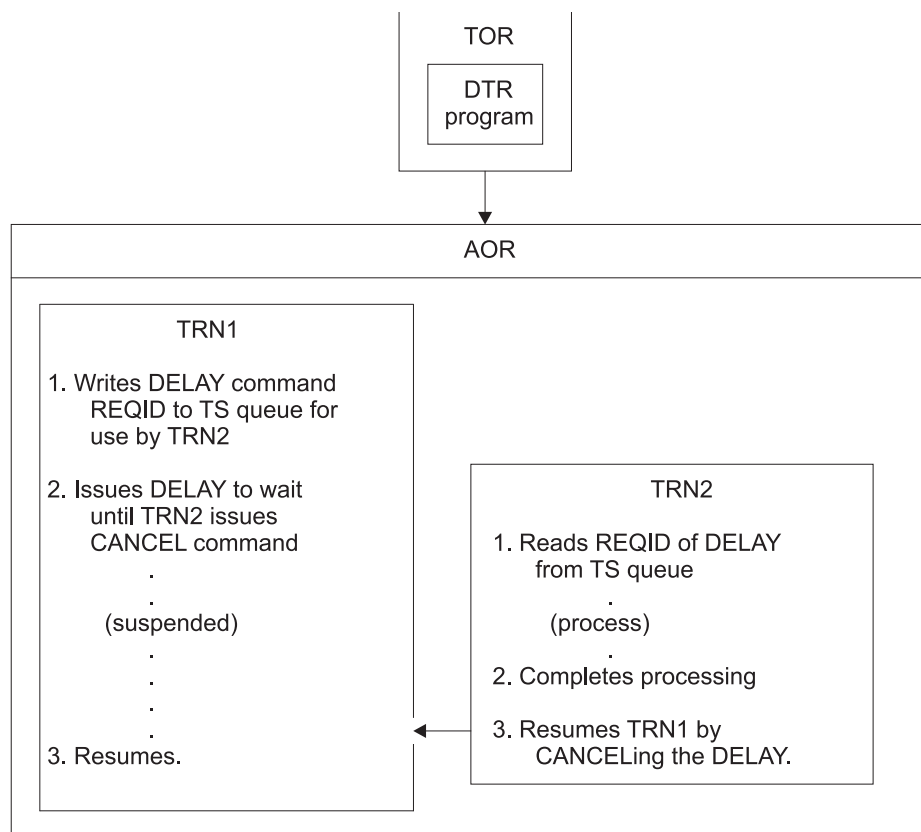


図 62. DELAY および CANCEL REQID コマンドの使用を説明する図

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じ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 コマンドを使用する場合の類縁性の回避

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 コマンドを発行します。

TS ストレージ・キューは、タスク間で POST 要求の REQID を受け渡すために使用できます。

図 63 に、この手法を示します。

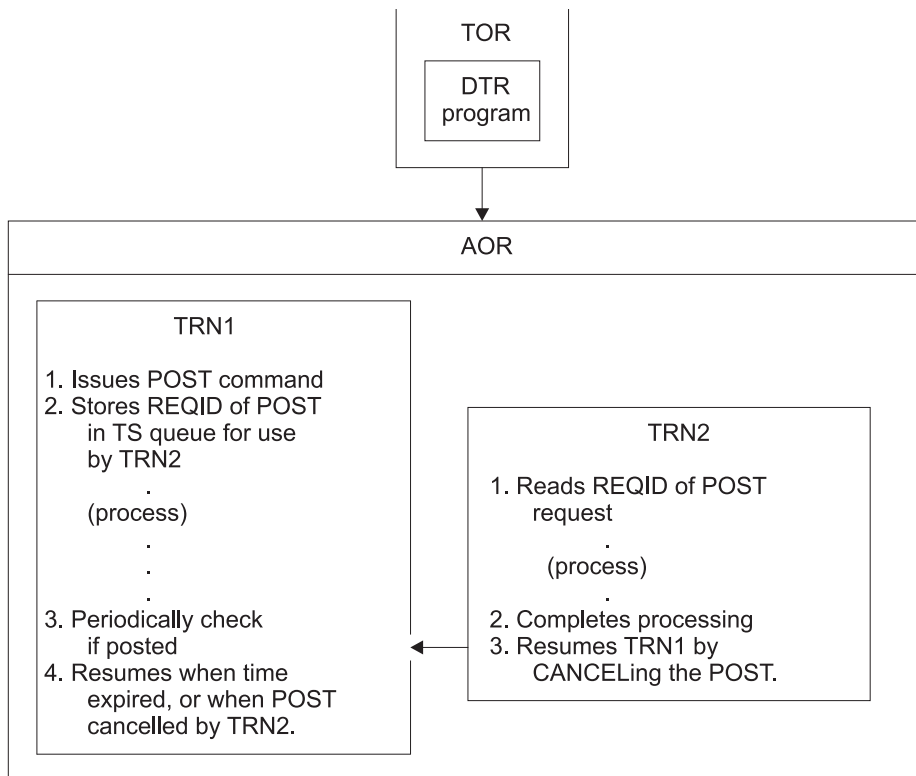


図 63. POST コマンドの使用を説明する図

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じ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期間と有効範囲

動的または分散ルーティング戦略と、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。トランザクション間の類縁性の関係および存続期間は、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します。

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』
- 204 ページの『類縁性の関係および類縁性の存続期間』

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する必要があります。そのための 1 つの方法とし

て、トランザクションをトランザクション・グループに分けます。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。

各類縁性トランザクション・グループ (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を表します。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。

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、類縁性の関係および類縁性の存続期間で分類します。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、およびグループに関連した類縁性の関係と類縁性の存続期間によって構成されます。

類縁性の関係および類縁性の存続期間

トランザクション・グループを作成するとき、**AFFINITY** 属性を使用して類縁性の関係を、**AFFLIFE** 属性を使用して類縁性の存続期間をグループに割り当てることができます。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ます。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は、BAPPL、グローバル、LOCKED、LName、およびユーザー ID の 5 つです。類縁性に割り当てる類縁性の存続期間は、類縁性の関係によって異なります。例えば、類縁性の関係が LOCKED の場合、類縁性の存続期間は UOW である必要があり、類縁性の関係が BAPPL である場合、類縁性の存続期間にアクティビティー型、永続型、処理型、またはシステムを設定できます。

BAPPL 関係:

BAPPL の類縁性の関係をトランザクション・グループに対して定義する場合、同じ BTS プロセスに関連付けられた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る必要があります。

類縁性の関係が BAPPL である場合、類縁性の存続期間は、次の値のいずれかを持っている必要があります。

処理 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。

図 64に、BAPPL 関係がある類縁性グループの例を示します。

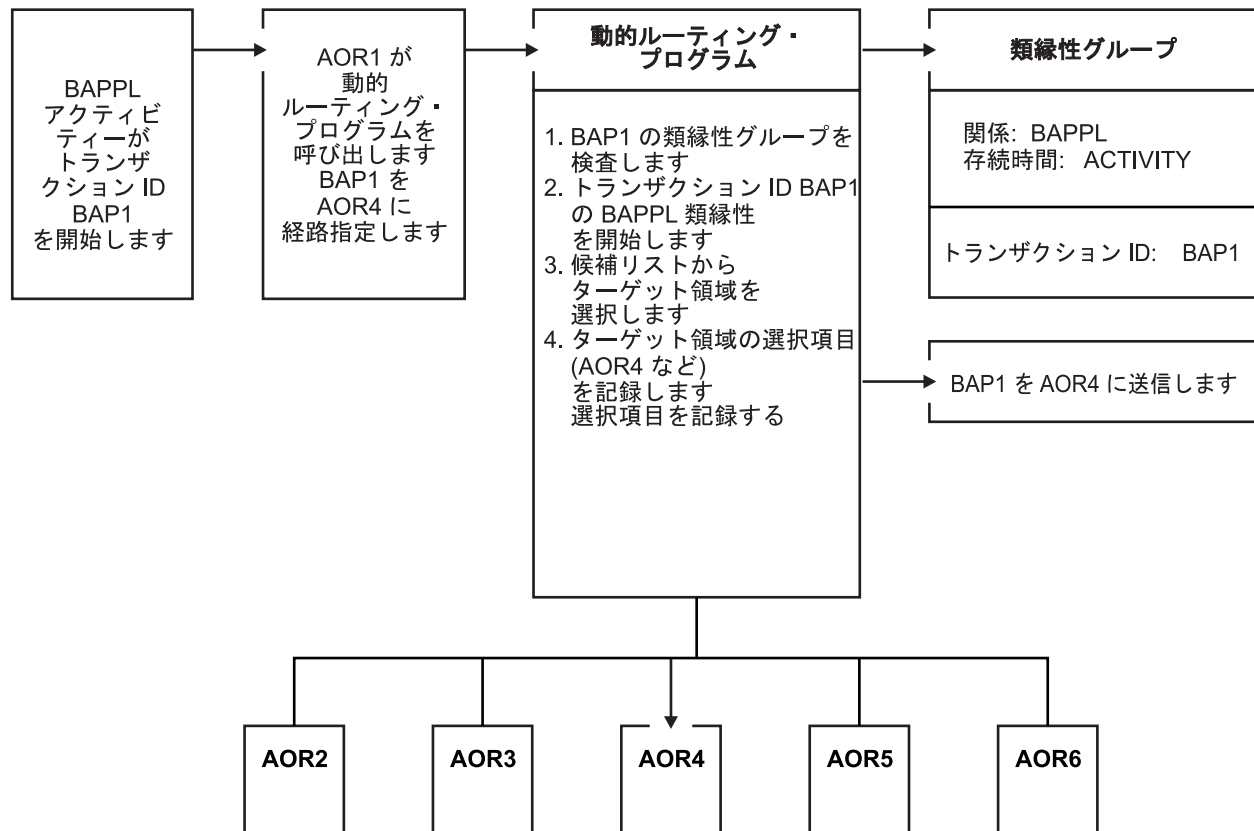


図 64. BAPPL 関係を有し、存続期間がアクティビティーであるトランザクション間の類縁性の管理

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ます。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で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。これらの類縁性によって、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 がアクティブである間、つまり PERMANENT の類縁性の存続期間である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、または予定されているシャットダウンが起こる場合は、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。

グローバル関係:

トランザクション・グループに対してグローバルの類縁性の関係を定義する場合、端末から開始される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、または 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る必要があります。

類縁性の関係がグローバルである場合、類縁性の存続期間は、次の値のいずれかを持っている必要があります。

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域キューを使用して (読み取りまたは書き込み)、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。

207 ページの図 65 に、グローバル関係の例を示します。

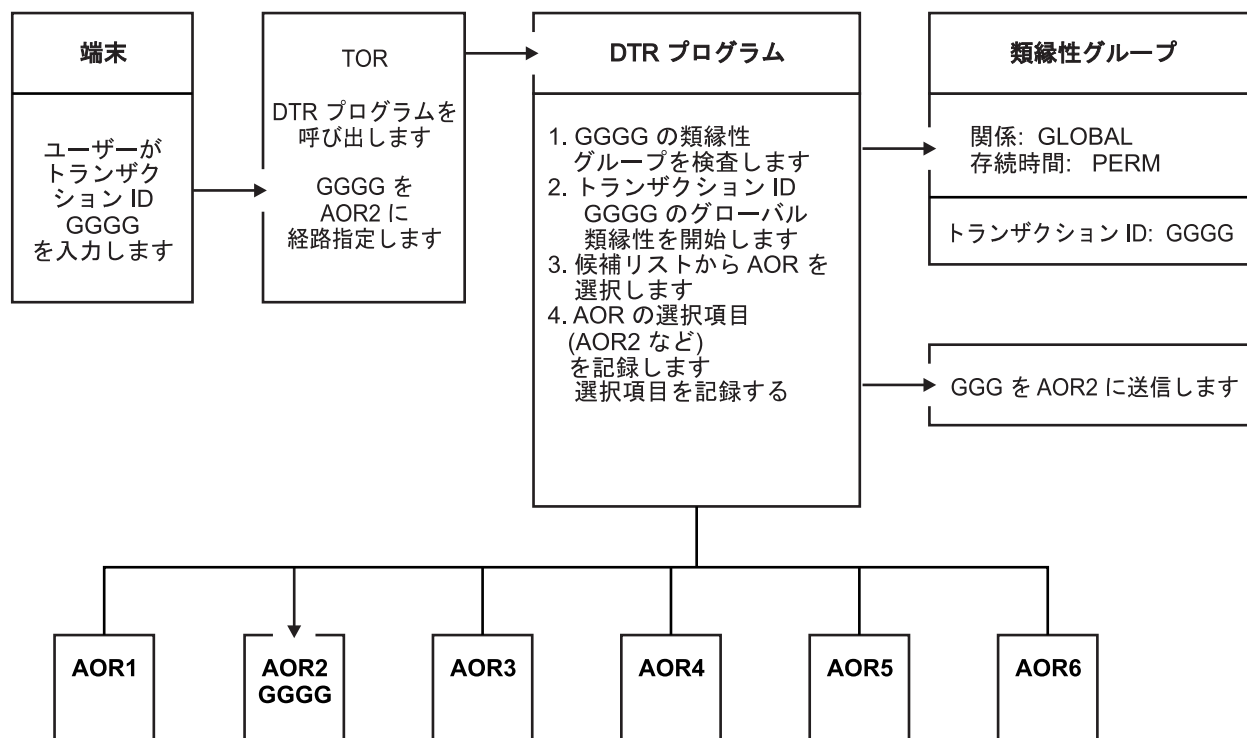


図 65. グローバル関係を有し、存続期間が永続であるトランザクション間の類縁性の管理

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。

LOCKED 関係:

LOCKED の類縁性の関係をトランザクション・グループに対して定義する場合、同じ作業単位を持つ動的にリンクされたプログラムに関連付けられたグループ内のトランザクションのすべてのインスタンスが、作業単位の存続期間中、同一のターゲット領域で実行される必要があります。

LOCKED 関係は、作業単位 (UOW) に関連付けられます。LOCKED 関係の類縁性存続時間は、常に UOW です。作業単位は、CICS SYNCPOINT または ROLLBACK 要求が実行されたとき、または親タスクが終了したときのいずれかに終了します。UOW は、動的プログラム・リンク要求間の LOCKED の類縁性に対してのみ有効です。

LOCKED 関係を持つトランザクションの典型的な例は、同じ動的にルーティングされたプログラムの複数の起動が、共通リソースにアクセスする場合です。プログラムの各起動が、呼び出し先プログラムをターゲット・システム上で実行するトランザクション・コードを持っています。そのトランザクション・コードを、LOCKED 類縁性を指定するトランザクション・グループの一部として指定することによって、そのプログラム名の動的プログラム・リンクのすべてのインスタンスが、呼び

出し元の作業単位が終了するまで、関連付けられたリモート・トランザクションを同じターゲット領域に転送します。これらの後続のプログラム・リンクが異なる領域にルーティングされた場合、デッドロックが発生する可能性があります。このため、作業はリソースをロックした領域から離れた場所にルーティングしないようにする必要があります。詳細は、同じサーバー領域に対する複数のリンクを参照してください。

図 66に、LOCKED 関係がある類縁性の例を示します。

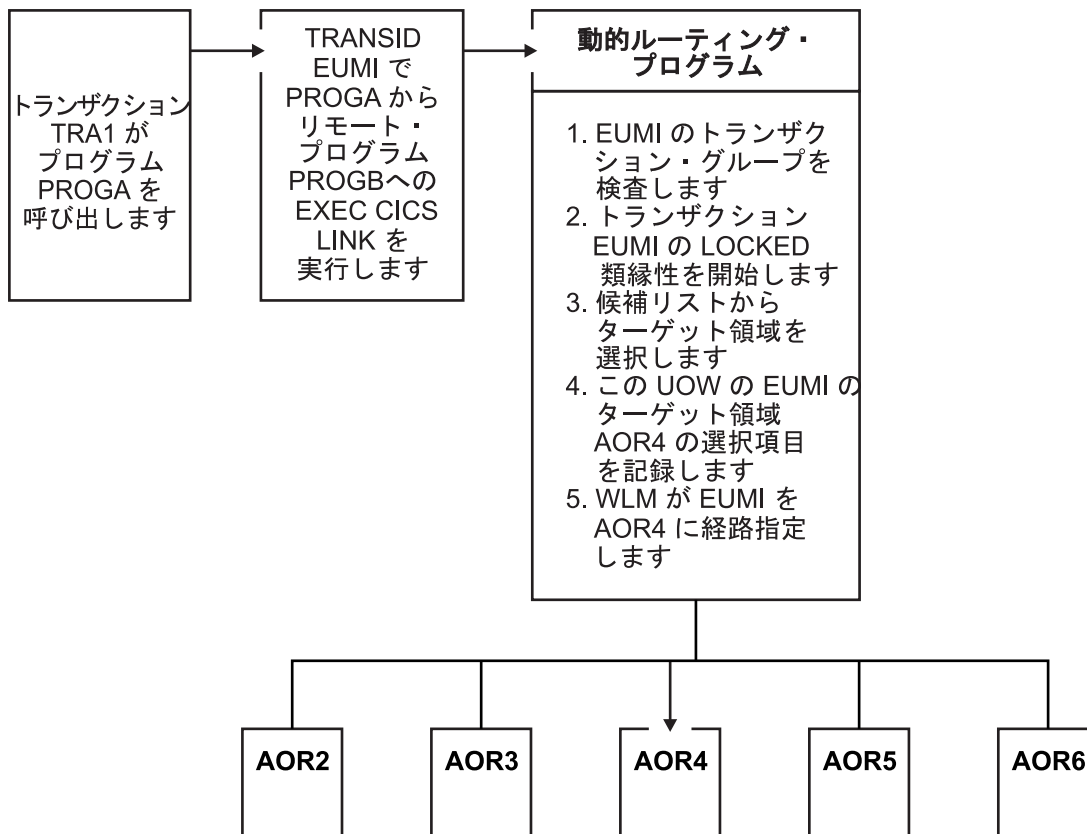


図 66. LOCKED 関係を有し、存続期間が UOW であるトランザクション間の類縁性の管理

この例では、トランザクション TRA1 は、WLMATAFF リソース・テーブルの **NETUOWID** 属性によって識別される作業単位の一部としてプログラム PROGA を呼び出します。PROGA は、リモート・システムのプログラム PROGB に対して **EXEC CICS LINK** 要求を作成します。PROGB は動的として定義され、プログラム定義の **TRANSID** 属性によって定義される関連付けられたトランザクションは、EUMI です。EUMI は、類縁性の関係 LOCKED、および類縁性の存続期間 UOW を使用して定義されるトランザクション・グループに属するユーザー定義のミラー・トランザクションです。ワークロード管理は、このトランザクションを CICS 領域 AOR4 にルーティングします。動的ルーティング・プログラムは、類縁性が既に存在するかどうかを検査した後で、EUMI に対して新しい類縁性を開始します。EUMI のすべての後続のインスタンスは、作業単位の存続期間中はこの類縁性を共用し、作業単位が終了するまで AOR4 にルーティングします。

LU 名 (端末) 関係:

LUnicode の類縁性の関係をトランザクション・グループに対して定義する場合、同じ端末に関連付けられ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る必要があります。

類縁性の関係が LUnicode である場合、類縁性の存続期間は、次の値のいずれかを持っている必要があります。

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から派生した名前を使用している (TS キュー名については 192 ページの『一時記憶域を使用する場合の類縁性の回避』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。

図 67に、このタイプの例を示します。

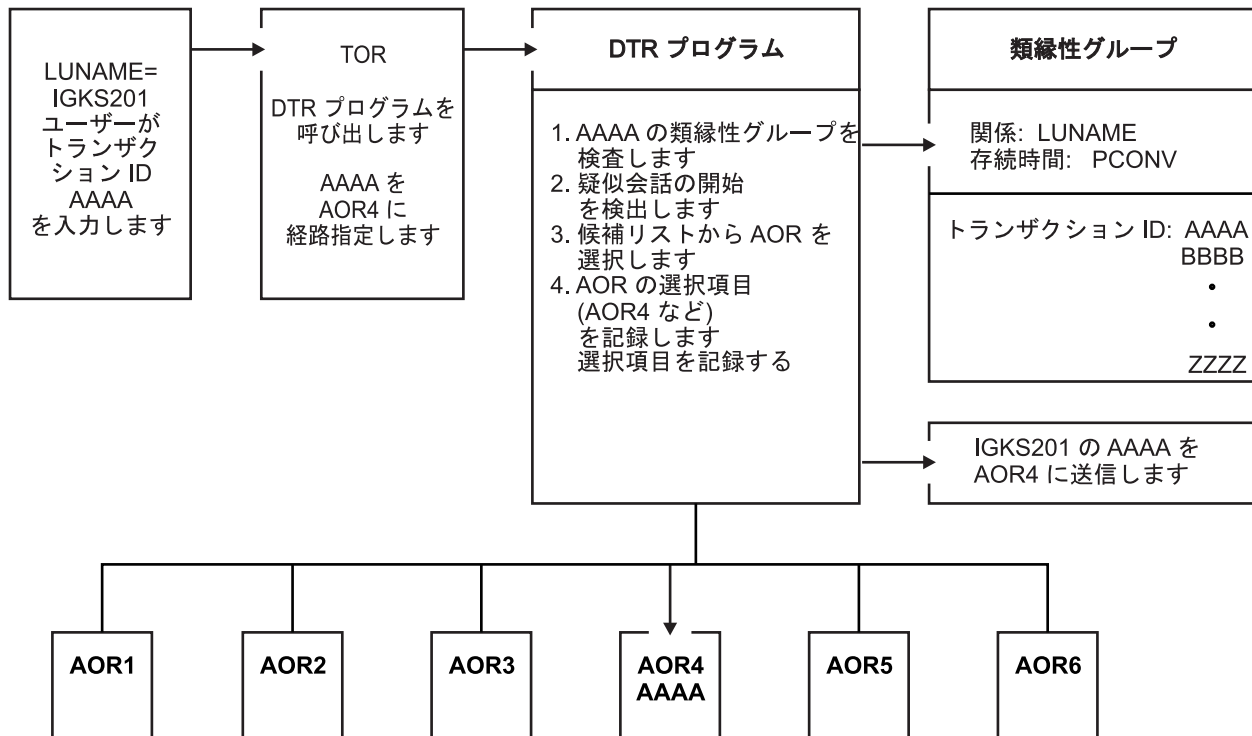


図 67. LU 名関係を有し、存続期間が疑似会話型であるトランザクション間の類縁性の管理

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:

トランザクション・グループに対して userid の類縁性の関係を定義する場合、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 のために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る必要があります。

類縁性の関係が **userid** である場合、類縁性の存続期間は、次の値のいずれかを持っている必要があります。

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 1 人のユーザーだけが許可されている状態でのみ、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 68 に示します。

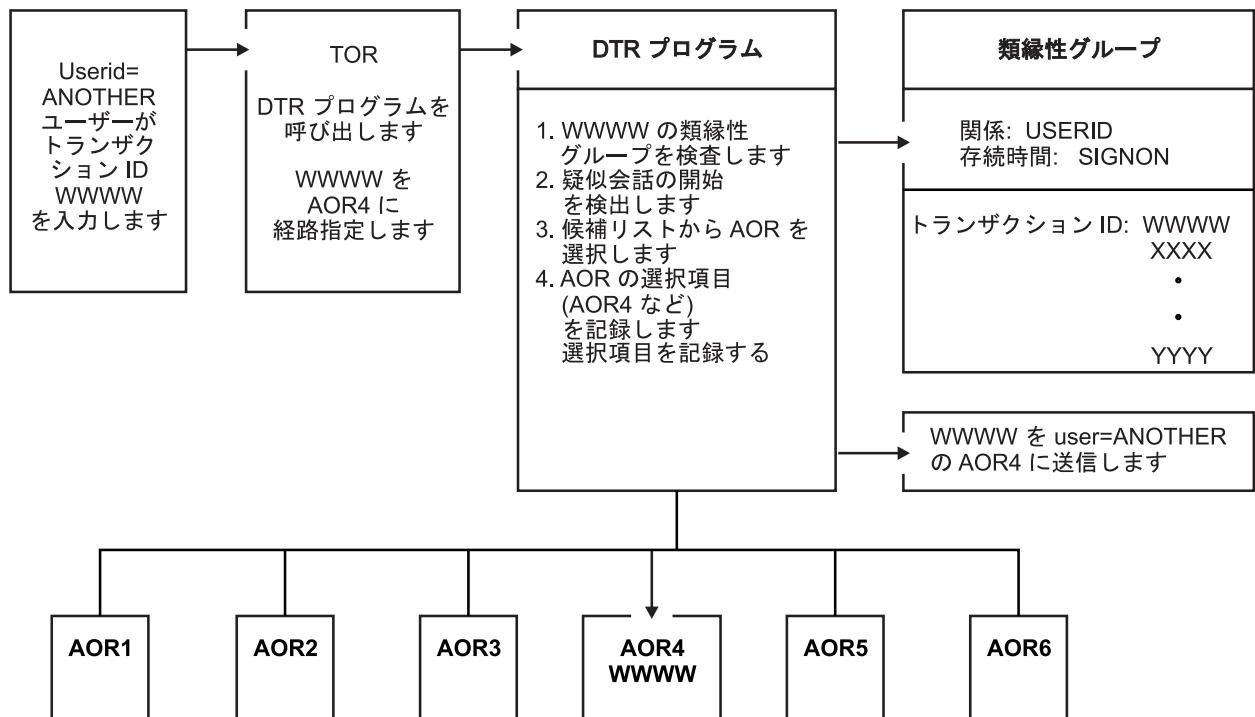


図 68. ユーザー ID 関係を有し、存続期間がサインオンであるトランザクション間の類縁性の管理

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。

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、ジャーナル処理と同期点処理の 2 つの手法を使用できます。

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、429 ページの『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渡されるトランザクションの記録 (多くの場合、ログ と呼ばれます) として機能する場合があります。各ジャーナルは、どのタスクからで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、217 ページの『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・バッファ領域に転送されます。すべてのバッファ・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。

ジャーナル出力の同期

WAIT オプションを指定した WRITE 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。

このコマンドを指定することで、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。このタスクのことを、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・バッファ領域にジャーナル・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がバッファの固定化を要求し、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・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・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ついて前述のいずれかのイベントが発生した場合
- 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 of 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・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 コマンドがいつで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・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)

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。これらの機能については、START=INITIAL パラメーターを使用した 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、Troubleshooting for recovery processing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 사용할 수 있습니다. CICS 시스템에서는, SAA 리소스·리카버리·인터페이스에 의해 EXEC CICS API と同じ機能が提供されます。

制約事項: フル機能 SAA 리소스·리카버리에는, CICS 인프리멘테이션에서는サポートされていないいくつか의戻りコードが用意されています (Systems Application Architecture Common Programming Interface Resource Recovery Reference의 CICS の付録を参照)。

SAA 리소스·리카버리·인터페이스는, 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イ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 리소스·리카버리·인터페이스について詳しくは, Systems Application Architecture Common Programming Interface 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, 251 ページの『分散プログラム・リンクの例』および CICS コマンド・サマリーの『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。

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。

この条件が CICS EXEC インターフェース・プログラムでアプリケーションに戻されるようにすることができます。この条件は、コマンドで RESP オプションを使用して取得することができるので、RESP 値とも呼ばれます。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。条件は、通常は NORMAL です。

何か異常が発生すると、例外条件 (つまり、NORMAL 以外の条件) が発行されます。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の値 (RESP2) を持っており、これが詳細情報を提供します。この RESP2 値を取得するため、コマンドで (RESP オプションに加えて) RESP2 オプションを使用することができます。あるいは、EIB から RESP2 値を読み取ることもできます。

条件の中には、NORMAL でなくてもエラー状態を示すわけではないものがあります。例えば、ファイルの参照中に READNEXT コマンドで ENDFILE 条件が発生した場合、それは、予期された動作を示すものであると思われます。発生する可能性があるすべての状態と、どのコマンドでそれらの状態が発生するかについては、リファレンス: アプリケーション開発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』
- 218 ページの『インライン・コードによる例外条件の処理』
- 221 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

COBOL、PL/I、およびアセンブラ言語アプリケーション (ただし、AMODE(64) アセンブラ言語アプリケーションを除く) の場合、特に指定されていない限り、CICS では、例外条件発生時に常に組み込みの例外処理が使用されます。AMODE(64) アセンブラ言語アプリケーションの場合や、C または C++ で書かれているアプリケーションの場合は、例外条件が発生したときに CICS は何も処理を行いません。アプリケーションで例外条件を処理する必要があります。

例外条件の処理については、218 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 のデフォルトの例外処理を使用する場合、例外条件が発生したときに最もよくとられる処置は異常終了です。それぞれの条件およびコマンドに関する具体的な動作について詳しくは、リファレンス: アプリケーション開発および Introduction to System programming commandsを参照してください。

CICS のデフォルトの例外処理が自身の要件に合致しない場合は、以下の方法で他の処置を指定することができます。

- 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、CICS のデフォルトの例外処理をオフにします。
- コマンドで RESP オプションを指定して、CICS のデフォルトの例外処理をオフにします。このオプションは、NOHANDLE コマンドと同様の方法で CICS のデフォルトの例外処理をオフに切り替えます。また、RESP オプションの引数で指定されている変数を、コマンドによって返される条件の値を使用して更新します。詳しくは、『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 のデフォルトの例外処理をオフにする場合は、コマンド呼び出しで発生する可能性があるすべてのことがプログラムで対処されるようにする必要があります。

HANDLE ABEND コマンド、HANDLE CONDITION コマンド、および IGNORE CONDITION コマンドの組み合わせを使用して CICS のデフォルトの例外処理を変更することも可能ですが、そのような方法は決して推奨されるものではありません。詳しくは、221 ページの『デフォルト CICS 例外処理の変更』を参照してください。

インライン・コードによる例外条件の処理

プログラムが C または C++ で書かれている場合や、AMODE(64) アセンブラー言語アプリケーションである場合は、例外条件を処理するために使用できる手法はインライン・コードのみです。プログラムが C または C++ で書かれておらず、AMODE(64) アセンブラー言語アプリケーションでもない場合は、例外条件を処理するには、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必要があります。そのようにすることにより、CICS でデフォルトの例外処理が実行されないようにします。

このタスクについて

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。

NOHANDLE または RESP オプションを使用する場合、コマンドの実行の過程で発生する可能性のあるすべての条件を、プログラムで確実に処理できることを確認してください。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、EIB フィールド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では、RESP 値をシンボリックに検査できるので、RESP 値のテストが簡単になります。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

COBOL の場合、RESP オプションを使用する EXEC CICS 呼び出しは、以下の例のようになります。PL/I の例と似ていますが、END-EXEC ではなくセミコロン (;) で終わります。

```
EXEC CICS WRITEQ TS FROM(abc)
QUEUE(qname)
NOSUSPEND
RESP(xxx)
END-EXEC.
```

例えば、DFHRESP を使用して RESP 値をテストする場合、コードは次のようになります。

```
IF xxx=DFHRESP(NOSPACE) THEN ...
```

C と C++ における RESP および DFHRESP の使用

C の場合、RESP オプションを使用する EXEC CICS 呼び出しは、以下の例のようになります。この例には、RESP 変数の宣言も含まれています。

```
long response;
...
EXEC CICS WRITEQ TS FROM(abc)
QUEUE(qname)
NOSUSPEND
RESP(response);
```

例えば、DFHRESP を使用して RESP 値をテストする場合、コードは次のようになります。

```
if (response == DFHRESP(NOSPACE))
{
...
}
```

アセンブラー言語における DFHRESP の使用

例えば、アセンブラー言語による RESP 値のテストの場合、コードは次のようになります。

```
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 69. 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多い場合があります。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、これに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 70. 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

アプリケーション・プログラムで **HANDLE CONDITION**、**IGNORE CONDITION**、および **HANDLE ABEND** コマンドを使用すると、CICS で例外と異常終了を処理する方法を変更できます。

このタスクについて

この情報は、COBOL、PL/I、またはアセンブラー言語アプリケーション（ただし、AMODE(64) アセンブラー言語アプリケーションを除く）のみに該当します。サポートされているその他のすべての高水準言語については、これらのコマンドはサポートされていません。

コマンドの概要は、以下のとおりです。

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 で例外条件を処理するときに最もよく使用される方法です。

IGNORE CONDITION、**HANDLE ABEND**、および **HANDLE CONDITION** の各コマンドの現行の効果は、**PUSH HANDLE** コマンドを使用して中断したり、**POP HANDLE** コマンドを使用して復元したりすることができます。

以下の方法で、特定のラベルに制御を渡すことができます。

- **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。
- **HANDLE CONDITION** ERROR(label) コマンドを使用する。

HANDLE CONDITION コマンドは、特定の条件を指定するように CICSコードをセットアップし、その後、それらの条件が発生した場合には、そのコードを使用して、制御をアプリケーションの該当のセクションに渡します。アクティブな **HANDLE CONDITION** コマンドにより、特定の条件に対して指定されたラベルに制御が渡されます。

多数の異なるコマンドで、また、異なる理由で、同じ条件が発生することもあります。例えば、ファイル制御操作やインターバル制御操作などの実行中に **IOERR** 条件が発生することがあります。そのため、特定の条件が発生した原因を調べるには、その前に、その条件を発生させたコマンドを突き止める必要があります。したがって、新しい CICS アプリケーションでは **RESP** オプションを使用することが妥当であると思われます。複数の条件に対するエラー処理を設定するために必要な **HANDLE CONDITION** コマンドが 1 つだけであっても、コード内のどこかで CICS コマンドが失敗した場合に、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特定するのが難しいこともあります。

指定されていない条件が発生した場合、CICS では、デフォルトの処置がとられます。ただし、デフォルトの処置がタスクを異常終了させるものである場合には、**ERROR** 条件が発生します。条件が指定されていても、そのラベルが指定されていない場合には、その条件に対するあらゆる **HANDLE CONDITION** コマンドが非活動状態になり、条件が発生すると、CICS ではデフォルトの処置がとられます。

HANDLE CONDITION コマンドによるエラーの一般的な原因がすべての条件に該当するわけではありません。使い慣れないコマンドを使用する場合は、コマンドの説明を参照して、どのような例外条件を指定できるか確認してください (リファレンス: アプリケーション開発を参照)。すべての例外条件に対して **HANDLE** コマンドを発行しても、エラー処理コードが完了しないことがあります。**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で不完全なデータ変更または誤ったデータ変更がコミットされる結果になる場合があります。

特定の問題を解決するための最善策は、**HANDLE CONDITION** コマンドを使用することですが、これがうまくいかない場合、特定の問題を回避する明白な方法がなければ、システムにデフォルトの処置を実行させるようにする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、225 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。229 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

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。

このタスクについて

HANDLE CONDITION コマンドを使用する場合には、条件の名前を組み込み、関連する条件を発生させる可能性があるコマンドの前に実行されるようにする必要があります。

制約事項: このコマンドは、COBOL、PL/I、およびアセンブラ言語アプリケーション (ただし、AMODE(64) アセンブラ言語アプリケーションを除く) でのみサポートされています。サポートされている他のすべての高水準言語では使用できません。

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、同一のリストで **ERROR** 条件を使用して、他のすべての条件の場合は制御を同一のラベルに渡すように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。コマンドは、プログラムが実行されている間、または以下のいずれかのイベントが発生するまで、活動状態に維持されます。

- 同じ条件について **IGNORE CONDITION** コマンドが検出される。 **HANDLE CONDITION** コマンドは無効になります。
- 同じ条件について別の **HANDLE CONDITION** コマンドが検出される。新しいコマンドが前のコマンドに優先します。

HANDLE CONDITION コマンドは、**NOHANDLE** または **RESP** コマンド・オプションによって一時的に非活動化されます。

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、163 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、**WRITE** コマンドを使用してデータ・セットにレコードを追加するときに発生する可能性のある条件 (**DUPREC** や **LENGERR** など) の処理方法を示しています。この例では、**DUPREC** は特殊ケースとして処理されます。**LENGERR** については標準システム動作がとられます (つまり、タスクが異常終了します)。その他のすべての条件はエラー・ルーチン **ERRHANDL** によって処理されます。

```
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、218 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。詳しくは、218 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、サンプルに記載されている「File A」サンプル・プログラム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、223 ページの『HANDLE CONDITION コマンドの使用』および 226 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、WRITEQ TS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

以下の例では、HANDLE CONDITION ERROR コマンドを使用して予期しないエラーをトラップする方法を示します。

このタスクについて

図 71 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 71. HANDLE CONDITION ERROR コマンドによる予期しない条件のトラップ

HANDLE CONDITION ERROR コマンドは、NOHANDLE または RESP を指定しないコマンドについて何らかの条件が発生した場合に制御をラベル OTHER-ERRORS のパラグラフに渡します。

このコマンドは、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。ただし、ユーザー・プログラムで HANDLE CONDITION コマンドが処理される時点では、その効果はわかりません。その効果がわかるのは、後で、指定されたいずれかの条件を発生させる CICS コマンドがユーザー・プログラムで発行されたときです。

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 オプションを指定するすべてのコマンドは、特定の条件に対する何らかの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) によって作成されます。したがって、明示的に予測される例外を除き、何らかの例外が発生した場合、各プログラムでは、制御が OTHER-ERRORS のパラグラフに渡されます。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小数のコマン

ドによって、NORMAL 以外のどの条件になった場合にも同じ場所に制御が渡されます。

IGNORE CONDITION コマンドの使用

IGNORE CONDITION コマンドを使用して、特定の条件が発生した場合にプログラムを続行するように指定することができます (これは、特定の条件が発生した場合に所定のラベルに制御を渡すように指定する HANDLE CONDITION コマンドとは異なります)。

このタスクについて

制約事項: このコマンドは、COBOL、PL/I、およびアセンブラ言語アプリケーション (ただし、AMODE(64) アセンブラ言語アプリケーションを除く) でのみサポートされています。サポートされている他のすべての高水準言語では使用できません。

あるコマンドに関して潜在的に発生する可能性がある 1 つまたは複数の条件を無視するように IGNORE CONDITION コマンドをセットアップすることができます。IGNORE CONDITION コマンドは、ある条件が発生した場合に何も処置を行わないことを意味します。したがって、制御がそのコマンドの次の命令に戻され、EIB に戻りコードが設定されます。以下の例は、MAPFAIL 条件を無視します。

```
EXEC CICS IGNORE CONDITION MAPFAIL  
END-EXEC.
```

1 つの EXEC CICS コマンドを処理するときに複数の条件が発生することがあります。例えば、ファイル制御コマンドが無効であり、なおかつ、そのコマンドが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ことがあります。CICS は、それらの条件をチェックし、(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・プログラムに渡します。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が、

所定の条件に対する IGNORE CONDITION コマンドは、そのコマンドが組み込まれているプログラムだけに適用されます。コマンドは、そのプログラムが実行されている間、または後で同じ条件が指定されている別の HANDLE CONDITION コマンドが検出されるまで、活動状態に維持されます。HANDLE CONDITION コマンドが検出されると、IGNORE CONDITION コマンドが無効になります。

指定されたスペースより長くなる可能性があるレコードがプログラムで読み取られてもエラーとは見なされずに何ら処置が行われないようにする場合には、IGNORE CONDITION コマンドを使用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。エラーが発生した場合、制御は次のステートメントに渡されるので、プログラムで EIB の戻りコードをチェックする必要があります。エラーと見なさない条件の例については 225 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。さらに条件を指定する場合は、別の IGNORE CONDITION コマンドを追加する必要があります。

HANDLE ABEND コマンドの使用

HANDLE ABEND コマンドは、アプリケーション・プログラム内のプログラム・レベルの異常終了出口を活動化または再活動化します。このコマンドを使用して、事前に活動化された出口を取り消すこともできます。

特定のプログラミング言語では、HANDLE ABEND コマンドを使用して、異常終了を処理するときに実行されるユーザー独自のコードを指定することができます。つまり、異常な状態が発生した場合にアプリケーションで通常の方法で処理して実行を続行できるということです。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 によってそれらのプログラムを呼び出します。

異常終了処理時の制御のフローが、240 ページの図 72 に示されています。

制約事項

- **HANDLE ABEND** コマンドは Java プログラムには適用されません。
- C プログラムおよび C++ プログラムの場合、例外条件によって異常終了が発生することはないため、それらのプログラムではこの方法で **HANDLE ABEND** コマンドを使用することはできません。ただし、**HANDLE ABEND** コマンドは、PROGRAM オプションと共に使用される場合には C および C++ でサポートされます。
- **HANDLE ABEND LABEL** は、DFHEIENT および DFHEIRET を使用しないアセンブラー・プログラムでは使用できません。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 LABEL** は AMODE(64) プログラムでは使用できません。

PUSH HANDLE および POP HANDLE コマンドの使用

PUSH HANDLE コマンドと **POP HANDLE** コマンドは、**HANDLE CONDITION**、**IGNORE CONDITION**、**HANDLE ABEND**、および **HANDLE AID** の各コマンドの効果をそれぞれ中断し、復元します。

PUSH HANDLE

HANDLE CONDITION、**IGNORE CONDITION**、**HANDLE ABEND**、および **HANDLE AID** の各コマンドの現行の効果を中断します。

POP HANDLE

HANDLE CONDITION、**IGNORE CONDITION**、**HANDLE ABEND**、および **HANDLE AID** の各コマンドの効果を、以前 **PUSH HANDLE** が呼び出された時点より前の状態に復元します。

制約事項: これらのコマンドは、COBOL、PL/I、およびアセンブラー言語アプリケーション (ただし、AMODE(64) アセンブラー言語アプリケーションを除く) でのみサポートされています。サポートされている他のすべての高水準言語では使用できません。

また、CICS は、対応する **POP HANDLE** コマンドが見つからない各 **PUSH HANDLE** コマンドに関する条件のテーブルも保持します。

それぞれの条件の発生時には、CICS で以下の一連のテストが使用されます。

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。それ以外の場合、CICS によって条件テーブルがスキャンされます。
2. 条件に対する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、その条件に関するデフォルトの処置が実行中断である場合は、次のような処置がとられます。
 - 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 **NOSUSPEND** オプションまたは **NOQUEUE** オプションが指定されていない場合には、タスクが中断される。
4. 項目が存在せず、その条件に関するデフォルトの処置が異常終了である場合は、CICS が **ERROR** 条件を検索します。
 - a. **ERROR** 条件が検出される場合、この項目によって処置が決まる。
 - 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、規則に対する例外です。

ALLOCATE、**ENQ**、**GETMAIN**、**WRITE JOURNALNAME**、**WRITE JOURNALNUM**、**READQ TD**、および **WRITEQ TS** コマンドはすべて、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対しては、NOSUSPEND オプションを使用すると、この待機を禁止して、即座にアプリケーション・プログラム内の次の命令に戻ることができます。

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 を使用するということは NOHANDLE を暗黙指定するということであるため、RESP を **RECEIVE** コマンドと共に使用するときには注意してください。なぜならば、**HANDLE CONDITION** コマンドだけでなく、**HANDLE AID** コマンドも無効になるからです。つまり、ファンクション・キー応答が無視されるということであり、そのため、それらの応答を事前に ACCT コードでテストしなければならなくなります。508 ページの『HANDLE AID コマンドの使用』を参照してください。

SOAP 障害メッセージの解析

EXEC CICS INVOKE WEBSERVICE コマンドまたは **EXEC CICS INVOKE SERVICE** コマンドを使用してリモート Web サービスを呼び出すと、SOAP 障害メッセージが返されたことを示す RESP2 値が 6 の INVREQ 応答を受け取ることがあります。

問題

アプリケーションから出された SOAP 障害メッセージをより具体的に理解する必要があります。例えば、障害メッセージが CICS とリモート・アプリケーションのどちらから出されたのかを把握したり、障害メッセージに含まれる埋め込みデータにアクセスしたりする必要があることがあります。

応答

XML 構文解析アプリケーション・プログラミング・インターフェース (API) コマンドの **TRANSFORM** を使用できます。(これまでは、CICS から返された DFHWS-BODY コンテナを読み取り、SOAP 障害メッセージの XML 表記にアクセスしてから、選択したメカニズムを使用して XML データを構文解析していました。) 同様の方法を使用して、SOAP V1.2 障害メッセージも処理できます。

DFHSC2LS を使用した SOAP 障害用の COBOL バインディングのビルド

SOAP エンベロープ用の XML スキーマのコピーを <http://schemas.xmlsoap.org/soap/envelope/> からダウンロードします。例えば、スキーマを UNIX ファイル・システム内の /u/example/source/SOAP11.xsd という場所に保管します。DFHSC2LS を使用して、XML スキーマを処理し、スキーマ用の COBOL バインディングのセットを作成し、バンドル・ディレクトリーに XSDBind ファイルとして出力します。次の例のような JCL を使用することもできます。

```
//EXAMPLE EXEC DFHSC2LS,  
//INPUT.SYSUT1 DD *  
MAPPING-LEVEL=3.0  
ELEMENTS=Body,Fault  
SCHEMA=/u/example/source/SOAP11.xsd
```

```

LANG=COBOL
PDSLIB=//EXAMPLE.COBOL.LIBRARY
PDSMEM=SOAP11
XSDBIND=SOAP11.xsdbind
BUNDLE=/u/example/output/bundle/SOAP11
LOGFILE=/u/example/output/logfile.log
*/

```

DFHSC2LS は、次のようないくつかの COBOL 言語構造体を作成します。

- SOAP エンベロープの本体のバインディングは次のようになります。

```

03 Body.
06 Body-num PIC S9(9) COMP-5 SYNC.
06 Body-cont PIC X(16).

```

```

01 SOAP1101-Body.
03 Body-xml-cont PIC X(16).
03 Body-xm1ns-cont PIC X(16).

```

この言語構造体には、SOAP 本体に XML タグがいくつでも表示できるようにするバインディングが含まれています。CICS では、検出されたタグの数が `Body-num` フィールドに、データに関する情報がコンテナの `Body-cont` というフィールドに保管されます。本体のそれぞれの XML タグには 2 つのフィールドが関連付けられています。タグの XML が入るコンテナの名前は `Body-xml-cont` フィールドで、スコープ内 XML 名前空間宣言が入るコンテナの名前は `Body-xm1ns-cont` フィールドで指定されます。

- SOAP エンベロープ本体の障害のバインディングは次のようになります。

```

03 Fault.
06 faultcode-length PIC S9999 COMP-5 SYNC.
06 faultcode PIC X(255).
06 faultstring-length PIC S9999 COMP-5 SYNC.
06 faultstring PIC X(255).
06 faultactor-num PIC S9(9) COMP-5 SYNC.
06 faultactor.
09 faultactor2-length PIC S9999 COMP-5 SYNC.
09 faultactor2 PIC X(255).
06 detail3-num PIC S9(9) COMP-5 SYNC.
06 detail2.
09 Xdetail-num PIC S9(9) COMP-5 SYNC.
09 Xdetail-cont PIC X(16).

```

```

01 SOAP1102-Xdetail.
03 detail-xml-cont PIC X(16).
03 detail-xm1ns-cont PIC X(16).

```

この言語構造体には、単一の SOAP 障害を解析できるようにするバインディングが含まれています。これは `faultcode`、`faultstring`、および `faultactor` フィールドへのアクセスを提供し、SOAP 障害の `detail` セクション内で検出された任意の数の XML タグをマップするための構造体も含まれています。

CICS へのバンドルのインストール

次の例のような `BUNDLE` 定義を作成してインストールします。

```
GROUP: EXAMPLE
```

```
DESCRIPTION: Bundle for mapping SOAP 1.1 SOAP Faults
```

```
BUNDLEDIR: /u/example/output/bundle/SOAP11
```

BUNDLEDIR は、DFHSC2LS の BUNDLE パラメーターを使用して指定した場所を指し示します。DFHSC2LS を CICS が使用するものとは異なる z/OS イメージに対して実行しているなら、バンドル・ディレクトリーをターゲット・マシンにコピーすることが必要になる場合もあります。そのようなときは、別のディレクトリー・パスを使用し、それに合わせて BUNDLEDIR の値を設定します。バンドルには自由に名前を設定できます。SOAP11 でなくてもかまいません。

バンドルを CICS にインストールすると、SOAP11 という名前の BUNDLE リソースと、同じく SOAP11 という名前の XMLTRANSFORM リソースが作成されます。XMLTRANSFORM の名前は、DFHSC2LS の XSDBIND パラメーターの値から取られています。

SOAP 障害メッセージの例

次の SOAP 障害メッセージの例は、**EXEC CICS INVOKE WEBSERVICE** コマンドの実行後に DFHWS-BODY コンテナで検出されたものです。

```
SOAP-ENV:Body>
<SOAP-ENV:Fault xmlns="">
<faultcode>SOAP-ENV:Server</faultcode>
<faultstring>Conversion to SOAP failed</faultstring>
<detail>
<CICSFault xmlns="http://www.ibm.com/software/http/cics/WSFault">
DFHPI1010 *** XML generation failed. A conversion error
INVALID_PACKED_DEC) occurred when converting field 'example' for
WEBSERVICE 'testWebservice'.
</CICSFault>
</detail>
</SOAP-ENV:Fault>
</SOAP-ENV:Body>
```

この例は、変換エラーが起きたときに CICS が作成した障害メッセージです。

TRANSFORM コマンドがこの障害メッセージを処理すると、Body-num が 1 に設定されます。これは、Body タグ内に 1 つの XML タグがあることを示します。Body-cont はコンテナの名前に設定されます (例えば、DFHPICC-00000001)。

コンテナ DFHPICC-00000001 の中に、さらに 2 つのコンテナの名前が置かれます (例えば、DFHPICC-00000002 と DFHPICC-00000003)。

コンテナ DFHPICC-00000002 には、次の例のような Body タグ内の最初のタグが入ります。

```
SOAP-ENV:Fault xmlns="">
<faultcode>SOAP-ENV:Server</faultcode>
<faultstring>Conversion to SOAP failed</faultstring>
<detail>
<CICSFault xmlns="http://www.ibm.com/software/http/cics/WSFault">
DFHPI1010 *** XML generation failed. A conversion error
INVALID_PACKED_DEC) occurred when converting field 'example' for
WEBSERVICE 'testWebservice'.
</CICSFault>
</detail>
</SOAP-ENV:Fault>
```

コンテナ DFHPICC-00000003 には、次の例のようなスコープ内名前空間宣言が入ります。

```
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance "
```

その後 DFHPICC-00000002 コンテナが 2 番目の **EXEC CICS TRANSFORM** コマンドによって解析されると、さらに出力が作成されます。 `faultcode` フィールドと `faultcode-length` フィールドは、それぞれ SOAP-ENV:Server と 15 に設定されます。 `faultstring` フィールドと `faultstring-length` フィールドは、それぞれ「Conversion to SOAP failed」と 25 に設定されます。 `faultactor-num` フィールドは 0 に設定されます。 `detail3-num` フィールドは、オプションの `detail` タグが障害の状態にあることを示す 1 に設定されます。 `detail2-num` フィールドは、オプションの `detail` タグ内に 1 つのサブタグがあることを示す 1 に設定されます。 `detail2-cont` フィールドはコンテナの名前に設定されます (例えば、DFHPICC-00000004)。

コンテナ DFHPICC-00000004 には、さらに 2 つのコンテナの名前が入ります (例えば、DFHPICC-00000005 と DFHPICC-00000006)。

コンテナ DFHPICC-00000005 には、次の例のような SOAP 障害の `detail` セクションで最初に検出される XML タグが入ります。

```
CICSFault
xmlns="http://www.ibm.com/software/http/cics/WSFault ">
DFHPI1010 *** XML generation failed. A conversion error
(INVALID_PACKED_DEC) occurred when converting field 'example'
for WEBSERVICE 'testWebservice'.
</CICSFault>
```

コンテナ DFHPICC-00000006 には、次の例のようなスコープ内名前空間宣言が入ります。

```
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance "
```

アプリケーション・コード

SOAP 障害を解析するアプリケーションを実装するには、以下の手順を行います。

1. DFHWS-BODY コンテナの内容を照会する **TRANSFORM** コマンドを呼び出します。以下に例を示します。

```
EXEC CICS TRANSFORM XMLTODATA CHANNEL(channel-name)
XMLCONTAINER('DFHWS-BODY') NSCONTAINER('DFHWS-XMLNS')
ELEMNAME(element-name) ELEMNAMELEN(element-name-len)
END-EXEC
```

2. `element-name` が Body に設定されている場合は、コンテナを解析します。設定されていない場合は、エラーが発生します。Body を解析するには、次のコマンドを使用します。

```
EXEC CICS TRANSFORM XMLTODATA CHANNEL(channel-name)
XMLTRANSFORM('SOAP11') XMLCONTAINER('DFHWS-BODY')
NSCONTAINER('DFHWS-XMLNS') DATCONTAINER('PARSEDBODY')
END-EXEC
```

```
EXEC CICS GET CONTAINER('PARSEDBODY') SET(body-ptr) END-EXEC
```

3. 解析されたデータにアドレス指定します。以下に例を示します。
SET ADDRESS OF Body TO body-ptr
4. Body-num を調べて、Body 内に少なくとも 1 つのエントリーがあることを確認します。もしあるなら、詳細がリストされているコンテナを読み取ります。以下に例を示します。

```
EXEC CICS GET CONTAINER(Body-cont) SET(body-cont-ptr)
END-EXEC
```

```
SET ADDRESS OF SOAP1101-Body TO body-cont-ptr
```

5. Body 内の最初のタグを照会する **TRANSFORM** コマンドをもう一度呼び出します。

```
EXEC CICS TRANSFORM XMLTODATA CHANNEL(channel-name)
XMLCONTAINER(Body-xml-cont) NSCONTAINER(Body-xmlns-cont)
ELEMNAME(element-name) ELEMNAMELEN(element-name-len)
END-EXEC
```

6. element-name が Fault に設定されている場合は、コンテナを解析します。

```
EXEC CICS TRANSFORM XMLTODATA CHANNEL(channel-name)
XMLTRANSFORM('SOAP11') XMLCONTAINER(Body-xml-cont)
NSCONTAINER(Body-xmlns-cont) DATCONTAINER('PARSEDFault') END-EXEC
```

```
EXEC CICS GET CONTAINER('PARSEDFault') SET(fault-ptr) END-EXEC
```

```
SET ADDRESS OF Fault TO fault-ptr
```

7. これで、障害によって生成されたデータを照会できるようになりました。例えば、faultstring フィールドが役立つことがあります。障害の「detail」セクションからアプリケーション固有の詳細を解析するために、DFHSC2LS を使用し、アプリケーション内でさらに **TRANSFORM** コマンドを実行することによって、アプリケーション固有の COBOL バインディングをさらにビルドすることができます。

注: ステップ 1 と 2 を組み合わせて、単一の **TRANSFORM** コマンドを作成できます (その場合、ステップ 5 と 6 も組み合わせてください)。

ストーム・ドレーン作用の回避

アプリケーションが、リソース・マネージャーへの接続がアクティブでないことを認識しているためにリソース・マネージャーの呼び出しを回避する場合、または接続が使用不可になったためにエラー戻りコードを処理してエラー・メッセージの発行に進むことにより、異常終了せずに正常に戻る場合、ワークロード・マネージャーは誤って CICS 領域により多くの作業をルーティングしてしまうことがあります。この状況をストーム・ドレーン作用 といいます。

アプリケーションが異常終了しなかったため、ワークロード・マネージャーは、リソース・マネージャーが関係する処理がこの CICS 領域では良好な応答時間で達成されていると見なし、より多くの作業を「ストーム・ドレーン」にルーティングします。この状態は、CICS から Db2、IMS、IBM MQ および VSAM RLS への接続を使用するアプリケーションで発生することがあります。

「ストーム・ドレーン作用」を回避するために、リソース・マネージャー・アダプターは z/OS® ワークロード・マネージャーに要求が失敗したことを通知する必要があります。次に、CICSplex® SM ワークロード管理でこの情報が使用されます。Abend probabilities and workload managementを参照してください。

以下のリソースのリソース・マネージャー・アダプターは、z/OS® ワークロード・マネージャーに対して、CICSplex® SM ワークロード管理でその異常終了の確率機能を使用可能にするよう指示し、既にアクティブになっている場合は、影響を受ける CICS 領域にさらに作業がルーティングされないようにします。

Db2

CICS と Db2 の接続の場合は、-923 の SQL 戻りコードが返された時点で、CICS Db2 接続機能により、要求が失敗したことが z/OS ワークロード・マネージャーに通知されます。DB2CONN 定義には、属性 STANDBYMODE=RECONNECT と CONNECTERROR=SQLCODE を指定して構成する必要があります。アプリケーションでは、EXEC CICS EXTRACT EXIT コマンドまたは EXEC CICS INQUIRE EXITPROGRAM コマンドを前もって使用しておいて CICS が Db2 に接続されているかどうかをテストするのではなく、無条件で EXEC SQL 要求を発行して SQLCODE -923 が返されるかどうかをテストする必要があります。-923 SQLCODE が返される場合にのみ、z/OS ワークロード・マネージャーに通知されます。

IMS

CICS と IMS の間の接続の場合、CICS と IMS DBCTL の間のインターフェースがアクティブでないと、**CALLDLI** 要求を発行するアプリケーションは UIB で戻りコード 08FF を受け取ります。この時点で、CICS-DBCTL インターフェースは要求が失敗したことを z/OS ワークロード・マネージャーに通知します。**EXEC DLI** 要求の場合は、**NODHABEND** キーワードを使用していないと、通常、要求はアプリケーション異常終了となります。この状態でも、z/OS ワークロード・マネージャーは通知を受けるので、ストーム・ドレーン作用が回避されます。

IBM MQ

CICS と IBM MQ の間の接続の場合、CICS がキュー・マネージャーに接続されたことが一度もないときは、要求は CICS-MQ アダプターに入る前にリジェクトされるため、ストーム・ドレーン作用を回避することができません。

CICS-MQ アダプターがアクティブである場合は、後に接続障害が発生して以下のいずれかの MQI 理由コードが返されると、要求が失敗したことが z/OS ワークロード・マネージャーに通知されます。

- mqrc_connection_broken
- mqrc_q_mgr_quiescing
- mqrc_connection_quiescing
- mqrc_connection_not_authorized
- mqrc_q_mgr_name_error
- mqrc_q_mgr_not_available
- mqrc_q_mgr_stopping
- mqrc_connection_stopping
- mqrc_adapter_not_available

VSAM RLS

CICS と VSAM RLS の間の接続の場合、CICS が RLS 障害を示す応答を受け取るか、RLS が無効であるか、または以前に RLS 障害が発生したことがあると、それぞれに対して通常の結果として、AFCR 異常終了、AFCS 異常終了、または AFCT 異常終了となります。異常終了が処理されると、CICS ファイル制御は z/OS ワー

クロード・マネージャーに要求が失敗したことを通知します。

異常終了のリカバリー

CICS では、タスクの異常終了中に制御を受け取ることができる独自の出口 (プログラムまたはルーチン) を作成できるようにするため、プログラム・レベルの異常終了出口機能が提供されています。こうした異常終了出口によって実行される機能の例として、開始しても正常終了していないプログラムの終結処理があります。

異常終了の原因として、以下のようなものがあります。

- 次のようなコードによるユーザー要求。

```
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 要求の結果、トランザクション異常終了コード ASCF が返される場合。
- プログラム・チェック。システム・リカバリー・プログラム (DFHSRP) が駆動され、タスクが異常終了して ASRA コードが出力されます。
- オペレーティング・システムの異常終了。DFHSRP プログラムが駆動され、タスクが異常終了してコード ASRB が出力されます。
- タスクのループ。DFHSRP プログラムが駆動され、タスクが異常終了してコード AICA が出力されます。

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、Troubleshooting and supportを参照してください。CICS によって開始される異常終了の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は、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・レベルの異常終了出口を活動化または再活動化します。また、このコマンドを使用して、既に活動状態になっている出口を取り消すこともできます。

出口を活動化にする場合には、以下のいずれかを行う必要があります。

- 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します。
- (COBOL アプリケーション、または AMODE(64) アセンブラー・アプリケーション以外のアセンブラー・アプリケーションの場合のみ) LABEL オプションを使用して、異常終了条件が発生した場合の制御の分岐先となるルーチン・ラベルを指定します。

HANDLE ABEND LABEL コマンドを C、C++、PL/I、または AMODE(64) アプリケーションと共に使用することはできません。PL/I の場合、同等の方法は ON ERROR ブロックを使用することです。

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、162 ページの『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、240 ページの図 72 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 をカスタマイズする方法に関するプログラミング情報については、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。操作を再び試行したい場合は、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。229 ページの『PUSH HANDLE および POP HANDLE コマンドの使用』で説明されているように、**PUSH HANDLE** コマンドと **POP HANDLE** コマンドを使用すると、**HANDLE ABEND** コマンドを中断できます。

異常終了が処理される場合、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ません。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、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 コマンドで ASPx 異常終了コードや APSJ 異常終了コードをインターセプト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。

- 238 ページの『プログラム・レベルの異常終了プログラムまたはルーチンの作成』
- 239 ページの『操作の再試行』
- 240 ページの『トレース』
- 242 ページの『アプリケーション・パフォーマンスのモニター』
- 242 ページの『ダンプ』

プログラム・レベルの異常終了プログラムまたはルーチンの作成

プログラム・レベルの異常終了プログラムを作成することができます。プログラミング言語によっては、プログラム・レベルの異常終了ルーチンを作成することができます。

プログラム・レベルの異常終了プログラム

RDO を使用するか、またはプログラム自動インストール出口を使用することにより、異常終了プログラム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ある場合は、プログラム名に関するセキュリティ・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、サポートされている任意の言語でコーディングすることができます。

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るアプリケーション・プログラムについて通常想定される以外のアドレス可能性を想定することはできません。

プログラム・レベルの異常終了ルーチン

制約事項: C、C++、PL/I、および AMODE(64) アプリケーションでは **HANDLE ABEND LABEL** コマンドがサポートされないの、それらについては、このセクションは適用されません。

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する必要があります。

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。

異常終了出口ルーチンへの入り口におけるレジスター値は、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 (ただし、AMODE(64) アセンブラーを除く)

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。

処理の終了

異常終了出口ルーチンまたはプログラムにおける処理は、以下のいずれかの方法で終了させることができます。異常終了ルーチンおよびプログラムは、CICS 内部論理によって呼び出される場合は、処理を続行するとさらに問題が発生する可能性がありますので、異常終了させなければなりません。

- **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てタスクを続行するように指定します。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。
- **ABEND** コマンドを使用して、より高い論理レベルの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、異常終了処理のための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ように指定します。
- 分岐して操作を再試行します。この方法を使用して操作を再試行し、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する場合は、分岐する前に、異常終了出口ルーチンまたはプログラムで **HANDLE ABEND RESET** コマンドを発行する必要があります。なぜならば、出口ルーチンまたはプログラムは、異常終了出口に再入できないように CICS によって使用不能にされるからです。

未解決の **RECEIVE** コマンドがタイムアウトになった結果発生する異常終了の場合は、CICS が **RECEIVE** を取り消せるように CICS 異常終了が続行されるようにすることが重要です。

操作の再試行

CICS サービスの呼び出し中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ります。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ます。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。

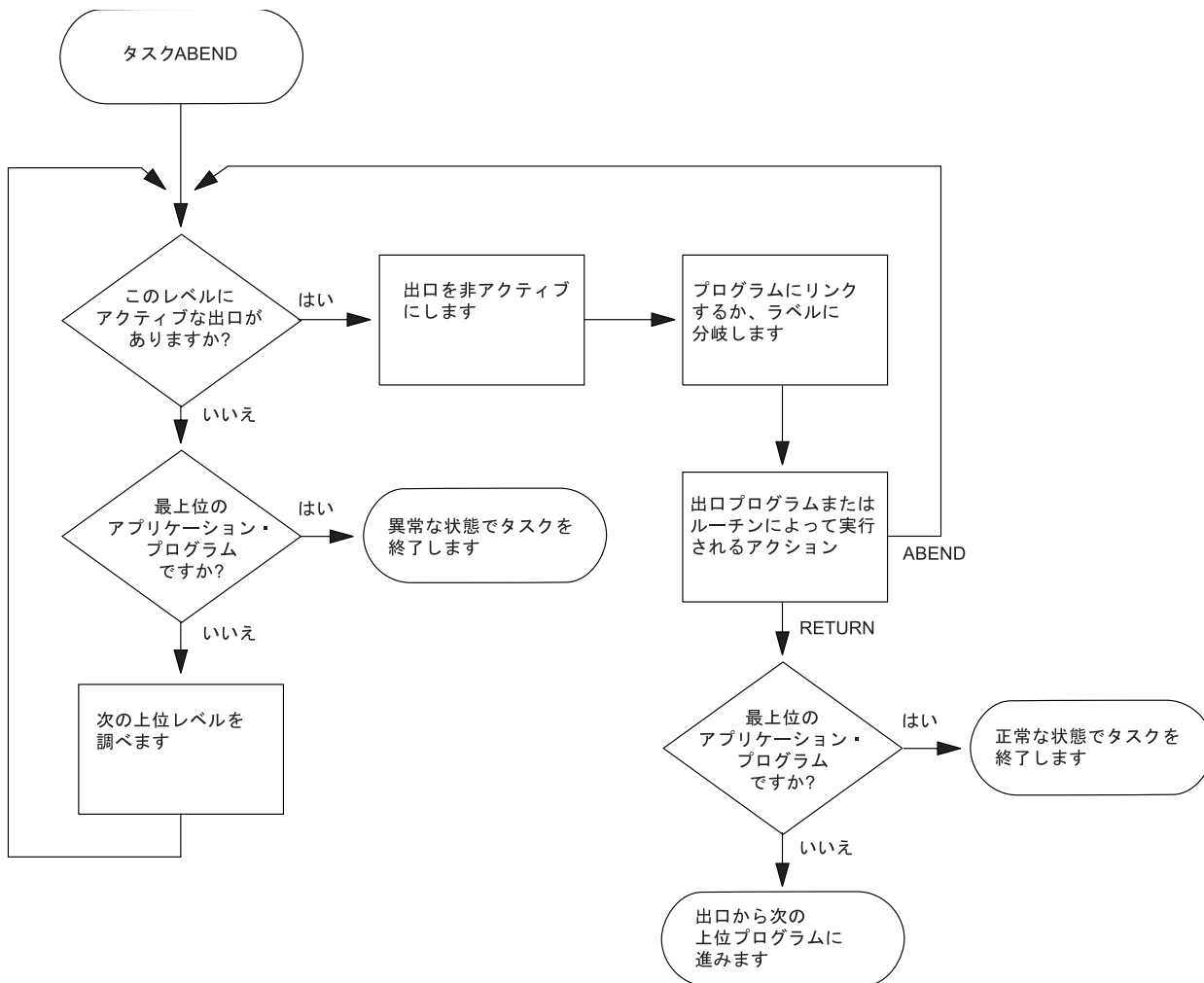


図 72. ABEND 出口の処理

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は、CICS トレースの使用を参照してください。

以下のことができます。

- 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。

トレースについて詳しくは、CICS トレース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・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、ユーザー例外トレース項目を参照してください。プログラミング情報については、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・ポイントで使います。

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 8192 バイト文字ストリング

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** コマンドによって、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 コマンドを使用すると、システム・ダンプを要求できます。

ダンプ・データ・セットの内容はフォーマット設定が可能です。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 (DFH DU720) 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ついての説明は、ダンプ・ユーティリティー (DFH DUnnn および DFH PDnnn) 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: CICS システム初期設定パラメーターと、CICS システム内の FILE、PROGRAM、TRANSACTION、および TERMINAL リソース定義のリスト。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良い場合があります。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領域なので、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プログラムのリストが出力されます。

QUERY SECURITY コマンドを使用した、リソースへのユーザーのアクセスの判別

EXEC CICS QUERY SECURITY コマンドを使用して、端末ユーザーが RACF などの外部セキュリティ・マネージャーに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 を使用して、**QUERY SECURITY** コマンドを含むトランザクションを呼び出すユーザーのセキュリティ許可を照会できます。この場合、USERID は指定されません。

また、**QUERY SECURITY** コマンドを含み、いずれかのユーザー ID で実行中のトランザクションで、別のユーザー ID のセキュリティ許可を照会することもできます。この場合、照会対象のユーザー ID は、**QUERY SECURITY** コマンドの USERID オプションに指定されている必要があります。CICS では、代理ユーザー検査を実行して、トランザクションを呼び出すユーザーが USERID に指定されたユーザーに対して許可されているかどうかを検査することに注意してください。

QUERY SECURITY コマンドに応答して、CICS は、ユーザーのセキュリティ権限に関する情報を返します。CICS は、外部セキュリティ・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。

典型的なユース・ケース: レコード・レベルまたはフィールド・レベルでのデータへのアクセスの制御

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。

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、**QUERY SECURITY** コマンドを使用したセキュリティ検査を参照してください。

その他の考慮事項

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。**RESID values**では、**SPCOMMAND** リソース・タイプで使用可能な **RESID** 値について詳しく説明しています。

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。

その他トピック

QUERY SECURITY コマンドを使用したセキュリティ検査では、アプリケーション・プログラムで **QUERY SECURITY** を使用して、ユーザーが特定のリソースに対して持つアクセス権のレベルを判別する方法について詳しく説明しています。

ユーザー定義リソースを使用するためのアプリケーションの設計

このトピックでは、ユーザー定義リソースを使用するようにアプリケーションを設計する方法の例を示します。

アプリケーションは、CICS ファイル制御を通常の方法で使用して、給与および個人の詳細ファイルからレコードを読み取ります。各レコード内の個々のフィールドを制御しているため、リソース・セキュリティをファイル・レベルで適用する必要はないかもしれません。したがって、トランザクションを定義する際に RESSEC(NO) を指定できます。ファイル・レコードを読み取った後、結果を表示する前に、QUERY SECURITY を使用して、ユーザーにレコード内の特定フィールドへのアクセス権限があるかどうかを判別します。例えば、給与額を表示する前に、以下のコマンドを発行します。

```
EXEC CICS QUERY SECURITY RESCLASS('$FILERECL')
                          RESID('PAYFILE.SALARY')
                          RESIDLENGTH(14)
                          READ(read_cvda)
```

これによりアプリケーションは、read_cvda で返された値に応じて、給与を表示するか、あるいはユーザーに給与の表示権限がないことを示すメッセージを表示します。同様に、個人の電話番号を更新するトランザクションの一部として、以下のコマンドを発行します。

```
EXEC CICS QUERY SECURITY RESCLASS('$FILERECL')
                          RESID('PERSONAL.PHONE')
                          RESIDLENGTH(14)
                          UPDATE(update_cvda)
```

update_cvda で返された値がユーザーに UPDATE アクセス権限があることを示す場合、トランザクションは続行し、ファイル内の電話番号を更新します。それ以外の場合は、ユーザーが電話番号の更新を許可されていないことを示します。

CICS の相互通信

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ときに考慮すべき領域について概説します。

CICS 相互通信について詳しくは、相互通信入門を参照してください。

以下の機能のうちの 1 つ以上を使用する CICS 相互通信環境でアプリケーション・プログラムを実行することができます。

トランザクション・ルーティング

トランザクション・ルーティングを使用すると、ある CICS システムの端末で別の CICS システムのトランザクションを実行することができます。

247 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

機能シップを使用すると、アプリケーション・プログラムから別の CICS システムのリソースにアクセスすることができます。 247 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)

DPL を使用すると、ある CICS 領域で実行されているアプリケーション・

プログラムからリモートの 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することができます。248 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

非同期処理を使用すると、CICS トランザクションからリモート・システムの別のトランザクションを開始し、必要な場合はデータをそのトランザクションに渡すことができます。262 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)

DTP を使用すると、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することができます。DTP に使用できるインターフェースは 2 つあります。それらは、コマンド・レベルの EXEC CICS インターフェースと、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 インターフェースです。262 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)

CPI-C は、APPC 接続で DTP を提供し、複数のシステム・プラットフォームで使用する API を定義します。262 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)

EXCI を使用すると、MVS で実行されている非 CICS プログラムで CICS システムに対してセッションを割り振ってオープンすることや、それらのセッションで DPL 要求を発行することができます。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。263 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。

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詳しくは、FEPI の概要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・プログラムで前述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する必要があります。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

通常は、COBOL、C、C++、PL/I、またはアセンブラー言語を使用して、CICS 相互通信機能を使用するアプリケーション・プログラムを作成することができます。ただし、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・プログラムの場合、使用できるのは C、C++、またはアセンブラー言語のみです。

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について詳しくは、CICS Interdependency Analyzer for z/OS の概要を参照してください。トランザクションの類縁性の詳細については、175 ページの『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、機能シップできません。

一時記憶域コマンド

リモート・システムの一時記憶域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場合 (IPIC の場合は AIPM 異常終了、ISC 接続の場合は ATNI、MRO の場合は AZI6) 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、機能シッパ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に (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 分散プログラム・リンクで説明されています。249 ページの図 73 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パする先の 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。

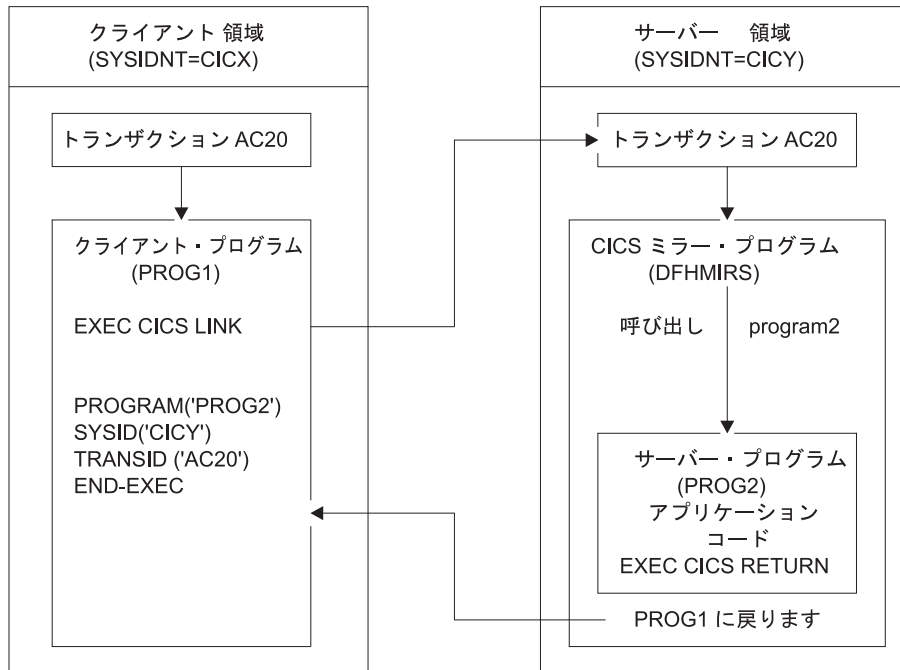


図 73. 分散プログラム・リンクの図

分散プログラム・リンク機能の使用

この表は、分散プログラム・リンク機能のオプションを理解するために使用します。

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、260 ページの表 24 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 22 に示したオプションが使用され、PROGRAM リソース定義には 表 23 に示したオプションが使用されています。

表 22. 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。
LENGTH	COMMAREA のバイト単位の長さをハーフワード・バイナリー値で指定します。LENGTH オプションに有効な値 (ゼロ以外) を指定してください。DPL 要求でゼロ長を使用すると、予測不能の結果になる場合があり、プログラム障害が発生する可能性があるためです。
SYSID	クライアント領域がプログラム・リンク要求をシッパする先の、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 コマンド・サマリー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 23. 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・リンク要求をシッパする先の、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、を参照してください。

表 23. DPL サポート用の PROGRAM リソース定義のオプション (続き)

キーワード	説明
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、コマンドの構文を示します。

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 74. 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、このリンク要求を、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他のすべてのプログラム名を参照せずに、サーバー領域にシップします。この状況では、LINK コマンドで指定する PROGRAM 名は、そのプログラムであることがサーバー領域で認識される名前でない限りなりません。

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。このオプションを使用すると、クライアント領域でサーバー・プログラムに渡す 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 には同じ長さで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、165 ページの『COMMAREA』を参照してください。

LENGTH オプションには、有効な値 (ゼロ以外) を確実に指定してください。DPL 要求でゼロの長さを使用すると、予測不能の結果になることがあり、プログラム障害が発生する可能性があるためです。

COMMAREA を使用する場合、リンクされているプログラムは、タスクの EIB の EIBCALEN フィールドが、プログラムの予期する内容に一致するか検証しなければなりません。詳しくは、165 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

以下の優先順位の規則に従います。

- 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 (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る場合)、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 (z/OS Communications Server 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) のロードも実行もできない場合、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、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関連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。

注: ミラー・トランザクションが Db2 にアクセスする場合、EIBTRNID を使用して、どの DB2ENTRY 定義を使用するかを決定します。

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。255 ページの図 75 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、クライアント・リソースがコミットされるときに、サーバー・リソースは同じ同期点の一部としてコミットされます。 256 ページの図 76 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。

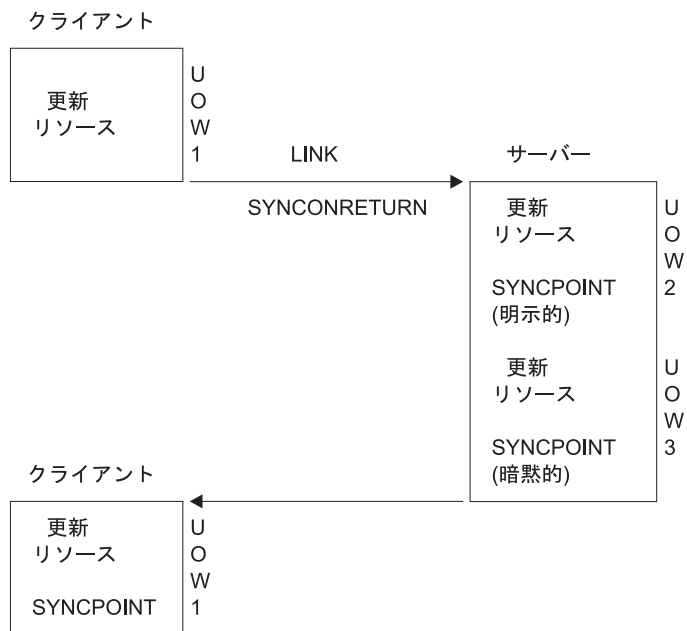


図 75. SYNCONRETURN オプションを使った分散プログラム・リンクの使用例

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。

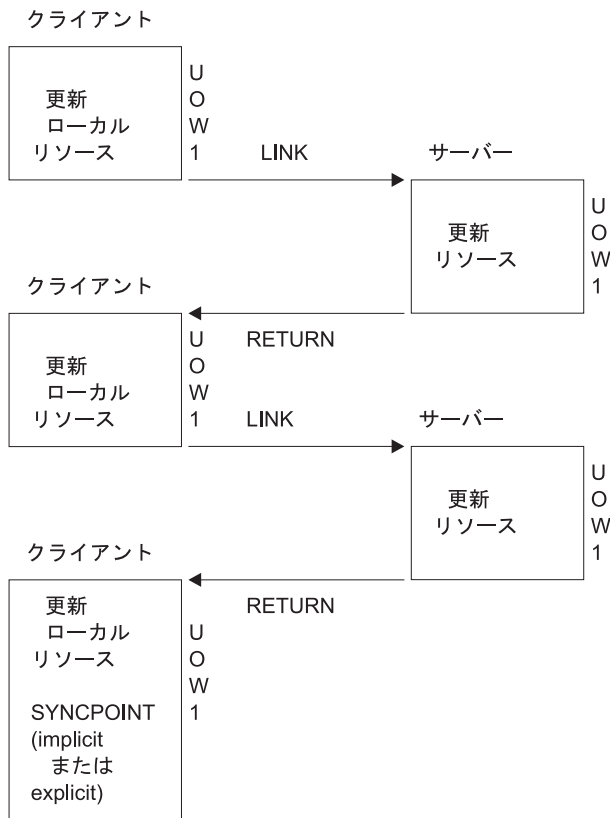


図 76. SYNCONRETURN オプションを使わない分散プログラム・リンクの使用例

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理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には、以下の要素を考慮してください。

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有

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パ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 と機能シッパの混合

同じクライアント・タスクから同一の CICS システムに対して、機能シッパと 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・タスクは要求を機能シッパできないため、同一セッション (同一の作業論理単位、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、または IPCONN MIRRORLIFE が UOW や TASK に設定されている) 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、要求を機能シッパし、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。
- 機能シッパされた要求が、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。
- 実行中の機能シッパされた要求が、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。機能シッパされた要求は、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。
- 実行中の機能シッパされた要求が、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。

機能シッパについて詳しくは、CICS 機能シッパ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。

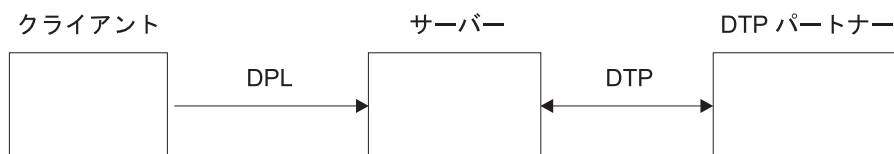


図 77. DPL と DTP を混合した例

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 260 ページの表 24 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。

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。ただし、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 API コマンド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・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. セッションでも接続定義でもユーザー 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で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接続要求で受け取ったユーザー ID です。

詳細については、Intercommunication securityを参照してください。

別のセキュリティ関連の考慮事項は、**ASSIGN** コマンドの **CMDSEC** オプションと **RESSEC** オプションの使用に関するものです。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:

クライアントおよびサーバー・プログラムに返される、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 API コマンド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

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 24 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。

EXEC CICS WEB コマンドをサーバーとして 使用しようとする、それらのコマンドは INVREQ および RESP2 値 1 で失敗します。EXEC CICS WEB EXTRACT、EXTRACT TCPIP および EXTRACT CERTIFICATE の 3 つのコマンドは、INVREQ および RESP2 値 5 で失敗します。CICS がクライアントとして機能している場合は、これらのコマンドを使用しても問題はありません。

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、LINKを参照してください。

表 24. 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 HIGHLIGHT 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 24. 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	TERMINID。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が端末ではなくセッション である。)
START CHANNEL	TERMINID。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が端末ではなくセッション である。)
SYNCPPOINT	LINK で SYNCONRETURN を指定した場合は、サーバー領 域で発行可能。
SYNCPPOINT ROLLBACK	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行進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を保有して、「会話」できることです。DTP が使用されるのは、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。

DTP を使用することにより、柔軟かつ効率的なアプリケーションを設計できます。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 によって、また、CICS 相互通信機能を使用するアプリケーションによって、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・プログラムで、C、C++、およびアセンブラ言語を使用することができます。

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、分散トランザクション処理の概要および相互通信入門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。

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) は、システム・アプリケーション体系 (SAA) 共通プログラミング・インターフェース (CPI) のコミュニケーション・エレメントです。

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、そのアプリケーションを 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されます。この方法については、1013 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』を参照してください。

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。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) 解説書を参照してください。

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 のプログラミング情報については、外部 CICS インターフェースの紹介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。

アプリケーション・プログラム用の CICS サービス

CICS は、ビジネス上重要なアプリケーションの構築を可能にする多数のサービスを提供します。

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・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、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) のタイプのデータ・セットへのアクセスをサポートします。

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。

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 4 GB 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 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 ESDS があり、これは 4 GB 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。

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 は索引を使用し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、前述の 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 CICS に対して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・セットを共用する 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・セットが共用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。(VSAM レコード・レベル共用の使用 を参照。)
- 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、常に見捨てられます。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

ユーザー・データ・セットの暗号化

z/OS データ・セットの暗号化がサポートされている CICS ユーザー・データ・セットを暗号化できます。

始める前に

z/OS データ・セットの暗号化がサポートされているデータ・セット・タイプを確認します。データ・セットの暗号化の計画を参照してください。

使用する鍵ラベルが設定されていることを確認します。「z/OS 暗号サービス ICSF 管理者ガイド」の『鍵生成プログラム・ユーティリティー・プログラムを使用した暗号鍵の管理』を参照してください。

暗号化されたデータ・セットを作成する権限および鍵ラベルへのアクセスを提供する RACF タスクが完了していることを確認します。以下のタスクがあります。

- STGADMIN.SMS.ALLOW.DATASET.ENCRYPT CL(FACILITY) への読み取り権限を付与して、ユーザーに暗号化されたデータ・セットの作成を許可します。
- 鍵ラベルへの読み取り権限を付与します。ユーザーが暗号化されたデータ・セットにアクセスしようとする、RACF は、CSFKEYS クラスおよび CSFSERV ク

ラス内の関連するプロファイルへのアクセスを検査することによって、暗号化の前にユーザーが鍵ラベルを使用する権限を持っているかどうかを検査します。これらのクラスについて詳しくは、「z/OS DFSMS データ・セットの使用法」の『データ・セット暗号化』を参照してください。

使用する鍵ラベルが設定されていることを確認します。「z/OS 暗号サービス ICSF 管理者ガイド」の『鍵生成プログラム・ユーティリティー・プログラムを使用した暗号鍵の管理』を参照してください。

このタスクについて

CICS に定義されたユーザー・データ・セットの場合、暗号化サポートには、基本 VSAM および VSAM レコード・レベル共用 (RLS) を介してアクセスされるキー順データ・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)、および可変相対レコード・データ・セット (VRRDS) が含まれます。共用データ・テーブルまたはカップリング・ファシリティ・データ・テーブルに使用されるバッキング VSAM キー順データ・セットについても、暗号化がサポートされます。

手順

1. 暗号鍵および関連付けられている鍵ラベルを作成するか、既存の鍵ラベルを使用します。
2. 拡張フォーマットおよびデータ・セット鍵ラベルを指定して、データ・セットの新規インスタンスを割り振ります。
 - a. 以下のいずれかの方法で、拡張フォーマットを指定します。
 - SMS データ・クラスで、**DSNTYPE=EXT** パラメーターおよび **R** (必須) または **P** (優先) サブパラメーターを使用します。データ・セットが拡張フォーマットで割り振られるようにするには、**R** を指定します。
 - JCL DD ステートメントで **DSNTYPE** パラメーターを使用し、**EXTREQ** (必須) または **EXTPREF** (優先) の値を指定します。データ・セットが拡張フォーマットで割り振られるようにするには、**EXTREQ** を指定します。JCL DD ステートメントで指定した **DSNTYPE** は、データ・クラスのどの **DSNTYPE** セットよりも優先されます。

拡張フォーマットを指定する場合、拡張アドレッシング属性は、データ・セットにそのオプションも必要な場合にのみ指定してください。

 - b. 以下のいずれかの方法で、データ・セット鍵ラベルを指定します。
 - RACF データ・セット・プロファイルで、**DATAKEY** キーワードを使用します。
 - JCL で、**DSKEYLBL** キーワードを使用します。
 - 動的割り振りで、**DALDKYL** テキスト単位を使用します。
 - TSO 割り振りで、**DSKEYLBL** を使用します。
 - IDCAMS DEFINE で、**DEFINE CLUSTER** の **KEYLABEL** パラメーターを使用します。
 - SMS データ・クラスで、「定義/変更」パネルの「データ・セット鍵ラベル (**Data Set Key Label**)」フィールドを指定します。

データ・セット鍵ラベルが複数のインターフェースで指定されている場合、優先順位は以下のようになります。

- 1) RACF データ・セット・プロファイル
- 2) JCL、動的割り振り、TSO 割り振り、または IDCAMS DEFINE
- 3) SMS データ・クラス

鍵ラベルの管理の利便性とデータ保護の最大化のバランスが取れるように、暗号化を計画しているデータ・セットのグループ全体で使用するさまざまな鍵ラベルの細分度を考慮してください。これは、鍵ラベルの指定に使用する適切なメカニズムを決定する際にも役立ちます。

3. 既存のデータ・セットからデータをコピーして、新しい暗号化されたデータ・セットに入れます。REPRO などの適切な DFSMS 機能や EXPORT および IMPORT 機能を使用してください。一般的な手順は以下のとおりです。
 - a. 新規データ・セットを作成します。
 - b. 古いデータ・セットからデータをコピーします。
 - c. 古いデータ・セットを削除します。
 - d. 新規データ・セットの名前を古い名前に戻します。

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』
- 271 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:

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。ただし、レコードを KSDS に書き込む場合、または代替索引パスによってレコードを書き込む場合は、コマンドの RIDFLD オプションに完全キーを指定する必要があります。

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同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、完全キーの長さと同じ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 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 ESDS があり、これは 4 GB 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。『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。

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。AMS REPRO 関数を使用して、これを実行できます。
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・クラスの定義方法については、z/OS DFSMSdfp Storage Administrationで説明しています。

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. 以下のすべてのコマンドで、RBA キーワードを XRBA キーワードで置き換えます。
 - 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 5.5 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 5.5 ファイル専用領域 (FOR) のファイルにアクセスしようとしています。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 応答を受け取ります。

バックレベル FOR へのCICS TS for z/OS, バージョン 5.5 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ています。

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 応答を受け取ります。

リカバリー可能ファイルにおける VSAM レコードのロック

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。

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、288 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 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 を使用したり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实际的でなく、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 (277 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 (274 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ではありません。

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 (277 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に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 78. 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 では、ロックのアクティブ状態と保存状態がサポートされます。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本性は、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、データ保全本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。

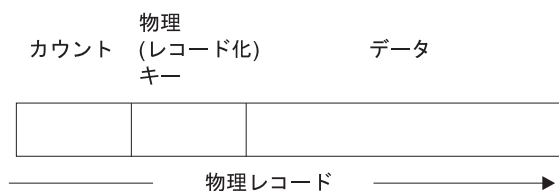


図 79. BDAM データ・セットの非ブロック化レコードの形式

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 80. 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、264 ページの『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、前述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。

282 ページの図 81 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

Byte Number															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
RELBLK#			N												
RELBLK#			KEY												
T	T	R	PH-KEY				KEY								
M	B	B	C	C	H	H	R	N							
T	T	T	T	T	T	R	R	PH-KEY				KEY			
T	T	R	KEY												

Search by relative block;
deblock by relative record

Search by relative block;
deblock by key

Search by relative track
and record and key;
deblock by key

Search by actual address;
deblock by relative record

Search by zoned decimal
relative track and record
and key; deblock by key

Search by relative track
and record; deblock by key

図 81. BDAM レコード識別の例

CICS 共用データ・テーブル

CICS 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。共用データ・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、「共用データ・テーブルの概要」を参照してください。

テーブルは FILE リソース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ません。

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・

セットから切り離されます。 テーブルに対する変更内容は、自動的にソース・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。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・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・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・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・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、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・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・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・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

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・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・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・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・データ・テーブル・プールは、カップリング・ファシリティ・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・データ・テーブルのリスト構造の作成については、カップリング・ファシリティ・データ・テーブル・サーバーのパラメーターを参照してください。

- 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・データ・テーブルのモデル

カップリング・ファシリティ・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

データの共用に使用できる CICS の手法が表形式で比較され、カップリング・ファシリティ・データ・テーブルの使用をいつ検討するかを示しています。

表 25. 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領域に推奨される方式	COMMAREA またはチャンネル	COMMAREA またはチャンネル	COMMAREA またはチャンネル
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 複数タイプの保管されているデータ	UMT	リモート UMT	CFDT (競合モデル)

表 25 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・テーブルを意味します。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 26. 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 26 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。ほとんどの場合、CICS 一時データおよび一時記憶域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 27. 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 27 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してください。この機能もシスプレックス全体の機能です。詳しくは、409 ページの『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・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 28. 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 28 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・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・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・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。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。

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。

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・ロックの解放方法について詳しくは、ロックを参照してください。)

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、215 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、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 AFCE で、また、自らデッドロックした場合、異常終了コードは AFCEG 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

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有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:

デッドロックを回避するには、以下の規則を使用します。

- 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。
- ファイルに対して READ UPDATE、WRITE、または DELETE (RIDFLD を指定した) コマンドを発行する前に、アプリケーションは、ENDBR コマンドでファイルのブラウズをすべて終了させる (VSAM ロックを解放する) 必要があります。

ファイル制御操作

CICS ファイル制御には、データ・セット VSAM および BDAM 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できるようにするオプションがあります。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・データ・テーブルにアクセスすることもできます。

VSAM リソースを待機した後の CICS の処理

CICS ファイル制御は、リソースが使用可能になったときに、リソースを待機するなどのタスク (制御インターバルの VSAM 排他制御など) を再ディスパッチするかを編成しません。VSAM が制御インターバルの排他制御を保留解除すると、CICS ディスパッチャーは、タスク優先順位や優先順位繰り上げなどの要素に基づいて次にディスパッチするタスクを選択します。リソースで中断されたタスクが使用可能になる順序でタスクが再ディスパッチされるとは限りません。

CICS コマンドを使用したレコードの読み取り:

アプリケーション・プログラムがレコードを読み取る方法は、READ コマンドを使用した直接読み取り、順次読み取り、およびスキップ順次読み取りの 3 つあります。これらのレコードの読み取り方法はすべて、VSAM データ・セットと BDAM データ・セットの両方で使用できます。

このタスクについて

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。次の場合にのみ、固定長レコードを定義します。

- (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 (通常は、ファイルの最大レコード長) 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):

レコードを指定するために、RIDFLD (レコード識別フィールド) オプション付きで READ コマンドを使用し、ファイル内のレコードを読み取ることができます。レコード識別フィールドの内容および長さを記述する追加のオプションを利用できます。

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。

手順

- 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、一致する必要が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 (293 ページ) を参照してください)。

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、読み取りの保安全性はなく、共用ロックは読み取り要求のためには使用されません。 275 ページの『RLS レコード・レベルのロック』に、共用ロックと排他ロックについての説明があります。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。215 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く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。277 ページの『ロックのアクティブ状態および保存状態』に、アクティブ・ロックについての説明があります。

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、読み取りの保安全性を指定してください。これは、読み取りの保安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安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、読み取りの保安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):

STARTBR、**READNEXT**、**READPREV**、および **RESETBR** の各コマンドを使用して、ファイル内のレコードをブラウズします。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。**ENDBR**、**SYNCPPOINT** および **SYNCPPOINT ROLLBACK** の各コマンドを使用して、ブラウズを終了します。

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 (292 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。271 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。これは、**READNEXT** コマンドと **READPREV** コマンドのどちらを使用している場合にも当てはまります。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、299 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。

手順

- 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。292 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。
- 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。292 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. SYSID オプションが指定されているリモート・ファイルについては、RIDFLD オプションがキーをファイル制御に渡している場合、KEYLENGTH オプションを指定する必要があります。リモート・ファイルが参照されている場合は、**READNEXT** コマンドと **READPREV** コマンドに KEYLENGTH オプションを指定する必要がありません。
 4. 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、いつでも切り替えられます。以下の考慮事項に注意してください。
 1. 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 2 回取り出されます。
 2. **STARTBR** コマンドの直後に **READPREV** コマンドを発行する場合、**STARTBR** コマンド **RIDFLD** でデータ・セットに存在するレコードのキーを指定する必要があります。そうでない場合は、**NOTFND** 条件が発生します。この応答を回避するには、データ・セット上に存在するレコードを返すために、**READNEXT** コマンドを **STARTBR** コマンドの後に発行することもできます。次に、**READPREV** コマンドを発行します。このコマンドは **READNEXT** コマンドによって取得されたものと同じレコードを取り出します。
 3. **KSDS** の場合、**STARTBR** コマンドで総称キーを使用すると、**READPREV** コマンドは使用できません。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、300 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う ENDFILE 条件が起こります。同じファイルに対する更新操作 (**READ UPDATE**、RIDFLD を指定した **DELETE**、または **WRITE** コマンド) を実行する前に、**ENDBR** コマンドを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。また、**SYNCPPOINT** コマンドおよび **SYNCPPOINT ROLLBACK** コマンドを使用して、ブラウズを終了することもできます。タスクの終了時に暗黙的な同期点が発生すると、ブラウズも終了します。

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、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 ID では更新されません。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同しく設定されます。

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、最初から位置変更するのではなく、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:

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同じ方法で識別されます。

このタスクについて

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、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ています。(TOKEN オプションの詳細な説明は、302 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 (305 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な値であり、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 (305 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。 **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。

292 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。

手順

- 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、305 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同じ場合は、(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。
- **DELETE** コマンドに **GENERIC** オプションを付けるときには注意が必要です。指定された総称キーに非常に多数のレコードが一致する場合、CICS の作業単位は、同期点まで非常に多数のレコード・ロックを保持することになります。これによって、CICS 領域内のストレージの問題が発生する可能性がありますし、ファイルが RLS ファイルである場合には、カップリング・ファシリティーのリソースが過度に使用されて他のシステムに影響する可能性があります。代わりに、

一連の総称削除 (一致するレコードがそれほど多くない総称キーを使用する方法) と、レコードの各範囲が削除された後の同期点を使用する方法を考慮してください。

ブラウズ中のレコードの更新および削除 (**VSAM RLS** のみ):

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。

このタスクについて

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、302 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、275 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:

VSAM データ・セットの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。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。

このタスクについて

ここでの説明は、レコードの書き込みに関する一般的な原則と、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を網羅しています。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、308 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。

手順

- 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。277 ページの『ロックのアクティブ状態および保存状態』に、アクティブ・ロックについての説明があります。
- 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 操作は、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 について詳しくは、309 ページの『効率的なデータ・セットの操作』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:

BDAM データ・セットに追加されるレコードは、未定義にすることも、可変長 (キー付きまたはキーなし)、キー付き固定長、キーなし固定長、可変長ブロック化にすることもできます。

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。以下の情報を利用して、VSAM データ・セットおよび BDAM データ・セットを使用した応答時間を最大化します。

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・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。

267 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ときなど) 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、FILE 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 29 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 29. 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・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・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

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ため、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 BDM 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。DELAY または SUSPEND コマンドを発行することにより、他のタスクが制御できるようにすることが可能です。

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、567 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。

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、端末管理コマンドと基本マッピング・サポート (BMS) 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。

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、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。

- 『端末アクセス方式のサポート』
- 313 ページの『端末制御コマンド』
- 316 ページの『データ伝送コマンドの使用』
- 318 ページの『装置制御コマンド』
- 319 ページの『端末装置のサポート』
- 323 ページの『ご使用の端末情報の検出』
- 326 ページの『SNA の使用』
- 330 ページの『順次端末サポートの使用』
- 332 ページの『バッチ・データ交換の使用』

端末アクセス方式のサポート

CICS Transaction Server for z/OS, バージョン 5 リリース 5 は、以下のアクセス方式へのインターフェースを介して、直接端末をサポートします。すなわち、z/OS

Communications Server、GDDM を使用したグラフィックス端末用の基本グラフィックス・アクセス方式 (BGAM)、および順次装置によってシミュレートされた端末用の順次アクセス方式 (SAM) です。

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、DFHTCT: CICS 端末リストに記載されています。

端末制御コマンド

端末制御コマンドは、基本的なデータ伝送、制御機能、および端末に関する情報の取得で使用されます。また、端末管理は、特殊装置グループ・コマンドも提供します。

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
- z/OS Communications Server 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。これについては、APPC 基本会話のコマンドの要約で説明されています。

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:

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 SNA 用の z/OS Communications Server によって形式が定義され、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。Communications Server の下で、タスク間の会話は、パートナーのどちらでも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 (69 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。

Communications Server またはネットワーク・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。

ただし、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 間で競合が起きます。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:

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な場合があります。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 (329 ページの『ブラケット・プロトコルを使用した割り込みの防止』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からです。

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のでしょうか? z/OS Communications Server の下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)。先読みキューイングは 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 接続と z/OS Communications Server 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、先読みキューイング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:

z/OS Communications Server は、受信モードになっている端末が送信しようとしてい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、Communications Server 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、z/OS Communications Server 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、z/OS Communications Server は割り込みを無視します。

端末待ち:

SEND コマンドで 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。

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、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。

データ伝送コマンドの使用

3 つのコマンドが、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います。

これらのコマンドは次のとおりです。

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 321 ページの表 32 にあります。

SEND、RECEIVE、および CONVERSE の各コマンドの詳細については、CICS コマンド・サマリー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。319 ページの『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、アプリケーションは端末によって送信されたデータを受信します。例えば、3270 ディスプレイはバッファ内では変更されたものはなんでも単一エンティティとして送信し、通常、端末に関連したタスクが、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。

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、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、326 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、 327 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

RECEIVE 当たりに 1 個の入力メッセージという規則に対する例外が起こるのは、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 (つまり、X'FF' に設定される) まで RECEIVE コマンドを発行してください。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、通常を送信/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、 447 ページの『表示画面の印刷』の『CICS 印刷キー』を参照してください)。

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。 319 ページの『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 30. 端末および論理装置用の制御コマンド

コマンド	アクション
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できるように、その端末をタスクから解放します。

表 30. 端末および論理装置用の制御コマンド (続き)

コマンド	アクション
ISSUE COPY	TERMINAL オプションに名前が指定されている端末のバッファの内容を、タスクが所有している端末のバッファにコピーします。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 z/OS Communications Server 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。

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。コマンドによっては、適用されるオプションと発生する可能性がある例外条件を判別するため、端末のタイプがわかっていることが必要な場合があります。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にも必要になります。以下の表に、どの端末とアクセス方式の組み合わせにどのコマンドが適用されるかを示します。複数タイプの端末をサポートする必要がある場合には、323 ページの『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 320 ページの表 31 の 1 列目から探し出してください。2 列目の値を使用し、321 ページの表 32 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 列目は使用可能な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 コマンド・サマリーにあります。1 つのコマンドについて複数のバージョンがある場合には、どのバージョンを使用すべきかが表に示されています。該当の情報はコマンドの後の括弧内で確認できます。

表 31. 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	3767
3270 ディスプレイ、3270 プリンター	LU タイプ 2/3
3270 ディスプレイ、3270 プリンター (非 SNA)	3270 論理
3270 ディスプレイ、3270 プリンター	3270 ディスプレイ
SCS プリンター	SCS
3600 パイプライン・モード	3600 パイプライン
3601	3600-3601
3614	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 (3767 として接続)	3767
3643 (LU タイプ 2 として接続)	LU タイプ 2/3
3642、3645 (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
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	3767
3770 対話式 LU)	3767
3770 全機能 LU	3790 全機能または照会
3770 バッチ LU (3771、3773、3774)	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 (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	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

表 31. CICS がサポートする装置 (続き)

装置	コマンドの使用対象
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) (3770 としてサポート)	3770 入力を使用
System/34 (5340) (3770 としてサポート)	3770 入力を使用
System/34 (5340)	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
System/38 (5381) (3770 として接続)	3770 入力を使用
System/38 (5381) (APPC として接続)	APPC
TWX 33/35 NTO	3767
WTTY NTO	3767

表 32. 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 z/OS Communications Server	RECEIVE (2260)、SEND (2260)、CONVERSE (2260)、ISSUE DISCONNECT (デフォルト)、ISSUE RESET
3270 ディスプレイ	非 z/OS Communications Server	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)	z/OS Communications Server	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)	z/OS Communications Server	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	z/OS Communications Server	SEND (SCS)、CONVERSE (SCS)、ISSUE DISCONNECT (デフォルト)、ISSUE PASS

表 32. 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
3600 パイプライン	z/OS Communications Server	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	z/OS Communications Server	RECEIVE (3600-3601)、SEND (3600-3601)、CONVERSE (3600-3601)、ISSUE DISCONNECT (デフォルト)、ISSUE PASS、WAIT SIGNAL
3600-3614	z/OS Communications Server	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	z/OS Communications Server	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	z/OS Communications Server	RECEIVE (3650)、SEND (3650-3270)、CONVERSE (3650-3270)、ISSUE DISCONNECT (デフォルト)、ISSUE ERASEAUP、ISSUE PASS、ISSUE PRINT
3650-3653	z/OS Communications Server	RECEIVE (3650)、SEND (3650-3653)、CONVERSE (3650-3653)、ISSUE DISCONNECT (デフォルト)、ISSUE PASS
3650-3680	z/OS Communications Server	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	z/OS Communications Server	RECEIVE (3767)、SEND (3767)、CONVERSE (3767)、ISSUE DISCONNECT (デフォルト)、ISSUE PASS、WAIT SIGNAL
3770	z/OS Communications Server	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	z/OS Communications Server	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	z/OS Communications Server	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	z/OS Communications Server	SEND (3790 3270 プリンター)、ISSUE DISCONNECT (デフォルト)、ISSUE ERASEAUP、ISSUE PASS
3790 SCS	z/OS Communications Server	SEND (3790 SCS)、ISSUE DISCONNECT (デフォルト)、ISSUE PASS

表 32. 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
LU タイプ 4	z/OS Communications Server	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)	z/OS Communications Server	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	z/OS Communications Server	RECEIVE (SNA 用 Communications Server デフォルト)、SEND (SNA 用 Communications Server デフォルト)、CONVERSE (SNA 用 Communications Server デフォルト)、ISSUE PASS
他のすべて	非 z/OS Communications Server	RECEIVE (非 z/OS Communications Server デフォルト)、SEND (非 z/OS Communications Server デフォルト)、CONVERSE (非 z/OS Communications Server デフォルト)

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 33 に、端末管理操作に直接関連するオプションをリストしています。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 コマンド・サマリー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 33. 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)

表 33. 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
DESTID	外部宛先の ID およびその長さ (BDI 操作専用)
DESTIDLENGTH	
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	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
GCODES	
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
NATLANGINUSE	現行タスクに使用中の各国語
NETNAME	SNA 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	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
OPCLASS	
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	現行タスクの端末画面の高さと幅
SCRNWD	
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	端末のステーションおよびテラー (2980 専用)
TELLERID	
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

表 33. 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
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。

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、表 34 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 コマンド・サマリー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 34. 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
EIBRID	最終入力操作からのアテンション ID (AID) (3270 専用。348 ページの『3270 端末からの入力』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

表 34. 端末制御コマンドに適用される EIB フィールド (続き)

フィールド	内容
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 (327 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	直前に実行したコマンドからの CICS 応答コード値
EIBRESP	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 (316 ページの『端末待ち』を参照してください)。
EIBRESP2	
EIBSIG	端末が SIGNAL を送信したかどうか
EIBTRMID	端末の (CICS) ID

SNA の使用

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:

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:

SNA 用 **z/OS Communications Server** は、アウトバウンドおよびインバウンドの端末データのチェーニングをサポートします。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SEMB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:

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 (入力に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SEMB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:

SNA 用 z/OS Communications Server の下で、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しなうヘッダー・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。

ヘッダーは、メッセージとその取り扱い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 定義を確認して、FMH を除去するか、あるいは RECEIVE を発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 による入力用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せん。

ブラケット・プロトコルを使用した割り込みの防止:

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、SNA 用の z/OS Communications Server の下で、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、Communications Server 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。

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。481 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 444 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、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。

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、各入力が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 (これはトランザクションの異常終了の原因になります) のを防ぐために、最後の入力

を CESF GOODNIGHT トランザクションにすることができます。これにより、端末をサインオフし、サービス休止にします。

または、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。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 が元の入力进行处理してしまうと、順次端末からさらに入力することはできません。

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 ガイド」を参照してください。図 82 では、バッチ・データ交換の概要を説明しています。

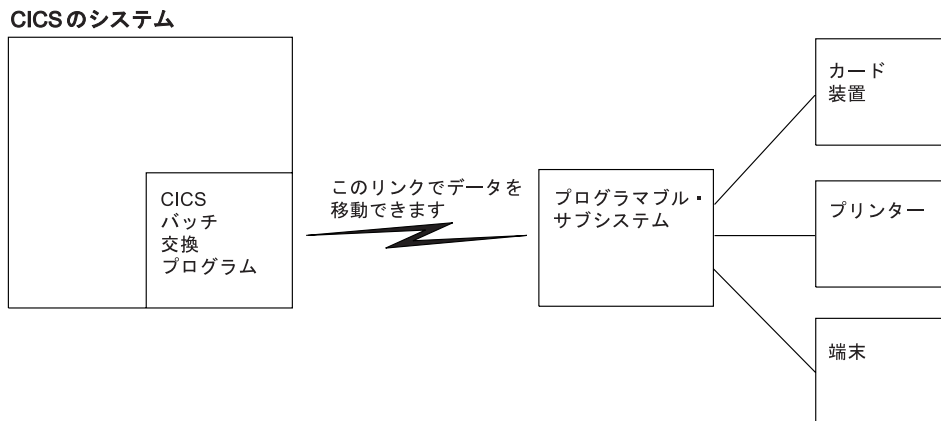


図 82. CICS バッチ・データ交換

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)。

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、217 ページの『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。ディスク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クットを識別します。VOLUME オプションが複数ディスク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、すべてのディスク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。

端末管理: パフォーマンスの設計

このリストは、パフォーマンスについて設計する場合に、端末管理で考慮する必要がある事項を示しています。

- 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、テレプロセッシング・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・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、プロセッサ時間の節約になります。

- メッセージ保安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。

- 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。

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、共通する特性を共用し、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、IBM 3270 Data Stream Programmers 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、556 ページの『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せん。

しかし、プリンターで利用できる追加の形式制御機能があり、印刷出力を意図したプリンターに送る場合には特別な考慮事項があります。詳しくは、431 ページの『印刷とスプール・ファイル』を参照してください。

3270 バッファ:

3270 装置との通信は、その文字バッファを介して行われます。これはプロセッサ内のメモリ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へ送られます。次に、バッファ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、348 ページの『3270 端末からの入力』に説明されているとおり、キーボード入力はバッファ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内の位置と対応し、そのバッファ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・ストレージの区域に補足情報が保持されます。表示特性について詳しくは、338 ページの『表示特性』を参照してください。

出力データ・ストリーム:

3270 表示を作成するためには、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)、および表示データ (可変数のバイト) から成るデータのストリームを送信します。

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るか、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 内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 バッファに送ります。 **erase/write** と **erase/write alternate** もデータを送信しますが、最初にバッファを消去します（すなわち、バッファ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・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。 **erase/write** コマンドは、(次の **erase/write** コマンドまたは **erase/write alternate** コマンドまで) デフォルトのサイズが後続の操作で使用されるようにします。 **erase/write alternate** コマンドによって、代替サイズが選択されます。

CICS では、**SEND** コマンドに **ERASE** オプションを指定しない限り、以下のように **write** コマンドを使用してデータが送信されます。

- **ERASE** を指定すると、CICS は、実行中のトランザクションに関連付けられている **PROFILE** 定義を使用して、**erase/write** と **erase/write alternate** のどちらを使用すべきか判別します。
- **ERASE DEFAULT** を指定すると、CICS は、**erase/write** コマンドを使用して、画面をデフォルトのサイズに設定します。
- **ERASE ALTERNATE** を指定すると、CICS は、**erase/write alternate** を使用して、画面を代替サイズに設定します。

erase unprotected to address コマンドにより、無保護フィールドに関するバッファのスキャンが実行されます。無保護フィールドのより正確な定義については、339 ページの『3270 フィールド属性』を参照してください。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、493 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』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コマンドによって、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 で **SEND MAP** コマンドに **ALARM**、**FREEKB**、**FRSET**、および **PRINT** の各オプションを指定して、**WCC** が作成されます。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** が CICS によって生成されます (これらのタグについては、339 ページの『3270 フィールド属性』で説明しています)。

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。

通常的环境のもとでは、351 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、バッファ内での開始位置)

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 つのカテゴリに分類されます。

フィールド属性

以下のものが含まれます。

- 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、339 ページの『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、グラフィカル・ユーザー・インターフェースを設計するときには、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。341 ページの表 35 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:

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。

ビット割り当てについては、IBM 3270 Data Stream Programmers 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、490 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。それら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があるものの、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、次にキー・ストロークする場所を指示します。この点の詳細については、348 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、前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どちら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、 560 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を使用して、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:

前に説明したフィールド属性に加えて、3270 端末の中には、拡張属性を持つものがあります。次の表には、1 列目に拡張属性のタイプが、2 列目にタイプごとの使用可能な値がリストされています。

表 35. 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
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 Data Stream Programmers Referenceでは、拡張属性の詳細、およびデフォルト値の決定方法について説明しています。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、 323 ページの『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、530 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、344 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:

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のに必要な端末管理コマンドを使用して、3270 に書き込みます。

このタスクについて

BMS を使用する場合には、このすべてが行われるので、348 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 83 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。

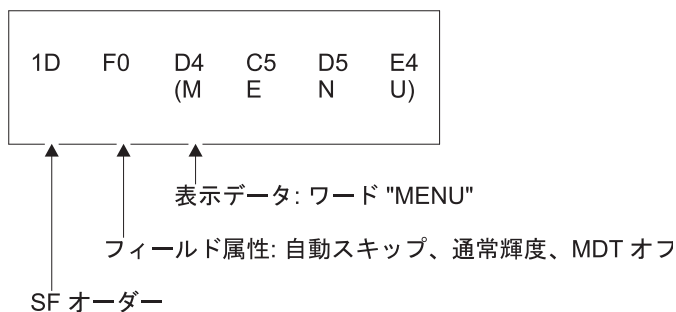


図 83. SF オーダーを使用したフィールド定義

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会的に取り扱われ、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、343 ページの図 84 に示されています。

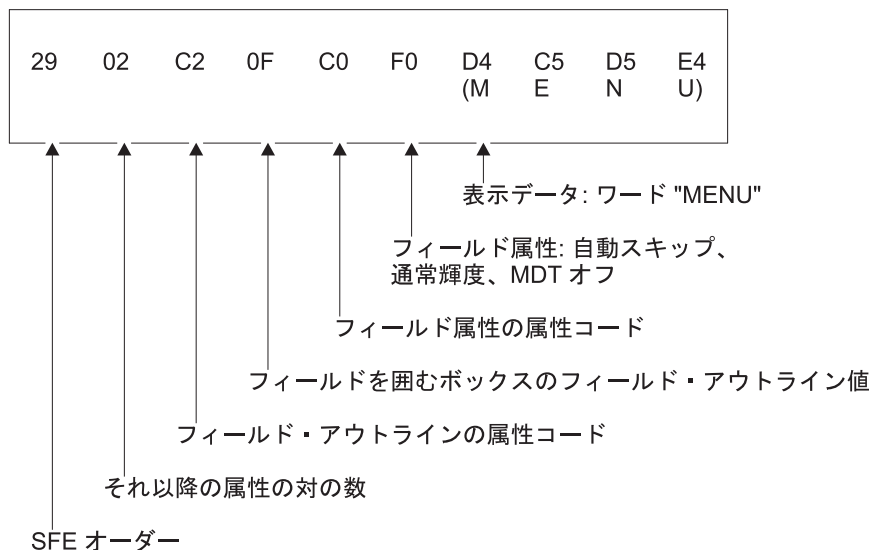


図 84. SFE オーダーを使用したフィールド定義

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。
- スル値は、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 85 に示されているシーケンスが必要です。

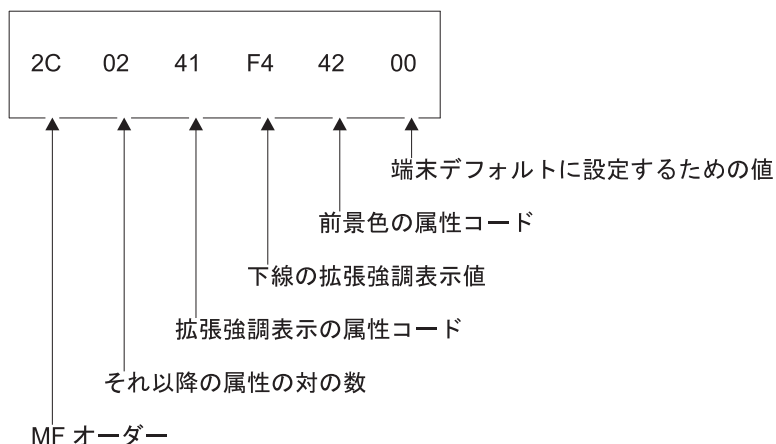


図 85. MF オーダー内のフィールド属性の変更

バッファ・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でない (すなわち、現行バッファ位置から始まっている) 限り、バッファ・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、図 86 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。

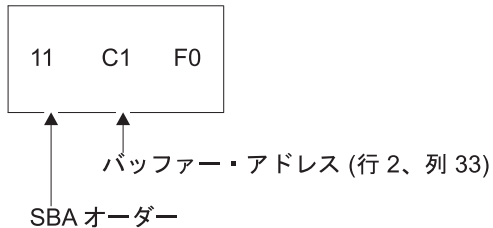


図 86. SBA シーケンス

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、行に沿って進みます。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 Data Stream Programmers Referenceでは、アドレスの形式を設定する方法について説明しています。

SF、SFE、または MF オーダーの後、現行バッファ・アドレスは埋め込まなかったバッファ内の先頭位置 (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、345 ページの図 87にあるシーケンスが必要です。

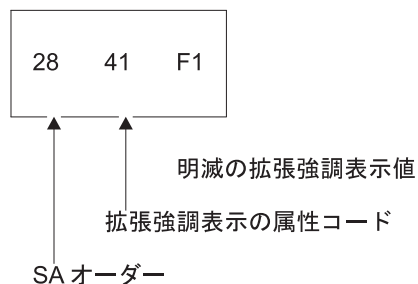


図 87. 文字を明滅させるための SA シーケンス

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位置に配置されるのと同じ方法で現行バッファ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:

アウトバウンド・データ・ストリームの例では、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 88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 88. 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)。

346 ページの表 36 に、アウトバウンド・データ・ストリームを示します。

表 36. 3270 出力データ・ストリーム

バイト	内容	注
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。
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ため、SF の代わりに SFE が必要です。このフィールドは、直前のフィールドの直後から開始されるので、前に SBA シーケンスを付ける必要ありません。
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、オペレーターがデータを入力する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。

表 36. 3270 出力データ・ストリーム (続き)

バイト	内容	注
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。
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、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:

キーボード入力はバッファ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、データ・キー、キーボード制御キー、またはアテンション・キーの 3 つのカテゴリに分類されます。

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。アテンション・キーは、バッファ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
- F (ファンクション) 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、以前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) を前もって適切にセットアップする必要があります。これらの機能は、556 ページの『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 つのファンクション・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ファンクション・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できます。)

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 キーおよびファンクション・キーはデータをバッファから伝送します。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ファンクション・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、351 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、バッファ読み取りと変更読み取りの 2 つの基本読み取りコマンドがあります。

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。

バッファ読み取りコマンドは読み取りヘッダーに続きバッファ全体を入手し、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、556 ページの『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、439 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:

端末管理コマンドを使用している場合には、3270 がデータを送信する形式に注意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、351 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・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、 559 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:

この例は、直前のアウトバウンド・データ・ストリームの例から、データ入力画面を使用して作成された入力データ・ストリームを示します。

インバウンド・データ・ストリームを説明するために、 345 ページの図 88 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
- 最後の州フィールドを指定せずに 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 37. 3270 入力データ・ストリーム

バイト	内容	注
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: 最後のデータ・キー・ストロークの後にオペレーターが使用した場所である、3 行目の 38 桁目。
4	X'11'	後にバッファ・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・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。 8 文字の長さのフィールドから取得されたのが 6 文字だけであることに着目してください。これは、残りのヌル桁が入力されなかったためです。

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ません。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) 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、513 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 キーはバッファをヌルに設定し、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。

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サービスにアクセスする Java プログラムについては、JCICS を使用した Java の開発および JCICS Javadoc の資料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 ファウンデーション・クラスの使用法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について詳しくは、CICS Interdependency Analyzer for z/OS の概要を参照してください。また、トランザクションの類縁性について詳しくは、175 ページの『類縁性』を参照してください。

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 オプションを使用し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。

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。以下に例を示します。

```
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ので、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

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。

タスク制御

CICS タスク制御機能は、タスク・アクティビティ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は、**JCICS** を使用した **Java** の開発および **JCICS Javadoc** の資料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** ファウンデーション・クラスの使用法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、プロセッサを集中的に使用するタスクによってプロセッサ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、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。

- 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について詳しくは、CICS Interdependency Analyzer for z/OS の概要を参照してください。また、トランザクションの類縁性について詳しくは、Affinity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 を使用します。

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 89 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、358 ページの図 90 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します。

```
Task A Task B
一時記憶域キューを削除する
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
Post ECB2 ClearECB2
Go to start of loop Read TS queue
Process data
Delete TS queue
Go to start of loop
```

図 89. 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

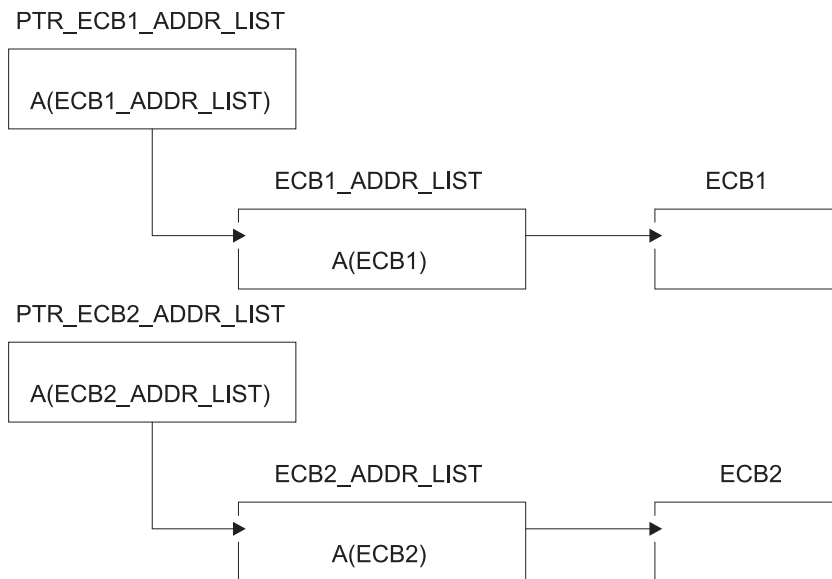


図 90. WAITCICS 例によって使用される ECB ポインター

217 ページの『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。

CICS 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を使用すると、CICS コードおよび制御ブロックがアプリケーション・プログラムによって誤って上書きされないようにすることができます。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機能は提供されていません。CICS では、アプリケーションが CICS ストレージを変更するのに必要なアクセス (実行キー) を取得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。アプリケーション・プログラムで誤って他のトランザクションのトランザクション・データを上書きすると、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整合性に影響する可能性があります。

ストレージ保護を使用するかどうかは任意です。CICS システム初期設定パラメーターを使用して、ストレージ保護機構を使用するかどうかを指定してください。関連するパラメーターについては、ストレージ保護を参照してください。

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

制御サービスにアクセスする Java プログラムについては、JCICS を使用した Java の開発および JCICS Javadoc の資料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 ファウンデーション・クラスの使用法を参照してください。

CICS ストレージ制御機能

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。103 ページの『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

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については CICS Interdependency Analyzer for z/OS の概要を、トランザクションの類縁性については 175 ページの『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。

- 358 ページの『CICS ストレージ保護およびトランザクション分離』
- 363 ページの『アプリケーション用のストレージ・キーの定義』
- 366 ページの『実行およびストレージ・キーの選択』
- 371 ページの『トランザクション分離による保護』

- 372 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 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、図 91 に示されています。

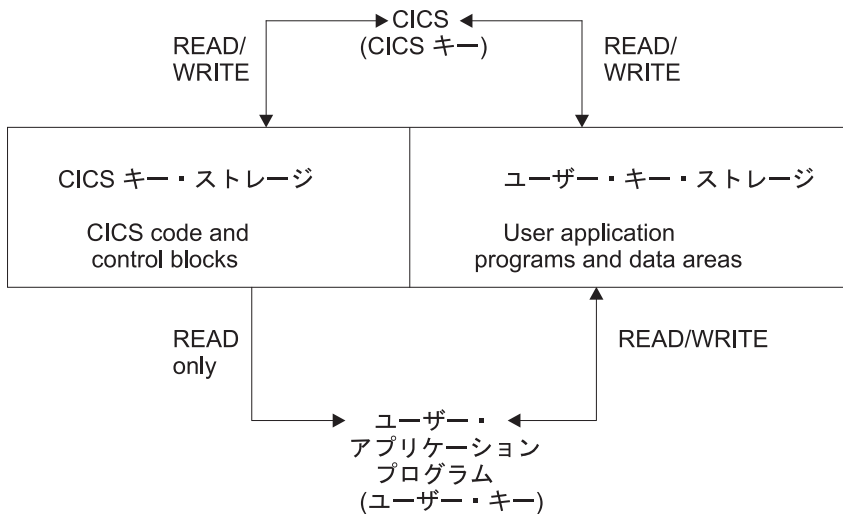


図 91. CICS コードおよび制御ブロックのユーザー・アプリケーション・プログラムからの保護

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 キー・ストレージおよびユーザー・キー・ストレージという、アプリケーション・プログラムを実行できる 2 つのタイプのストレージがあります。

CICS キー・ストレージ

ほとんどの 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

ほとんどのアプリケーション・プログラムおよびそれらのデータ域で使用されます。

以下のように 2 つの関連した実行モードがあります。

1. 多くの 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/書き込みアクセス権を持ちます。
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/書き込みアクセス権を持ちますが、CICS キー・ストレージに対しては読み取りアクセス権だけを持ちます。

『ユーザー・キー』および『CICS キー』という用語は、このように、ストレージのカテゴリーとプログラムの実行の両方に適用されます。これらの用語は、TRANSACTION 定義で使用されるリソース定義キーワードにも反映されます。詳細については、TRANSACTION 属性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、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、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、368 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、CICS キーでの実行をユーザーが選択するプログラムによって上書きされないように、CICS キー・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) を使用します。EXECKEY の説明については、366 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ため、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、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

アプリケーションで利用できる多数の CICS データ域およびアプリケーション・プログラム・データ域について、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。

このタスクについて

以下のいずれかを使用して、データ域に応じてストレージ・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 コマンドのオプション

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。

これらの領域がユーザー・キー・ストレージ内にある場合は、すべてのプログラムがこれらの領域に読み取り/書き込みアクセスすることができます。これらの領域が CICS キー・ストレージ内にある場合、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のみが可能です。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。

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ができます。

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指定するには、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ます。このオプションにより、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
- 以下のストレージ要求の応答としてアプリケーション・プログラムのために取得される任意のストレージ
 - GETMAIN コマンドの結果としての明示的な要求
 - 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要求

TASKDATAKEY パラメーターの指定方法については、TRANSACTION 属性を参照してください。

タスク存続期間ストレージとプログラム作業用ストレージの両方に関して TASKDATAKEY によってどのようなことが制御されるかが、365 ページの図 92 に示されています。

EXEC CICS コマンドのプログラミング情報については、CICS コマンド・サマリー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、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。

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 コマンドおよび GETMAIN64 コマンドは、関連するトランザクション・リソース定義で指定されている 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 オプションおよび CICS DATAKEY オプションを提供します。

例えば、これらのオプションを使用すると、TASKDATAKEY(CICS) を指定して実行されているアプリケーション・プログラムが、ユーザー・キーで実行しているプログラムに渡すため、あるいは戻すための、ユーザー・キー・ストレージを取得することができます。

CICS キー・ストレージを取得するために GETMAIN コマンドを発行する EXEC CICS を指定して定義されているプログラムでは、FREEMAIN コマンドまたは FREEMAIN64 コマンドが、EXEC CICS を指定して定義されているプログラムによって発行される場合にのみ、CICS キー・ストレージが解放されます。EXEC CICS を指定して定義されているアプリケーション・プログラムで 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、

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。

CICS は、タスクの終了時に、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期間ストレージを解放します (それらのストレージが、CICS キーとユーザー・キーのどちらであるかは関係ありません)。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。そのようにすると、誤って機密データが他のタスクで表示されないようにするため、ストレージがクリアされます。

コマンドのプログラミング情報については、CICS API コマンドを参照してください。リソースの定義については、CICS リソースを参照してください。

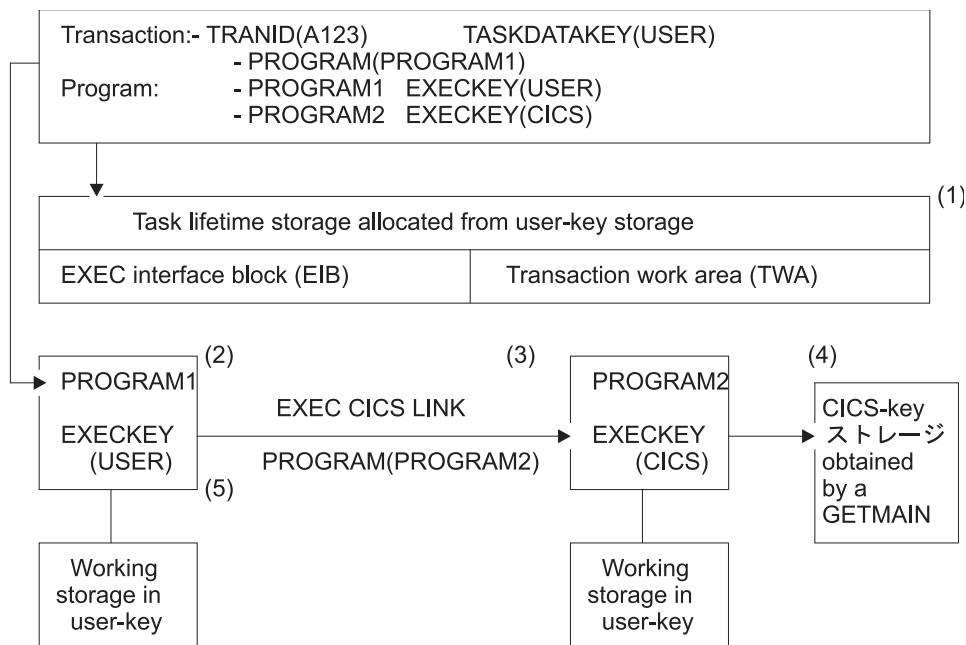


図 92. TASKDATAKEY および EXECKEY オプションの使用の図

この例の場合、トランザクション A123 は、TASKDATAKEY(USER) および PROGRAM(PROGRAM1) を指定して定義されています。PROGRAM1 は EXECKEY(USER) を指定して定義され、PROGRAM2 は EXECKEY(CICS) を指定して定義されます。PROGRAM1 は PROGRAM2 にリンクし、PROGRAM2 は GETMAIN 要求を使用して CICS キー・ストレージを取得します。

注:

1. TASKDATAKEY オプションは、EXECKEY(USER) で指定されるユーザー・キーで実行される PROGRAM1 で必要とされるとおりに、トランザクション作業域 (TWA) および EXEC インターフェース・ブロック (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

コマンドを使用して、または CICS コマンドの SET オプションを使用してプログラムが取得するその他のストレージも、ユーザー・キー・ストレージで取得されます。

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、CICS DATAKEY 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

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。

このタスクについて

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 38 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 38. 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。
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で制限されています。以下のものが含まれます。

- 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:

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。

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。これはほとんど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。

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。ここで説明している機能がアプリケーション・プログラムで正しく使用されることが確認できている場合にのみ、プログラムの定義を EXECKEY(CICS) に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・オプションには、オペレーティング・システム機能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、653 ページの『第 8 章 COBOL アプリケーションの開発』、593 ページの『第 7 章 C および C++ アプリケーションの開発』、および 693 ページの『第 10 章 PL/I アプリケーションの開発』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る、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、114 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る、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、118 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が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
- カスタム EP アダプター処理同期イベント

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よ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 (372 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、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 の概要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、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とき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、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による保護

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。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。

TRANISO システム初期設定パラメーターを使用して、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。

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できます。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。

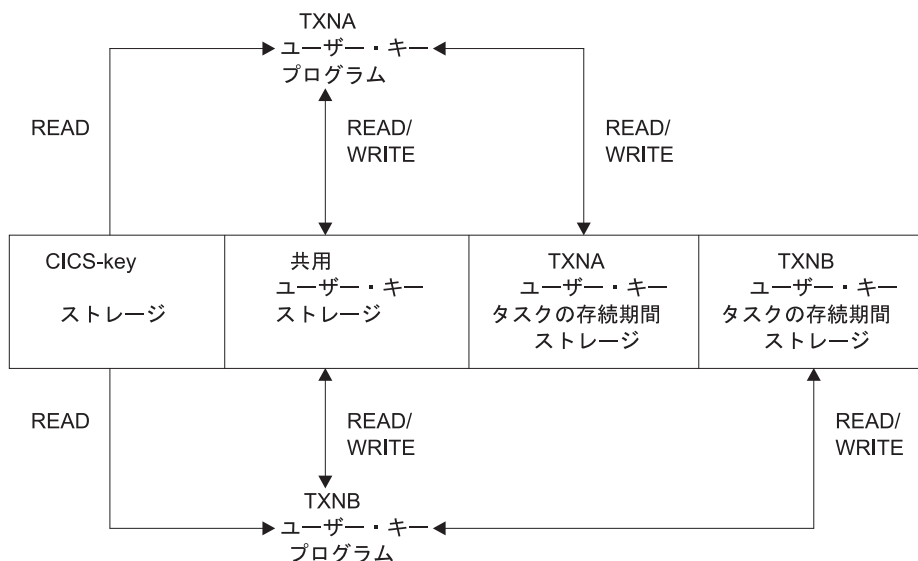


図 93. ISOLATE(YES) として定義された 2 つのトランザクション

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。

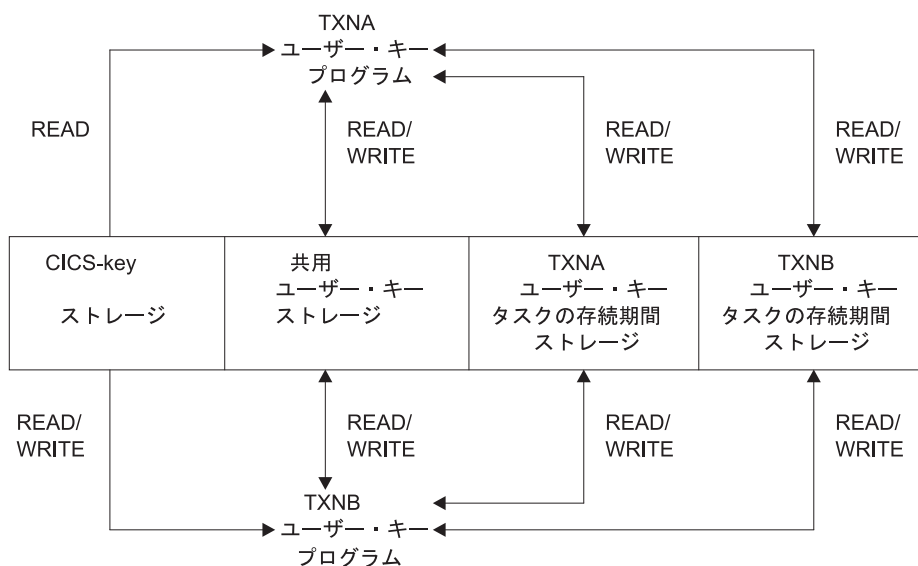


図 94. お互いのタスク存続時間ストレージに読み取り / 書き込みアクセスのある ISOLATE(NO) として定義された 2 つのトランザクション

ユーザー・キー・ストレージは、ISOLATE(YES) を設定した場合でも、CICS キー・プログラムから保護されません。

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。

しかし、少数のアプリケーションは、以下のような場合に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
- CICS 制御ブロックを変更する場合
- 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。

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 (一時記憶域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。

CICS の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 5 リリース 5 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について詳しくは、CICS Interdependency Analyzer for z/OS の概要 を参照してください。

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、後続の内部または外部処理のために、キューに入れる (格納する) 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・キューの内容を削除する (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、379 ページの『一時記憶管理』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は、JCICS を使用した Java の開発および JCICS の Javadoc 資料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 ファウンデーション・クラスの使用法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、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。

区画内データの一般的な使用法は、以下のとおりです。

- 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、TDQUEUE リソース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・セットの定義について詳しくは、CICS へのデータ・セットの定義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、TDQUEUE リソース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ラされています。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。トランザクションの類縁性について詳しくは、175 ページの『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。

一時データ・トリガー・レベル・トランザクションのために開始されたトランザクションは、直前のホップ・データを作成しないため、起点として実行しているものとして処理されます。前のホップ・データについて詳しくは、直前のホップのデータの特性を参照してください。

一時記憶管理

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域キュー、直接アクセス記憶装置の補助記憶装置の一時記憶域キュー、または一時データ共用プールの一時記憶域キューに、データを保管することができます。一時記憶域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。

さまざまな一時記憶域の場所と各場所の一時記憶域キューで利用可能な機能の概要については、主一時記憶域: モニターおよび調整を参照してください。

CICS 一時記憶域管理機能には、次の目的のためのコマンドが含まれます。これらの各コマンドで、TS キーワードは省略できます。これを指定しなければ、一時記憶域が想定されます。

- 一時記憶域キューにデータを書き込む (WRITEQ TS コマンド)。
- 一時記憶域キューのデータを更新する (WRITEQ TS REWRITE コマンド)。
- 一時記憶域キューからデータを読み取る (READQ TS コマンド)。
- 一時記憶域キューから次のデータを読み取る (READQ TS NEXT コマンド)。
- 一時記憶域キューを削除する (DELETEQ TS コマンド)。

プログラマーがどのようにして一時記憶域キューを使用できるかについて詳しくは、95 ページの『一時記憶域キュー』を参照してください。

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、217 ページの『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について詳しくは CICS Interdependency Analyzer for z/OS の概要 を参照してください。また、トランザクションの類縁性について詳しくは、175 ページの『類縁性』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、JCICS を使用した Java の開発および 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 ファウンデーション・クラスの使用法を参照してください。

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域キューは、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域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。

一時記憶域キューは、以下のように使用されます。

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尋ねます。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めます。タスクは、1 つ以上の単位の入力を読み取り、その後なんらか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。このデータを受信して、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。

CICS の文書および文書テンプレート

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、または別の 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。

Java アプリケーションでは、CICS Java クラス・ライブラリー (JCICS) を使用して文書サービスにアクセスできます。 **Document** クラスは、EXEC CICS DOCUMENT コマンドの Java 実装を提供します。クラスの資料については、JCICS クラス解説書にある Javadoc を参照してください。Java アプリケーションは、他のプログラミング言語で作成されたアプリケーションによって作成された文書を検索し、JCICS クラスを使用して文書进行处理します。

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、終了前に文書を保管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 (例えば一時記憶域キュー) に保管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
- z/OS UNIX システム・サービス・ファイル
- 一時記憶域キュー
- 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・インターフェースの記号および記号リストのサポートは、HTML 4 仕様のトピック フォームのコンテンツ・タイプで説明されているように、**application/x-www-form-urlencoded** のコンテンツ・タイプでデータを解釈す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 でエンコードされるデータの形式は、アンパーサンドで区切られた名前と値のペアのシーケンスです。例えば、次のようになります。

```
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、スペースは非常に一般的であるため、URL でエンコードされるデータ・タイプでは、エスケープ・シーケンスとしてではなく、正符号 (+) としてスペースを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、シングルのスペース、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、代わりに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
- 一時記憶域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。


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。

関連情報:

 データ・セットの暗号化

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。

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。このアーカイブ方法については、CICS 領域に対する z/OS UNIX ディレクトリーおよびファイルへのアクセス権限の付与で説明しています。

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 バッファの終わりに追加されます。

一時記憶

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 ロード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。

手順

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は、『DFHDHTL - プログラム・テンプレート prolog マクロおよび epilog マクロ』で説明しています。

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、

```

graph LR
    S1[DFHDHDL—TYPE=INITIAL—, ENTRY=entryname] --> S2[PROLOG=macroname]
    S2 --> S3[LOCTR=label]
    S3 --> S4[DFHDHDL—TYPE=FINAL—]
    S4 --> S5[CRLF=]
    S5 -- NO --> S6[ ]
    S5 -- YES --> S6
    S6 --> E[ ]
  
```

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、CICS 提供のコピーブックによりマップされます。

提供されるコピーブックは以下のとおりです。

- 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送ら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。

手順

- 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。以下に例を示します。

```
<!--#command-->
```

サーバー・サイド・インクルードの開始および終了で利用できる文字は、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が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。

以下に例を示します。

<!--#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。

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です。

このタスクについて

データを含んだ文書を作成するために、EXEC CICS DOCUMENT CREATE コマンドに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)。

DOCUMENT CREATE コマンドには、16 バイトのデータ域を必要とする DOCTOKEN という必須オプションがあります。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。

以下の例では、アプリケーションが EXEC CICS DOCUMENT コマンドを使用して文書を作成することができるさまざまな方法を示します。Java アプリケーションでは、CICS Java クラス・ライブラリー (JCICS) を使用して文書サービスにアクセスできます。**Document** クラスは、EXEC CICS DOCUMENT コマンドの Java 実装を提供します。クラスの資料については、JCICS クラス解説書にある Javadoc を参照してください。

手順

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、空の文書を作成し、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。

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。
- 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。
 - 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。以下に例を示します。

```
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。

- 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。以下に例を示します。

```
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。以下に例を示します。

```
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、文書に既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、400 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。

手順

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。400 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。
382 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、以下の点に注意します。
 - 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。

- ノル (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 39 に示します。

表 39. 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 仕様が拡張されます。

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、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既に定義されており、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。

手順

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。以下に例を示します。

```
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。以下に例を示します。

```
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。398 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を挿入します。データのバッファには、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(MYDOCBBOOK)
TEXT('Pre-bookmark text. ')
LENGTH(19)
```

```
EXEC CICS DOCUMENT INSERT
DOCTOKEN(MYDOCBBOOK)
BOOKMARK('ABookmark ')
```

```
EXEC CICS DOCUMENT INSERT
DOCTOKEN(MYDOCBBOOK)
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(MYDOCBBOOK)
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、置換で利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。

手順

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。

このタスクについて

以下の一連のコマンドでは、アプリケーションが EXEC CICS DOCUMENT コマンドを使用して、文書の作成、その取り出し、一時ストレージ・キューへの保管、および同一または別のアプリケーションの文書としての再利用を行う方法を示します。

Java アプリケーションは、他のプログラミング言語で作成されたアプリケーションによって作成された文書を検索し、JCICS クラスを使用して文書进行处理します。クラスの資料については、JCICS Javadoc 情報にある Javadoc を参照してください。

手順

1. アプリケーション・プログラムでは、以下の変数を定義して初期化します。
 - 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 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
2. 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・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) を格納するために必要なバッファの最大サイズです。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、バッファを割り振る前に実際の文書の長さを判別できます。

3. 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。
4. 一時記憶域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)
```
5. 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)
```
6. 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

また、DOCUMENT RETRIEVE コマンドと DOCUMENT INSERT コマンドを使  
用して、文書全体を既存文書に挿入することもできます。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
- 検索データ長を保持するための、RETRIEVLN というフルワード・バイナリ  
ー・フィールド
- バッファ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(RETRIEVLN)
 MAXLENGTH(MAXLEN)
```

```
EXEC CICS DOCUMENT INSERT
 DOCTOKEN(ITOKEN)
 FROM(DOCBUF)
 LENGTH(RETRIEVLN)
```

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 RETRIEVLN 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、以下のようになります。

1. 削除する文書の DOCTOKEN を指定します。 以下に例を示します。

```
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取得の制限について詳しくは、WEB RETRIEVE を参照してください。

## 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用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

このセクションでは、以下について説明します。

- 『名前付きカウンター・フィールド』
- 411 ページの『名前付きカウンター・プール』
- 413 ページの『名前付きカウンター EXEC インターフェースの使用』
- 414 ページの『名前付きカウンター呼び出しインターフェースの使用』
- 429 ページの『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

それぞれの名前付きカウンターは、カウンター名、現行値、最小値、および最大値から構成されます。

### 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 (413 ページの『名前付きカウンタ EXEC インターフェースの使用』を参照)、名前付きカウンタを定義し、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・プール

名前付きカウンタは、名前付きカウンタ・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 256 KB で定義されていても、最大で 1,000 の名前付きカウンタを保持することができます。

名前付きカウンタ・プールは、プール用にカップリング・ファシリティ・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・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、(DFHNCSTST や DFHNCDEV のような名前の) 別のプールを作成することが可能です。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、『名前付きカウンタ・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・サーバーのリスト構造の定義、および名前付きカウンタ・サーバーの開始については、Setting up and running a named counter serverを参照してください。

名前付きカウンタ・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。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・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。

EXEC CICS COUNTER および DCOUNTER コマンドについて詳しくは、413 ページの『名前付きカウンタ EXEC インターフェースの使用』を参照してください。

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 コマンドで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。Setting up shared data sets, CSD and SYSINでは、このすべてが取り上げられています。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*) オプションとともに動作します。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。

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。

## 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。

EXEC CICS コマンドを使用して、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。
- 現行番号 (100 025) が 50 ずつ更新されるので、次の要求のために用意される現行番号は 100 075 です。

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

プシオンを指定します。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。

- 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。

### 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、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:

名前付きカウンター呼び出し可能インターフェースを使用するには、アプリケーション・プログラムで以下のことを考慮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・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。コピーブック名は、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**  
または  
**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。
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、CICSTS55.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、コピーブック 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。以下に例を示します。

```
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,CTRLLEN,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。

CICS がサポートする初期バージョンの COBOL では、名前にアンダースコアの使用は許可されていません。このため、コピーブック DFHNCCOB に提供されているシンボル名は、下線の代わりにハイフンを使用します (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:

この図は、名前付きカウンター DFHNCTR の呼び出しの構文を示します。

:

```
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 95. 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を確認します。 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、424 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 要求に *counter\_options* パラメーターを使用すると、NC\_CREATE 要求で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。詳しくは、*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。以下に例を示します。

現行値が 109 で、  
さらに  
*update\_value*

が 25 を指している場合

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  $((\text{maximum\_value} + 1) - \text{minimum\_value})$  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 オプションに関する情報については、*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。

*update\_value* パラメーターに新規の値を指定します。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。これらについて詳しくは、425 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 \$ @ # \_ 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、411 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。

この名前に使用できる文字は、A-Z 0-9 \$ @ # \_ 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、424 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* によって指定されているよりも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 条件のままになります。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、または指定された値と等しい名前付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                  |
|-----------------|----------------|--------------------------|
| <i>function</i> | 01 NC-FUNCTION | PIC S9(8) COMP VALUE +1. |

| 呼び出しパラメーター             | COBOL 変数            | フィールド定義                  |
|------------------------|---------------------|--------------------------|
| <i>return_code</i>     | 01 NC-RETURN-CODE.  | PIC S9(8) COMP VALUE +0. |
| <i>pool_selector</i>   | 01 NC-POOL-SELECTOR | PIC X(8).                |
| <i>counter_name</i>    | 01 NC-COUNTER-NAME  | PIC X(16).               |
| <i>value_length</i>    | 01 NC-VALUE-LENGTH  | PIC S9(8) COMP VALUE +4. |
| <i>current_value</i>   | 01 NC-CURRENT-VALUE | PIC S9(8) VALUE +0.      |
| <i>minimum_value</i>   | 01 NC-MIN-VALUE     | PIC S9(8) VALUE +0.      |
| <i>maximum_value</i>   | 01 NC-MAX-VALUE     | PIC S9(8) VALUE -1.      |
| <i>counter_options</i> | 01 NC-OPTIONS       | PIC S9(8) COMP VALUE +0. |
| <i>update_value</i>    | 01 NC-UPDATE-VALUE  | PIC S9(8) VALUE +1.      |
| <i>compare_min</i>     | 01 NC-COMP-MIN      | PIC S9(8) VALUE +0.      |
| <i>compare_max</i>     | 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.
```

以下に、前述の WORKING-STORAGE SECTION で指定されるデータ名と、LINKAGE SECTION に記述されている NULL-PTR 名を使用して、名前付きカウンター・サーバーを呼び出す場合の例を示します。ここでは、オプション・パラメーターとして、*value\_length*、*current\_value*、*minimum\_value*、および *counter\_options* のみが指定されるものとします。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 NC-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 NC-FUNCTION NC-RETURN-CODE NC-POOL-SELECTOR
NC-COUNTER-NAME NC-VALUE-LENGTH NC-CURRENT-VALUE
NC-MIN-VALUE NULL-PTR NC-OPTIONS.
```

DFHNCTR で COUNTER の代わりに DCOUNTER を使用する場合は、NC-CURRENT-VALUE 定義、NC-MIN-VALUE 定義、および NC-MAX-VALUE 定義を符号付き 10 進数値 (前述のリスト参照) からダブルワード符号なしバイナリ一定義に変更する必要があります (例: PIC 9(16) COMP)。さらに、MOVE 8 TO NC-VALUE-LENGTH も必要です。

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 まで) があり、これらは、(本来は正常に完了する) 要求が結果オーバーフローになったことを示します。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る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、指定された名前で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、アプリケーション・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)

インターフェースは、適切な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、アプリケーション・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、アプリケーション・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 IEANTxx 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行します。その結果、アプリケーション・プログラムでは時間的遅延のみが生じ、エラー応答を得ることはありません。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。

### 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使用して、他の情報から再構成した適切な値を使用して再作成します。
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

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異常なりカバリー状態でのみ実行されるため、検証とテストに注意を払う必要があります。

既存のデータ・リポジトリ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 以上を実行しているシステムの場合、システム管理二重化を使用して、名前付きカウンターの重複コピーを異なるカップリング・ファシリティーで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。

## 印刷とスプール・ファイル

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。

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、比較的短い文書のオンデマンド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。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。

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調整の問題については、440 ページの『CICS プリンターの使用』で説明します。

460 ページの『BMS サポート・レベル』は、BMS がサポートする装置をリストしています。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、332 ページの『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

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 コマンドはチャン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、336 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、438 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 342 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。完全なリストについては、IBM 3270 Data Stream Programmers Referenceを参照してください。

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。 345 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 345 ページの図 88 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、435 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 345 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 40. 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    | 注                               |
|------|-------|---------------------------------|
| 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。  |

表 40. 印刷制御オーダーを使用するデータ・ストリームの例 (続き)

| バイト   | 内容                                       | 注                                             |
|-------|------------------------------------------|-----------------------------------------------|
| 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<br>_____ 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、3270 プリンターに適用される個別制御機構は、コマンド・オプション PRINT、ERASE、L40、L64、L80、HONEYCOMB、NLEOM、FORMFEED、および PRINTERCOMP の形式をとります。

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「IBM 3270 Data Stream Programmers 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、装置バッファが更新されます。表示の際に、結果はバッファ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。336 ページの『出力データ・ストリーム』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を構築し、既にバッファ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。ただし、NLEOM オプションはこの機能に影響を与えます。『NLEOM オプション』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・バッファと同様、3270 プリンター・バッファ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・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。

ただし、行の長さが 40、64、または 80 文字 (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 の下で L40、L64、または L80 を使用する場合には、端末定義のページ幅に対応する値のみを使用します (439 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・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があります。

#### 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、439 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

バッファ制御オーダーを使用している場合には、バッファ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・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。

使用可能な線幅が 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・バッファを持っていないので、バッファ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、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、「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、348 ページの『3270 端末からの入力』で説明されているアテンション・キーと類似していません。代わりに、「APAK *nm*」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、*nm* 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。

## CICS におけるプリンターの使用

以下の情報を使用して、CICS プリンターおよび非 CICS プリンターに対してプログラムする方法を示すことができます。

### 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。この目的には、ASSIGN および INQUIRE TERMINAL コマンドを使用できます。

323 ページの表 33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 の各オプションが示されています。

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 41 に示されています。

表 41. 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 (有効な値については、Introduction to System programming commandsを参照) |

表 41. プリンター用の INQUIRE TERMINAL オプション (続き)

| INQUIRE オプション | TERMINAL または TYPETERM 定義の中のソース | 説明                        |
|---------------|--------------------------------|---------------------------|
| 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。次の表に、使用される値をリストします。

表 42. 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 SCRNHHT 値および SCRNWWD 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。

## CICS プリンターの使用:

START コマンドを使用した印刷、一時データを使用した印刷、または BMS ルーティングを使用した印刷を使用して、CICS プリンターに印刷できます。

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、使用するプリンターを所有していない場合には、次のいずれかの方法を使用し、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。

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:

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 **TERMINID** 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。

このタスクについて

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。以下に例を示します。

```
(build output in OUTAREA, formatted as expected by the
STARTed task)
EXEC CICS START TRANSID(PRNT) FROM(OUTAREA) TERMINID(PRT1)
LENGTH(OUTLNG) END-EXEC.
```

図 96. 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 97. 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:

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。

このタスクについて

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。サンプルでは、このプログラムについて詳しく説明しています。

印刷するタスク (プリンター *PRT1*)::

この例は、プリンター PRT1 に対する事前定義された記号宛先である、「PRT1」という一時データ・キューにデータを書き込むための構文を示します。

```
(必要なフォーマットや他の任意の処理を行います)
EXEC CICS WRITEQ TD QUEUE('PRT1') FROM(OUTAREA)
LENGTH(OUTLNG) END-EXEC.
```

トリガーされるタスク::

この例は、印刷タスクが、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する方法を示します。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。

```
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)
```

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、 377 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、タスクと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 がインターリーブを防ぎます。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・キューについての説明は、 375 ページの『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:

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。

このタスクについて

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、 533 ページの『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:

ここにリストするステップは、CICS の外部で管理されるプリンターについての、出力の形式設定、出力の送達、および出力の印刷の準備ができたときのアプリケーションへの通知方法を示します。

#### 手順

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では、**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。

**BMS** を使用した出力の形式設定については、『非 **CICS** プリンターに関するプログラミング』で説明します。プリンターを駆動するアプリケーションの形式設定の要件は、装置形式、またはアプリケーションが指示した中間形式となります。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、これは表 43 の 1 列目にリストされているファイル・タイプの 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 43. 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   | <b>CICS</b> スプール・コマンド ( <b>SPOOL</b> OPEN、 <b>SPOOL</b> WRITE など)、一時データ・コマンド ( <b>WRITEQ</b> TD)、端末管理および <b>BMS</b> コマンド ( <b>SEND</b> 、 <b>SEND</b> MAP など) |
| <b>BSAM</b> | <b>CICS</b> スプール・コマンド ( <b>SPOOL</b> OPEN、 <b>SPOOL</b> WRITE など)一時データ・コマンド ( <b>WRITEQ</b> TD)                                                              |
| <b>VSAM</b> | <b>CICS</b> ファイル制御コマンド ( <b>WRITE</b> )                                                                                                                      |
| <b>Db2</b>  | <b>EXEC SQL</b> コマンド                                                                                                                                         |
| <b>IMS</b>  | <b>EXEC DLI</b> コマンドまたは <b>CALL DLI</b> 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 (またはそれがいないこと) によって、使用するコマンドがどれかが決まります。

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、 375 ページの『一時データ管理』を参照してください)。
- 順次端末の出力側 ( 330 ページの『順次端末サポートの使用』および 479 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 ( 449 ページの『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 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用されます。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、442 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ピ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ピングも防ぎますが、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ます、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する必要があります。) 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。限界は大きいのが普通ですが、長時間 (CICS で可能な限り) 実行され、非常に多く (CICS の下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。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ます、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ます。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、444 ページの表 43 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。詳しくは、TERMINAL リソース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。このオプションにより、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。DISPLAY and PRINT options for combined lists of information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 z/OS Communications Server 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 z/OS Communications Server 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。

**ISSUE COPY** コマンドを使用して、コピーされる端末を所有するタスクに対応するものとして、プリンターを所有するタスクで画面をコピーすることができます。これは **TERMINAL** オプションに指定されている端末のバッファを、タスクを発行したプリンシパル装置のバッファにコピーします。コピーが発生した際のコピーの方法および印刷の開始は、**ISSUE COPY** コマンドの **CTLCHAR** オプションで定義される「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 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 については、CSPG - ページ検索を参照してください。

## JES に対する CICS インターフェース

CICS には、JES (MVS のジョブ入力サブシステム・コンポーネント) とのプログラミング・インターフェースが用意されています。これを使用すると、CICS アプリケーションでスプール・ファイルを作成および取得できるようになります。

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ス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ス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 (451 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。450 ページの図 98 を参照してください。

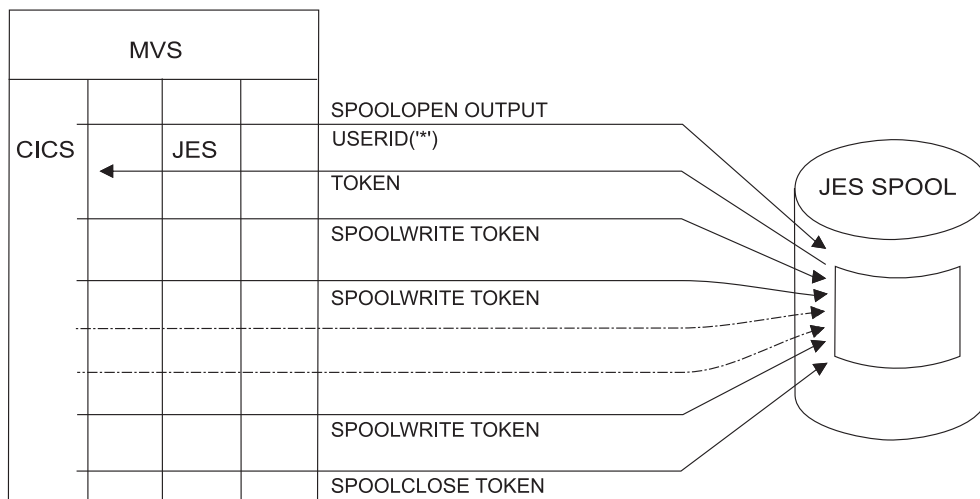


図 98. ファイルを作成して JES スプールに書き込む

- 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 99 を参照してください。

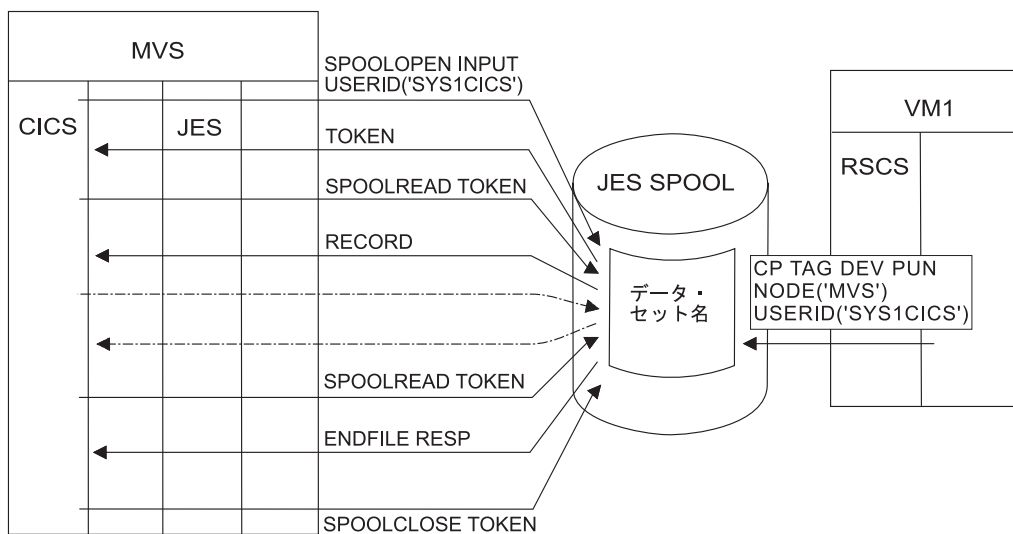


図 99. JES スプールからのデータの取得

- 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。451 ページの図 100を参照してください。

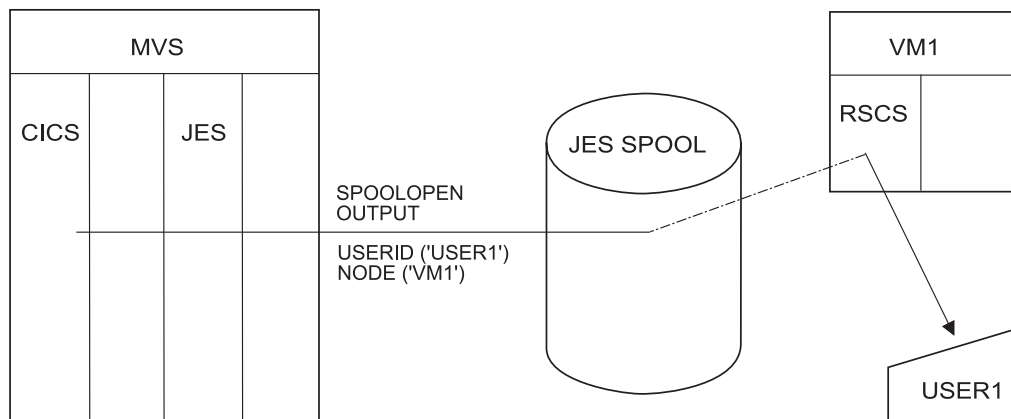


図 100. リモート宛先への書き込み済みファイルの送信

### JES に対する CICS インターフェースの使用:

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。

このタスクについて

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、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、キュー・エレメント、ディスク・スペース) を必要とします。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:

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。SPOOLWRITE コマンドを使用してレコードがファイルに追加され、その後、送達または処理のためにファイルを JES にリリースするために SPOOLCLOSE コマンドが使用されます。

このタスクについて

JES は、SPOOLOPEN OUTPUT 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用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。

出力スプール・ファイルの MVS 内部読み取りプログラムへの書き込み:

出力を MVS 内部読み取りプログラムに書き込むには、USERID(『INTRDR』) を SPOOLOPEN OUTPUT コマンドに指定して、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 予約名です。

このタスクについて

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 コマンドで NOCC オプションを指定してください。

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に入れ、このバッファ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、455 ページの『スプール・ファイルの識別』を参照してください。

内部読み取りプログラムに書き込む別の方法としては、外部 TDQ を使用します。これを行うには、TDQ で次のような DD カードを参照する必要があります。

```
//ddname DD SYSOUT=(A,INTRDR)
```

CICS は、代理ユーザー ID 検査を実行して、このユーザーがジョブ・カードに指定されたユーザー ID でジョブを実行依頼することを許可されているかどうかを検査するように構成できます。JOB ステートメントに USER オプションがない場合は、ジョブを実行するユーザー ID を構成できます。セキュリティ・オプション、およびその構成方法について詳しくは、内部読み取りプログラムに JCL ジョブを実行依頼する場合のセキュリティを参照してください。

入力スプール・ファイルの読み取り:

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。

このタスクについて

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じ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で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 450 ページの図 99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います。

アプリケーション間でファイルを転送する場合、唯一の命名要件は、CICS タスクでスプール・ファイルを読み取るときと同様、送信側が使用した名前が受信側 (CICS タスク) で認識され、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でも同じ名前が他の目的に使用されていないことです。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、ユーザーの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 z/OS Communications Server のアプリケーション 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手に入れることとなります。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 z/OS Communications Server 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 の場合は特定のユーザーを指します。MVS の場合は、JES 外部書き出しプログラムまたはその他の JES 宛先、TSO ユーザー、あるいはそのシステムで実行されている別のジョブを指します。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 を通じ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 101. 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 要求を使用して取得された領域のアドレスが、L-ADDRESS である  
L-FILLER が指す最初の語に移動します (つまり、自らを指すようになります)。  
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。したがって、W-POINTER は、  
OUTDESCR データを指すポインターを含む領域を指すようになります。

**SPOOL コマンドの例:**

次の例は、COBOL、PL/I、C、および ASSEMBLER を使用した SPOOL コマンドを示しています。

**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)

```

## 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、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。 特定の端末について *device-independent* 出力コマンドを解釈し、*device-dependent* 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。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。

## 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さい。

ここでは機能について要約し、また最小 BMS がない機能の説明がある個所では、これらについての注記があります。

### 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「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)

- TWX 33/35 型
- 1050
- 2740-1 (受信バッファなし)、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)

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。

## BMS 3270 侵入検知サービス

この機能により、CICS は、BMS マップによって生成された保護フィールドを 3270 エミュレーターが不正に変更した場合に検出できます。機能切り替えによってこの機能を選択できます。機能切り替えの指定を参照してください。

保護フィールドの変更によって、アプリケーションのセキュリティが損なわれる場合があります。この機能は、IBM Communications Server が提供する 3270 侵入検知サービスと連動します。構成されている場合は、IBM Communication Server がすべての 3270 アプリケーションの保護を行います。

両方のサービスが使用可能である場合、BMS で生成された 3270 データは CICS で処理され、BMS 以外の 3270 データは IBM Communications Server で処理されます。この両方を使用可能にする利点は、すべての 3270 アプリケーションを完全に対象とし、BMS を使用してパフォーマンスを最大化することにより、何らかの侵入について返される情報を向上させることです。

3270 IDS の構成および使用法については、「z/OS Communications Server SNA ネットワーク・インプリメンテーション・ガイド」の『3270 侵入検知サービス』を参照してください。

注: サービスを適用するアプリケーションやマップを特定するように BMS 3270 IDS を構成する必要がある場合は、URM DFHBMSX を使用して BMS 3270 IDS を構成することもできます。一般に、これは、アプリケーションで 3270 データ・ストリームが異常に使用され、正しくないヒットが報告された場合にのみ必要になります。

この機能を有効にするための機能切り替え

```
com.ibm.cics.bms.ids={true| false }
```

構成オプションを設定するための機能切り替え

```
com.ibm.cics.bms.ids.action={abend|ignore| log }
```

3270 エミュレーターで上書きされる保護フィールドの検出を CICS が処理する方法を指定します。値は以下のとおりです。

異常終了

CICS でトランザクション ABSX が異常終了されます。

ignore

CICS ではアクションは実行されません。

ログ CICS では、上書きの詳細とともに DFHTF0200 メッセージが出されます。これはデフォルトです。

この構成オプションにより、URM DFHBMSX に渡されるデフォルトが設定されます。より細かく CICS アクションを構成する場合は、構成に URM DFHBMSX を使用します。URM DFHBMSX は、この構成オプションをオーバーライドします。

```
com.ibm.cics.bms.ids.vtamignore={ true |false}
```

CICS で BMS 要求に関連する 3270 データを送信する際に、IBM Communications Server に対して、CICS がデータの検査を行うことを通知するかどうかを指定します。これにより、IBM Communications Server の侵入検知サービスに対して、要求を無視できることが通知されます。このオプションは、IBM サービス担当員の指示のもとでのみ使用してください。

## BMS の出力例

この例は、BMS による形式設定画面の作成方法を示します。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に表示します。

BMS は、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) とを組み合わせることによって、形式設定画面を作成します。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。431 ページの『印刷とスプール・ファイル』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 479 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。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変更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。より实际的で完全な 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。詳細については、サンプル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、顧客番号を取得し、応答で、図 102 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 102. 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入力し、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 103 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 103. 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。

最初に表示画面を定義する必要があります。この特定のマップを定義する方法については、465 ページの『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 464 ページの図 104 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 104. 「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、「クイック顧客取引チェック (Quick Customer Account Check)」や「取引 (Account)」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、488 ページの『表示特性の設定』および 500 ページの『マップ・データの受信』で説明しています。

この特定の例題のキー・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、画面に表示する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 104) にコピーする必要がある、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 ACCTNO0.
MOVE ACCTFILE-SURNAME TO SURNAME0.
MOVE ACCTFILE-FNAME TO FNAME0.
COMPUTE CHGO = ACCTFILE-CREDIT-LIM - ACCTFILE-UNPAID-BAL
- ACCTFILE-CUR-CHGS.
IF CHGO < ZERO, MOVE ZERO TO CHGO
MOVE 'OVER CHARGE LIMIT' TO MSG0.
IF ACCTFILE-WARNCODE = 'S', MOVE DFHMBRY TO MSG0
MOVE 'STOLEN CARD - SECURITY NOTIFIED' TO MSG0
EXEC CICS LINK PROGRAM('NTFYCOPS') END-EXEC.
EXEC CICS SEND MAP ('QCKMAP') MAPSET ('QCKSET') END-EXEC.
EXEC CICS RETURN END-EXEC.

```

図 105. BMS の出力例

## 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済みます。

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 Library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。

### DFHMDF

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、335 ページの『3270 ファミリーの端末』を参照してください。

#### マップ・フィールドの定義: DFHMDF:

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。

このタスクについて

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)
- 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 ( \_ ) 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、フィールド名にハイフン (-) 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、その他の文字は、前述の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、@ 文字や (初期のバージョンでは) アンダースコアは使用できません。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 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、BMS マクロ DFHMDf で説明しています。

図 106 は、463 ページの図 103 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 106. 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。詳しくは、494 ページの『出力画面の作成』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 339 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 488 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。464 ページの図 104 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、475 ページの『複合フィールドの使用』で、マップに関するいくつかの追加情報の後に説明があります。

#### マップの定義: DFHMDI:

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。

このタスクについて

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。
- DFHMDI 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) (481 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**:

マップ・セットを定義する DFHMSD マクロを使用して、マップを作成します。

このタスクについて

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、474 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 (473 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 464 ページの図 104 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、485 ページの『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であるため、アセンブラー言語の構文規則に従う必要があります。

以下の一連の規則が適用されますが、実際の規則はこれほど厳格ではありません。アセンブラー言語の構文規則の完全なセットについては、高水準アセンブラー言語の参照情報を参照してください。

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してはいけません。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、486 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。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 (471 ページ)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)。アンバ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、アンバーサンドを 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. マップにコメントを入れたい場合は、単一マクロを構成する行の間ではなく、マクロの間でコメント行を使用します。コメント行には、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 では通常の大文字を使用します。

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 (前述の例では同一の定数に対して異なる LENGTH 値が指定されていること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tringを終了する SI) の後でも停止できます。DBCS スtring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tring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。

```

GXML1 DFHMDF
POS=(02,21),LENGTH=20,PS=8,GINIT='<D1D2D3D4D5D6>*****
<D7D8D9D0>'
IXML1 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 (470 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:

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。

このタスクについて

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:

TYPE=MAP アセンブルとそれに続くリンク・エディットを使用して、物理マップ・セットと呼ばれるロード・モジュールを生成できます。TYPE=DSECT アセンブルは一連のデータ構造を生成するために使用され、ひとまとめにしてシンボリック・マップ・セットと呼ばれます。

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あり、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 (464 ページの図 104 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ル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・アセンブリーを伴う SYSPARM の説明については高水準アセンブラー言語の参照情報を参照し、マップのアセンブルについて詳しくは、1029 ページの『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。再配置では DFHMDF マクロを参照します。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。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:**

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、アセンブリーとリンク・エディットのステップは必要ありません。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 Library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・セットを最初を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、487 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 1.2 以上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、IBM 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 のバージョン 1.2 より前の各 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 44. 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 |
| PL/I   | DFHBRARL |
| COBOL  | DFHBRARO |

ADS 記述子について詳しくは、外部インターフェースに向けた開発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 が提供されています。

DFHBMSUP について詳しくは、BMS マクロ生成ユーティリティー (DFHBMSUP) を参照してください。

## 複合フィールドの使用:

BMS は、複合フィールドの定義のために、DFHMDF マクロに GRPNAME と OCCURS の 2 つのオプションを提供します。

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 ( 464 ページの図 104 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:

GRPNAME オプションは、記号ストレージ定義を生成し、いくつかのフィールドを 1 つのグループ名で結合するために使用されます。グループに属させるフィールドごとに、同じグループ名を指定してください。

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。

DFHMD F 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、DFHMD F 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。前述の日付を、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、次のように書きます。

```
MO DFHMD F POS=(10,1),LENGTH=2,ATTRB=BRT,GRPNAME=DATE
SEP1 DFHMD F POS=(10,3),LENGTH=1,GRPNAME=DATE,INITIAL='- '
DAY DFHMD F POS=(10,4),LENGTH=2,GRPNAME=DATE
SEP2 DFHMD F POS=(10,6),LENGTH=1,GRPNAME=DATE,INITIAL='- '
YR DFHMD F 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 DFHMD F 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:

OCCURS オプションは、示された数のフィールド項目をマップ上に生成することと、フィールドが行列または配列の項目としてアドレッシングできるような方法で、マップ定義を生成することを指定します。

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、および初期値と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 DFHMBRY to TELNOA (I).
```

(DFHMBRY 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。 490 ページの『属性値の定義: DFHBMSA』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。改訂されたシンボリック・マップは元のものと同一のフ  
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。

シンボリック出力マップには、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、463 ページの図 103 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) では下記の様  
になります。これを、同一のマップ定義からの (464 ページの図 104 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 107. 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:

非 3270 端末のサポートは、標準 BMS を必要とします。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。

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。全リストについては、IBM 3270 Data Stream Programmers Referenceを参照してください。

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 483 ページの表 45 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、438 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、出力操作よりも入力操作に多くの影響があります。これは、BMS の機能の多くが、変更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変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、それによってマップします。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、または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。

これらのオプションには、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。 316 ページの『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、 563 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。 435 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。

装置依存マップは、異なる特性の複数装置をサポートしなければ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 108. 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 109. 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、501 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 および PROFILE のリソース定義については、RDO resources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:

装置依存サポート (DDS) は、装置依存マップを使用可能にするインストール機能です。

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 45 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 45. 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)                            |
| 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                                                      |
| 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                                         |

表 45. BMS 用の端末コード (続き)

| コード  | 端末または論理装置                                       |
|------|-------------------------------------------------|
| 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)                     |

インストールで、前述のような小型画面などの追加の端末タイプを定義することもできます。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、323 ページの『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は最大になるので、後の 525 ページの『累積処理用の ASSIGN オプション』で説明します。

## BMS マップ出力の送信

BMS マップ出力を送信する前に、出力データをマップ構造に移動するための以下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

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。465 ページの『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。

### 手順

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

マップ用ストレージの獲得および定義:

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。

このタスクについて

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。95 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。以下に例を示します。

```
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 358 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。

465 ページの図 105 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、前述の例のように、言語機能を使用してこの長さを判別することができます。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:**

出力の作成を開始する前に、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間違えて使用されなくなります。

#### **このタスクについて**

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、503 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、512 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、464 ページの図 104 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 QCKMAP0.
```

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 (511 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。まず、データについて、次に、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 493 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 494 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。BMS は名前付きフィールドに対して、プログラム内でマップが割り当てられた値を指定変更するために使用できるサブフィールドを生成します。

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。(属性については 339 ページの『3270 フィールド属性』を参照してください。)

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、489 ページの表 46 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 46. 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           |
| 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はない限り、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 ( 316 ページの『データ伝送コマンドの使用』および IBM 3270 Data Stream Programmers Reference を参照)。

適用される属性のタイプは、実行時のプリンシパル装置の機能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、323 ページの『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。481 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:

マップ定義内のフィールド属性は、オプション MAPATTS および DSATTS を使用して変更できます。

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い場合はオフにすることを指示します。

変更する必要が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 DFHMDI
POS=(5,13),LENGTH=8,ATTRB=(ASKIP,NORM),PICOUT='$,$$0.00', X
COLOR=NEUTRAL
MSG DFHMDI 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、464 ページの図 104 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。494 ページの『出力画面の作成』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。CICS は、DFHBMSCA と呼ばれるソース・コードを提供します。このコードは、すべての属性に対して一般に使用される値を定義し、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。

DFHBMSCA をプログラムにコピーできます。 489 ページの『属性の変更』のプログラム DFHBLINK がその例です。 DFHBLINK を定義するには、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.
...
COPY DFHBMSCA.
```

それぞれのプログラミング言語について、いずれかのバージョンの DFHBMSCA が提供されています。値の名前は、すべてのバージョンで同じです。DFHBMSCA に含まれていない属性の組み合わせが必要な場合は、値を判別するため、「IBM 3270 Data Stream Programmers Reference」を参照してください。値を頻繁に使用する場合は、その値が含まれるように DFHBMSCA を変更することができます。

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。

## SEND MAP コマンドの使用

このリストは、SEND MAP コマンドが BMS に通知する内容を示します。

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 498 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。

### 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、端末にバッファがある場合には、出力をバッファに書き込む前に、装置バッファの内容を修正します。ERASE はバッファ全体をヌル (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・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、バッファのクリアとバッファ・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、336 ページの『出力データ・ストリーム』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。493 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』の『DATAONLY オプション』に説明があります。

FRSET (フィールド・リセット) はバッファにあるすべてのフィールドの変更データ・タグをオフにします。このオプションについては、339 ページの『3270 フィールド属性』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います。
- **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、434 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 z/OS Communications Server 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変更されません。

### 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。この入力为正しければ、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、491 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、511 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) 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) 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。336 ページの『出力データ・ストリーム』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、表示データ也没有ありません。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 データがバッファにロードされる前に適用されます。これらのオプションのどちらも SEND MAP で指定されてい

ない場合、画面バッファは、直前の書き込み操作後と同じ状態、つまり、オペレーターが実行した何らかの操作によって変更された状態で開始されます。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、バッファで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。『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) との間のバッファ位置は、変更されません (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。

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、プログラム、マップ、ハードウェアのデフォルト、または表示画面の直前の内容の 4 つのソースに由来します。

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、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 (489 ページの表 46 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の場合、値はヌル (X'00') または入力操作から残される値のいずれか (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

マップは、表示画面またはプリンターのページと同じサイズ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。515 ページの『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) 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ひでしょう。

送信されたマップの外側ではバッファ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

バッファ・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:

カーソルの位置決めは、入力にマップを使用するときに重要です。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

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ひ、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 506 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 500 ページの『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。 510 ページの『マップ入力処理』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 および SEND CONTROL コマンドで発生する可能性がある例外的な条件のほとんどは、高水準 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しようとす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は検査しません。

SEND MAP および SEND CONTROL コマンドで発生する可能性があるエラー条件について詳しくは、CICS API コマンドを参照してください。

無効データの影響は、以下のようなものですが、個々の端末によって、また、不正データの性質によって、異なります。

- 無効データが制御シーケンスとして解釈されてしまうことがあり、その結果、データが装置で受け入れられても誤った出力が生成される。
- 画面にエラー標識が表示されることがある。
- ATNI 異常終了が発生することがある。タスクが ATNI の通知を受けるタイミングは、WAIT 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 ( 491 ページの『SEND MAP の制御オプション』を参照)。

出力後処理オプション: **TERMINAL**、**SET**、および **PAGING**:

出力後処理オプションは、形式設定された出力ストリームで BMS が何をする必要があるかを指定します。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 **SET** と **PAGING** の 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。 486 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に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 533 ページの『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。543 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 47. ページ・リスト項目形式

| バイト          | 内容                         |
|--------------|----------------------------|
| 0            | 端末タイプ ( 483 ページの表 45 を参照)  |
| 1 から 3<br>まで | 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 48. 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                                    |
| 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。

## 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。

入出力の例:

この例は、「クイック・アップデート」と呼ばれる単純なデータ入力プログラムでのマップ定義を示します。

入力構造の詳細を説明する前に、462 ページの『BMS の出力例』の「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 110. 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 111. 「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。)

この例に必要なマップ定義は次のとおりです。

```

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 112. 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 の使用について詳しくは、BMS マクロ DFHMDFを参照してください。

図 113 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 MSG1 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 113. 「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。比較については、463 ページの図 102 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削除したかどうか、また、カーソルがフィールドに置かれていた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、463 ページの図 102 と 503 ページの図 113 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、誤った入力を入力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、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

「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 485 ページの『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、512 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。

コマンドは次のものを指定します。

- 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 ( 515 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、 507 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。

RECEIVE MAP コマンドを使用すると、画面上のデータだけでなく、オペレーターが最後にカーソルを使用した場所、および送信に使用したキーを確認することができます。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。EIBRID によって伝送キー (「アテンション ID」または AID) を識別することができ、EIBCURSR はカーソルが最後に置かれていた場所を示します。

## 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 GETMAIN コマンドを使用して、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じの選択であり、同じ方法で使います (これらの詳細および例については、 486 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かか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、受け取られる変数は、シンボリック入力マップによって定義される領域であり、BMS によって上記のような文字「I」を接尾部とするマップ名が割り当てられます。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 (358 ページの『ストレージ制御』を参照)。

形式設定画面の入力:

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。

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 キーまたはファンクション・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、どの方法が行われたかを知ることができます。508 ページの『アテンション ID の使用』に、それを知る方法についての説明があります。また 3270 入力操作の詳細は、348 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、513 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 キーおよびファンクション・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、556 ページの『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 キーまたはファンクション・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:


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「変更データ・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。

変更については、339 ページの『3270 フィールド属性』で説明します。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、509 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。

関連資料:

 BMS マクロ DFHMDF

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。

これらの仕様がどのように相互作用するかについては、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、端末入力が始ま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。69 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使用できません (515 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:

アテンション ID 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それが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 (490 ページの『属性値の定義: DFHBMSCA』を参照)。DFHAID コピーブックの内容は、アテンション ID 定数、DFHAID を参照してください。

#### HANDLE AID コマンドの使用:

HANDLE AID コマンドにより、使用されるアテンション・キーを特定することができます。

HANDLE AID を使用して、ディスプレイ装置からアテンション ID (AID) を受け取ったときにどのラベルに制御を渡すかを指定することができます。

制約事項: このコマンドは、COBOL、PL/I、およびアセンブラ言語アプリケーション (ただし、AMODE(64) アセンブラ言語アプリケーションを除く) でのみサポートされています。サポートされている他のすべての高水準言語では使用できません。

HANDLE AID は、他の HANDLE コマンドと同じように動作します。HANDLE AID を適用対象の最初の RECEIVE コマンドの前に発行すると、HANDLE AID で指定されたキーが使用される場合は、後続の RECEIVE の完了時にプログラムが分岐します。

以下に、「エスケープ」コードの代わりとなるコードの例を示します。

```
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らず、制御は HANDLE AID で指定されたラベルに移動します。そのため、HANDLE AID は、HANDLE CONDITION を使用して検査する場合は、例外条件をマスクできます。AID、例外条件、またはその両方については、代わりのテストを使用の方が望ましいものと思われます。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 を検査するといいいでしょう。そのような状況は、特に MAPFAIL 条件の場合に顕著です。513 ページの『MAPFAIL およびその他の例外状態』を参照してください。

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最後に置いた位置を知ることが必要になる場合があります。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、カーソルが最後に置かれたマップ・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、名前が割り当てられているマップ・フィールドにカーソルが置かれている場合にのみ有効になります。

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。値 X'80' または X'82' は消去のシグナル通知であり、X'02' または X'82' はカーソルを示します。BMS マクロの資料で取り上げられている 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、左上隅 (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:

この例は、「クイック・アップデート」プログラムを使用して、入力サブフィールドの使用方法を説明します。

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' ステートメントとそれに続くステートメントは、497 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きます。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 DFHMBRY 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:

オペレーターが入力処理を行う場合は常に、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済みます。
- 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た (DFHBMRY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

オペレーターが適切なデータや不適切なデータを入力した場合、適切なデータを保管する必要があります。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済みます。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 114 ページの『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 (エラー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初期入力を処理した後の次のステップは、トランザクションの出力を準備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 485 ページの『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、487 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、493 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・フィールドを「了解」応答によって完成させる必要があります (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:

MAPFAIL 例外条件は、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場合に発生します。

MAPFAIL 条件は、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 で発生します。また、以下の両方の状態に該当するときにオペレーターが画面から ENTER キーまたは PF キーを使用した場合にも発生します。

- 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)。
- カーソルが、マップで定義および命名されているフィールド内にないか、またはマップで CURSLOC=YES と指定されていない。

オペレーターがあまりにも性急に ENTER キーを安易に押したり、誤って「短縮読み取り」キーを押したりすることがあるかもしれません。ユーザーが使いやすいようにするため、MAPFAIL 条件が発生した後は、トランザクションをエラー終了させるのではなく、画面を最新表示にすることをお勧めします。

また、MAPFAIL 条件は、現行または直前のタスクで、SEND MAP または同等のものを指定して、最初に形式設定せずに RECEIVE MAP を発行した場合にも発生します。さらに、送信したのと異なるマップを使用すると発生することがあります。そのような状況は、論理エラーが通知される結果とな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ことを意味する場合もあります。例えば「クイック・アップデート」の例では、準備処理、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に画面に空マップを作成することが許可されません。そのようなこと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 後の画面の最新表示に必要なコードを利用することができます。必要なコードは以下のとおりです。

```
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.
```

このコードは、オペレーターにエスケープの方法を示します。エスケープしようとする操作が MAPFAIL の根本原因となることがあるためです。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、次のように 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 が発生すると、入力マップ構造はヌルにクリアされないで、他の場合と同様、そのようなクリア設定にプログラム論理が依存するかどうか、この条件をテストする必要があります。

他の例外条件の場合と同じく、HANDLE CONDITION コマンドを発行して MAPFAIL をインターセプトすることができます。ただし、HANDLE CONDITION コマンドを発行する場合に受け取られる AID について HANDLE AID がアクティブにすると、制御は AID について指定されているラベルに移るので、MAPFAIL について指定されているラベルには制御が移りません。詳細については、508 ページの『HANDLE AID コマンドの使用』を参照してください。この状況では、MAPFAIL について HANDLE を発行していても、EIBRESP のテストも行われない限り、MAPFAIL に気付きません。

## 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 z/OS Communications Server 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 EIBAUD および EIBCUSR 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。

## 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、521 ページの『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:

論理メッセージを構成するときは、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 IREQID 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。
- 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。
- 前述のオプションが競合しない限り、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することができます。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。528 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 547 ページの『区分画面サポート』および 556 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、532 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。『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後で作成できます。

#### **RETAIN および RELEASE:**

**SEND PAGE** コマンドのオプションは、いつ、どのようにページを端末に送達するかを BMS に指示します。

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が、その代わりに 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 ( 519 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** コマンドのオプションは、ページを端末に送達する方法を 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。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、前述のように、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いずれも、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 トランザクションについて詳しくは、CSPG - ページ検索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、Troubleshooting for recovery processing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、533 ページの『メッセージ・ルーティング』を参照してください。

## 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:

マップは浮動できます。つまりマップは、同一ページに既書き込まれたマップから、およびページのいずれかの端からの相対関係で位置決めできます。

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る、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。

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。

この機能は、**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 (525 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 (524 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 (525 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。SEND MAP 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 (525 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 (525 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。

525 ページの図 114 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。

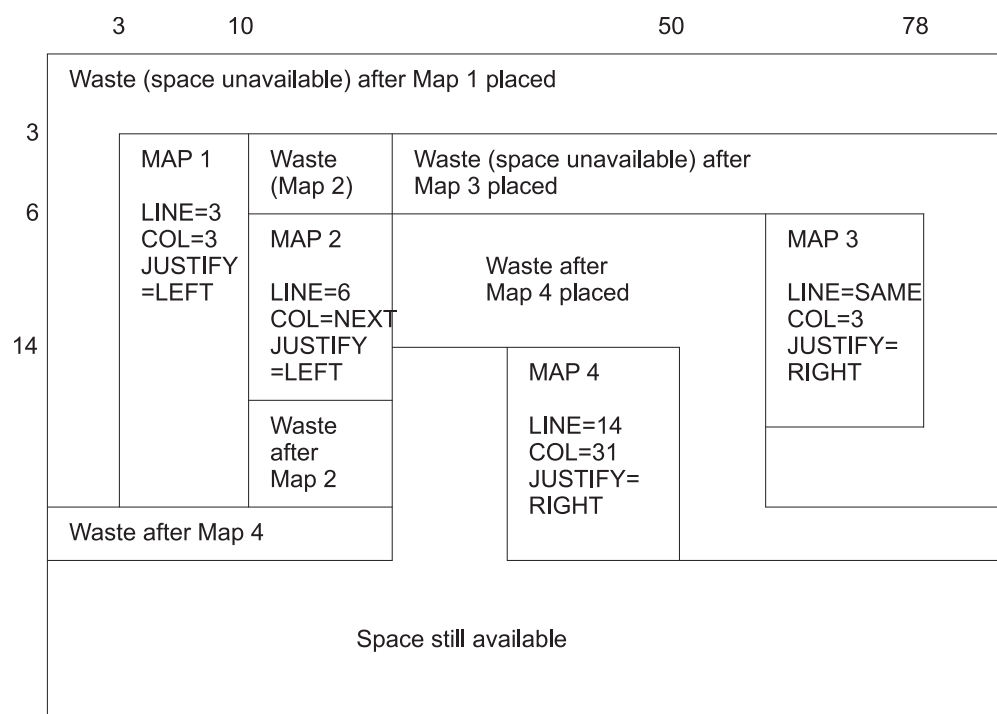


図 114. ページ上のマップの連続配置。それぞれ使用不能になるスペースを示す。

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、522 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 (523 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。

以下のオプションが、累積の処理に適用されます。

- **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:

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・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も、ユーザーにとってできるかぎり作業を簡単に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。所定のタスクのためのパス長は、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、494 ページの『出力画面の作成』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。見ることで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、114 ページの『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、460 ページの『BMS サポート・レベル』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
- 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 ( 493 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』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・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 を使用する場合は、これらは BMS が行います)。

## 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 ( 517 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 529 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と同一の規則が適用されます ( 516 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 ( 515 ページの『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。499 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、531 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 ( 531 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 (529 ページ) から繰り返します。

テキスト行:

BMS は、端末での表示のためにテキストを行に分割する、適切に定義された規則を使用します。

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。437 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らず、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、432 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。344 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またはトレーラーを配置するために、定様式データを使用してデータのブロックを示します。

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。

|   |   |   |   |       |
|---|---|---|---|-------|
| L | L | P | C | PNFLD |
|---|---|---|---|-------|

<———— TEXT ————>

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。ここで各部の意味は次のとおりです。

- 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。P に対して X'0C'、X'15'、X'17'、X'26'、または X'FF' を使用しないでください。これらの値は、他の目的のために予約されています。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.
```

各部の意味は次のとおりです。

```
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

プシオ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 (つまり、ACCUM オプションが使用されない) 限り、これらの形式は通常の SEND TEXT コマンドまたは SEND MAP コマンドと混同することができます。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。

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。

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・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なことがあります。 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)。

### 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ことができます。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、519 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 (440 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、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、543 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:

ルーティングされたメッセージに対して、特定のクラスのオペレーターを要求する、特定のオペレーターを指定する、または特定の端末を指定するなど、いくつかの方法で宛先を指定できます。

- 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せん。

- メッセージを受信す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なら、**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いわけではありませんが、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、**TYPETERM** 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用されている場合、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 ROUTEDMSG (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 は上記の条件の最初の 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 ROUTEDMSG 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らず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)。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 を入力します。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 バイト長です。

表 49. 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) がいずれにも伴います。LDC について詳しくは、556 ページの『論理デバイス・コンポーネント』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 50. 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                         |
| 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は異なります ( 556 ページの『論理デバイス・コンポーネント』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:

メッセージは、**ROUTE** コマンドの発行後しばらくしてから送達されることがありますが、これは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いとではありません。

- 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、534 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 は適格な端末にメッセージを送達できない場合、送達を定期的に継続します。

送達は次の条件のいずれかが起こるまで再試行されます。

- 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

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、またはペ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。(CSPG - ページ検索で、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。以下に例を示します。

```
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 115. 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、それ以外のメッセージと同じ方法で作成します。ただし、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。

## 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 ( 522 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れなくなるまで、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

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。

516 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。

## 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。

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、アプリケーションで FREEMAIN 要求によって解放されない限り、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。長時間実行トランザクションの場合は FREEMAIN 要求を使用してこれらのストレージ域を解放することをお勧めします。ただし、これらの領域は、タスクが完了した時点で CICS によって解放されます。

## 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ています。

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戻され、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、アプリケーションで FREEMAIN 要求によって解放されない限り、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。長時間実行トランザクションの場合は FREEMAIN 要求を使用してこれらのストレージ域を解放することをお勧めします。ただし、これらの領域は、タスクが完了した時点で CICS によって解放されます。

### **MAPPINGDEV アセンブラー・アプリケーションの例:**

この例は、変更されたバージョンの FILEA オペレーター命令サンプルを使用し、MAPPINGDEV 機能に関連付けられたキーワードをコーディングする方法を示しています。

547 ページの図 116 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推奨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 116. MAPPINGDEV アプリケーションの ASM 例

## 区分画面サポート

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。

区分画面には、標準 BMS が必要です。

3270 ファミリーのメンバーである IBM 3290 ディスプレイと、IBM 8775 は、区分化をサポートする装置の主な例です。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、553 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、550 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じコマンドで、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・バッファ空間も分割します。区分化した装置では、一般にバッファ容量が画面より大きいので、区分画面の中には余分なバッファ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・ストレージを区分画面の表示スペースといいます。

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 551 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 (555 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。

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。

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全体です。バッファ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。

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。

「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が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、ユーザビリティが高まります。こうすると、オペレーターは常に同じ位置でメッセージを見ることができます。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、551 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 (552 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 DFHPD 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。

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、470 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 사용할 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

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 CHAR.SIZE 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 CHAR.SIZE ディメンションの積です。区分画面の幅 は、桁数と水平 CHAR.SIZE の積です。

DFHPDI 区分画面定義に CHAR.SIZE サイズを指定しないと、BMS は DFHPD 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。DFHPD にも CHAR.SIZE 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 ゾルトを使用します。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 (481 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:

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、481 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、データを入力する区分画面 (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 回目まで続けて行くと、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。515 ページの『その他の入力形式設定』に説明があります。

区分画面の ASSIGN オプション:

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 ( 542 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せんが、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 556 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 ( 563 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。プログラム論理で、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 ( 508 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。これらのすべての状態と、それぞれに該当するコマンドについては、BMS マクロの資料を参照してください。

端末の共用:

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。

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。( 549 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、入力を入力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。

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、69 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。

## 特殊ハードウェアのサポート

区分画面に加え、BMS では他の特殊ハードウェア機能もサポートします。

BMS は、以下の機能をサポートします。

- 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用する場合があります。

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、TYPETERM リソースおよび Terminal control table (TCT) 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)。519 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 ( 554 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。論理デバイスのページ検索について詳しくは、CSPG - ページ検索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 メッセージをルーティングすることはできません。

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 アプリケーション開発のリファレンスには、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 Data Stream Programmers 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。

以下の機能がサポートされています。

- 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、オペレーターがブ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ブ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 Data Stream Programmers 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をサポートします。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 (494 ページの『出力画面の作成』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 キーかファンクション・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、ファンクション・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 Data Stream Programmers Reference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・プロセッサと接続されたサブシステムの間で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、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、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、332 ページの『バッチ・データ交換の使用』を参照してください。

## **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、一般の入力フィールドについて 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。

入力の合間に、ユーザーが画面で入力しなかった情報を保管できます。プログラムで MDT をオンにするのはこのためです。しかし、この記憶技法はデータ量が少ない場合にのみ適しており、関係する伝送オーバーヘッドのためリモート端末よりもローカル端末に適しています。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。アプリケーションによっては、ユーザーはいくつか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了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。ただし、変更されたフィールドのみが次の読み取りで送信されるため、プログラムで、各サイクルからの入力を保存し、新しいデータと古いデータをマージする必要があります。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、487 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。以下のことを確認します。

- 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 オプションを指定すると、マップ上の 定数 データしか送信されません。また、プログラムからの変数データのマージも行われません。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

不定形式データを画面に送信するか、または定様式画面を BMS の外側に作成する場合、set buffer address (SBA) および repeat-to-address (RA) 命令をデータ・ストリームに挿入することにより、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、画面上に未使用領域がかなりあって、その後にデータが続いている場合はいつも有効です。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。

## ブランクの代わりにヌルの使用

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。ただし、ブランク・フィールドがユーザーによって変更され、その後に読み取られる可能性がある場合は、NULL は送り戻されることがないため、NULL を使用してください。

## 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、強力な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、かなりのオーバーヘッドを必要とします。

詳しくは、309 ページの『効率的なデータ・セットの操作』を参照してください。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、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・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・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ヘスキップなど)、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な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、前述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 コマンドのプログラミング情報について詳しくは、START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。

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付

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 (432 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 5 章 非同期要求のための開発

CICS 非同期 API で使用するためのアプリケーションを開発する場合は、非同期 API コマンドのフロー、および各コマンドの使用可能なオプションを考慮する必要があります。非同期要求の一部として入力を渡したり応答を受け取ったりする場合は、チャンネル処理も考慮する必要があります。

### 非同期 API コマンドの使用

1 つ以上の子タスクを開始してそこから応答を取り出すために非同期 API コマンドを使用する CICS アプリケーションを開発することができます。

次の例では、クレジット・カード・アプリケーションを使用して、コマンドの使用方法を示しています。

### 信用調査の子タスクの開始

1. 以下のようにして、顧客の詳細を新しいチャンネルに配置します。

```
EXEC CICS PUT CONTAINER('CUSTDETAILS') CHANNEL(customerDetails)
```

2. チャンネルで最初の信用調査サービスを呼び出します。

```
EXEC CICS RUN TRANSID('CRD1') CHILD(creditCheck1_tkn) CHANNEL(customerDetails)
```

3. 同じチャンネルで 2 番目の信用調査サービスを呼び出します。

```
EXEC CICS RUN TRANSID('CRD2') CHILD(creditCheck2_tkn) CHANNEL(customerDetails)
```

親プログラムは、**EXEC CICS RUN TRANSID** コマンドを発行した後に他の処理を続けることが可能で、その後の任意の時点で **EXEC CICS FETCH** コマンドを発行して子タスクの結果を取得することができます。

### 特定の子タスクからの応答の取得

1. 以下のように、最初の子から応答を取り出します。

```
EXEC CICS FETCH CHILD(creditCheck1_tkn) CHANNEL(credReply1_chan)
 COMPSTATUS(credReply1_status)
```

この場合、子タスクから結果が返されるまで、親処理はブロックされます。オプション・パラメーター **NOSUSPEND** および **TIMEOUT** を使用することで、必要に応じてブロックを防止または制限できます。

2. **EXEC CICS FETCH CHILD** コマンドの応答コードを調べ、コマンドが正常に実行されたことを確認します。
3. **EXEC CICS FETCH CHILD** コマンドが正常に実行された場合は、子の **COMPSTATUS** を調べて正常に完了したことを確認します。

- 子タスクが正常に完了したことが **COMPSTATUS** に示されている場合は、応答チャンネルから応答を取得します。

```
EXEC CICS GET CONTAINER('RESULT') CHANNEL(credReply1_chan)
```

- 子タスクが異常終了したことが **COMPSTATUS** に示されている場合は、親アプリケーションのロジックによって実行内容が決まります。親トランザクションが完了すると、CICS はすべてのリソースをクリーンアップします。

4. 2 番目の子タスクに対して手順 1 から 3 を繰り返します。

## 完了した子タスクからの応答の取得

**EXEC CICS FETCH ANY** コマンドを使用して、完了した子タスクから応答を取り出すこともできます。このコマンドは、最初に完了した子タスクの結果を返します。

```
EXEC CICS FETCH ANY(creditCheck_tkn) CHANNEL(credReply_chan)
 COMPSTATUS(credReply_status)
```

**FETCH ANY** コマンドが完了したことと、子の **COMPSTATUS** が正常であることを確認してください。

## 子の解放

親タスクの一環として、子タスクを解放することができます。これにより、関連するリソースが整理され、システム・ストレージの使用を最小限に保つことができます。これは特に、親タスクが長時間実行される場合に効果があります。例えば、3 つの子タスクを開始し、1 つの結果のみが必要な場合は、**FREE CHILD** を使用して残りの 2 つの子タスクをクリーンアップすることができます。

```
EXEC CICS FREE CHILD(creditCheck1_tkn)
EXEC CICS FREE CHILD(creditCheck2_tkn)
```

## オプション・パラメーター

### EXEC CICS RUN TRANSID

- **CHANNEL** はオプションです。これは次の目的で使用します。
  - データを子に渡す。
  - 子からデータを受け取る。
  - 子にデータを渡してその子からデータを受け取る。
- **RESP** はオプションです。これは、コマンドが正常に実行されたかどうかを確認する目的で使用します。

### EXEC CICS FETCH コマンド

- **CHANNEL** はオプションです。これは、最初の **RUN TRANSID** コマンドにチャンネルが指定されている限り、子タスクからデータを受け取る目的で使用できます。
- **NOSUSPEND** はオプションです。これは、**FETCH** コマンドによるブロック機能を無効にするために使用します。子タスクが完了したかどうかにかかわらず、コマンドは即時に戻ります。
- **TIMEOUT** はオプションです。これは、**FETCH** コマンドが戻るまでに待機する時間を指定する目的で使用します。
- **ABCODE** はオプションです。子が異常終了したことが **COMPSTATUS** に示されている場合、**ABCODE** には子の異常終了コードが格納されます。

## 非同期 API コマンドでのチャンネルの使用

次の手順は、チャンネルを使用して親タスクと子タスクの間で入力と出力を受け渡す方法の例を示しています。

1. 親タスク PROG1 は、入力をチャンネル内の子タスクに渡し、それを別のプログラムにリンクします。

```
EXEC CICS PUT CONTAINER('REQ') FROM(REQUEST) CHANNEL('ASYNC-CHANNEL')
EXEC CICS RUN TRANISD('CHILD') CHANNEL('ASYNC-CHANNEL') CHILD(CHILD-TOKEN)
EXEC CICS LINK PROGRAM('PROGP2')
```

親によって作成された ASYNC-CHANNEL のコピーが子に渡され、そのコピーが子の初期プログラムの現在のチャンネルになります。チャンネルの名前はコピー・プロセスによって保持されるため、子プログラム PROGC1 が **EXEC CICS ASSIGN CHANNEL** を発行した場合は、ASYNC-CHANNEL が返されます。

2. 子タスク PROGC1 は、現在のチャンネルから入力を取得します。

```
EXEC CICS GET CONTAINER('REQ')
```

次に、子はそのロジックを続行し、親タスクに対する応答を生成し、それを現在のチャンネル内のコンテナに入れます。

```
EXEC CICS PUT CONTAINER('RESP') FROM(RESPONSE)
EXEC CICS RETURN
```

3. 親タスク PROGP2 は、子応答をフェッチし、ビジネス・ロジックを続行してから、PROGP1 に戻ります。

```
EXEC CICS FETCH CHILD(CHILD-TOKEN) CHANNEL (FETCHED-CHANNEL)
EXEC CICS GET CONTAINER('RESP') CHANNEL(FETCHED-CHANNEL)
```

親タスクが子から応答チャンネルをフェッチすると、CICS によって固有のチャンネル名が生成され、チャンネルが現在のプログラム・リンク・レベルにバインドされます。この例では、フェッチされたチャンネルは、PROGP2 によって所有されます。PROGP2 が PROGP1 に戻ると、フェッチされたチャンネルは CICS によって削除されます。子の応答チャンネルは、親タスクによって 1 回だけフェッチされます。



---

## 第 6 章 アセンブラー言語アプリケーションの開発

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、16 MB 境界の上下いずれかに割り振られます。

### サンプル・プログラム

アセンブラー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。これらのプログラムは、AMODE(64) であり、DFH\$AREP を除き、相対アドレッシングを使用します。DFH\$AREP は、相対アドレッシングを使用しますが、HANDLE CONDITION LABEL コマンドの使用方法を示すためのものであるので、AMODE(31) です。

表 51. 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\$AMNU オペレーター用指示 (3270) | DFH\$AGA | DFH\$AMA | AMNU           |
| DFH\$AALL 更新 (3270)        | DFH\$AGB | DFH\$AMB | AADD、AINQ、AUPD |
| DFH\$ABRW ブラウズ (3270)      | DFH\$AGC | DFH\$AMC | ABRW           |
| DFH\$AREN 受注 (3270)        | DFH\$AGK | DFH\$AMK | AORD           |
| DFH\$ACOM 受注キュー印刷 (3270)   | DFH\$AGL | DFH\$AML | AORQ           |
| DFH\$AREP レポート (3270)      | DFH\$AGD | DFH\$AMD | AREP           |

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\$AFLA 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ことができます。

CEDA INSTALL GROUP(DFH\$AFLA)

以下のレコード記述ファイルが提供されています。

- DFH\$AFIL: FILEA レコード記述子
- DFH\$AL86: L860 レコード記述子

---

## 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、いくつかの制約事項および要件が適用されます。

## LEASM オプション

LEASM オプションによって変換され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。

- 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、Language Environment 共通アンカー域 (CAA) を指すために Language Environment によって予約されています。したがって、適切に保管および復元されていないプログラムでは、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。
- AMODE(64) プログラムはサポートされていません。

## EXEC CICS コマンドの LENGTH オプション

CICS コマンドに LENGTH オプションを指定する場合、アセンブラー言語での使用方式には有効なハーフワード長を指定してください。長さをゼロに指定したり、CICS 変換プログラムが認識できない変数を指定したりしないでください。これらを指定すると、記憶保護違反またはプログラム・チェックが起きる可能性があるためです。詳しくは、Assembler language argument valuesを参照してください。

次に、長さを正しく指定し、変数 COMMAL のアドレスをコマンド・プロセッサに渡している LINK コマンドの例を示します。

```
EXEC CICS LINK PROGRAM('PROG2') COMMAREA(COMMA)
LENGTH(COMMAL)
...
COMMA DC CL20'This is the COMMAREA'
COMMAL DC H'20'
```

次の例も、長さを正しく指定しています。

```
EXEC CICS LINK PROGRAM('PROG2') COMMAREA(COMMA)
LENGTH(=AL2(COMMAL))
...
COMMA DC CL20'This is the COMMAREA'
COMMAL EQU *-COMMA
```

次の例は正しくありません。CICS 変換プログラムは変数 COMMAL のタイプを認識できず、長さを含むハーフワード・フィールドのアドレスとして COMMAL の値を渡すためです。このようにすると、ランダムな長さの値が提供される場合があり、あるいは、該当のアドレスのストレージが使用不可の場合はプログラム・チェックになることがあります。

```
EXEC CICS LINK PROGRAM('PROG2') COMMAREA(COMMA)
LENGTH(COMMAL)
...
COMMA DC CL20'This is the COMMAREA'
COMMAL EQU *-COMMA
```

## 31 ビット・アドレッシング

31 ビット・アドレッシング・モードで実行され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。

- 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、169 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。
- CICS では、DFHEIENT マクロおよび DFHEIRET マクロ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。Language Environment のスタブ CEESTART を使用するアセンブラー言語プログラムでは、HANDLE ABEND PROGRAM と Language Environment サービス (CEEHDLR など) のいずれかを使用しなければなりません。CEEHDLR の詳細については、681 ページの『言語環境プログラム (Language Environment) の異常終了と条件処理』を参照してください。

64 ビットのアドレッシング・モードまたは 64 ビットの 2 項演算を使用するために 64 ビット・レジスターを使用する AMODE(24) または AMODE(31) 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

CICS は、非 Language Environment AMODE(64) アセンブラー言語 CICS アプリケーションのプログラム実行をサポートしています。ご使用のプログラムで、相対アドレッシングを使用しなければなりません。591 ページの『AMODE(64) アセンブラー言語プログラムの開発』を参照してください。このセクションで以降に記載するもの以外のすべての CICS API コマンドがサポートされています。CICS API については、CICS API コマンドを参照してください。

AMODE(64) プログラムについては、以下の API はサポートされていません。

- 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 (API)
- APPC 基本会話で使用するための CICS API コマンド (GDS コマンド)
- フロントエンド・プログラミング・インターフェース (FEPI)
- 共通プログラミング・インターフェース (CPI) コミュニケーション API
- DL/I 要求

AMODE(64) プログラムについては、以下のインターフェースはサポートされていません。

- CICS Db2 インターフェース
- CICS-MQ ブリッジ外部インターフェース

以下の CICS API コマンドは、AMODE(64) プログラム用として用意されています。

- FREEMAIN64
- GETMAIN64

- GET64 CONTAINER
- PUT64 CONTAINER

以下の CICS API コマンドは、非構造化例外処理に関連するものであり、AMODE(64) アセンブラー言語プログラムについてはサポートされていません。

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

これらのコマンドが使用されていると、変換プログラムによって検出されます。

AMODE(64) アセンブラー言語プログラムは、COBOL、C、C++、または PL/I の CALL ステートメントによって呼び出したり、アセンブラー言語の CALL マクロによって呼び出したりすることはできません。しかし、アセンブラー言語アプリケーション・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、また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。

AMODE(64) アセンブラー言語プログラムの場合、すべての EXEC CICS コマンドで NOHANDLE 条件を使用するものと想定されています。つまり、このコマンドを実行した結果として発生するいかなる条件についても処置は一切とられないということです。任意のコマンドで RESP オプションを使用して、このコマンドの実行中に何らかの条件が発生したかどうかをテストすることができます。RESP および RESP2 オプションおよび NOHANDLE オプションを参照してください。

AMODE(64) アセンブラー言語プログラムの場合、CICS は 64 ビット・レジスタの全体を使用します。レジスタ全体が有効であることを確認してください。

COMMAREA は、64 ビット・ストレージに含めることはできません。COMMAREA について詳しくは、165 ページの『COMMAREA』を参照してください。

AMODE 64 タスク関連のユーザー出口 (TRUE) はサポートされないため、AMODE(64) アプリケーションは TRUE を呼び出すことができません。

64 ビット・アドレッシング・モードおよび 64 ビット 2 項演算の使用方法について詳しくは、「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。

## 64 ビット常駐

CICS では、64 ビット常駐モード (RMODE(64)) はサポートされておらず、すべての RMODE(64) プログラムが RMODE(31) として処理されます。つまり、RMODE(64) プログラムは、64 ビット (2 GB 境界より上) ストレージではなく、31 ビット (16 MB 境界より上) ストレージにロードされます。

## アクセス・レジスター

z/Architecture® システムの拡張アドレッシング機能を利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、以下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・モードにいた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は基本アドレス・スペースにいたなければなりません。
- 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る、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。

## 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、リンク先プログラムが 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 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。これらの呼び出しによってリンケージ・スタックが変更されると、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。

## 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されるアセンブラー言語プログラムでは、以下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。

## コメント

CICS コマンドについてのコメントを追加する場合は、最後の引数の後ろに区切り文字としてピリオドまたはコンマを使用します (ただし、これが行えるのはアセンブラー言語だけです)。以下に例を示します。

```
EXEC CICS ADDRESS EIB(MYUEIB), @F1A
```

EXEC CICS コマンドでピリオドまたはコンマが使用される場合、後続の行は、列 2 と列 16 の間で始まる必要があり、継続文字は列 72 に指定します。後続の行は、列 17 より後ろから開始することはできません。コンマもピリオドも追加されていない場合、後続の行は、列 2 またはそれ以降から始まって列 71 までに終了する必要があります。継続文字は列 72 に指定します。

## EXEC CICS アセンブラー・インターフェースのコーディング

AMODE(24) および AMODE(31) アセンブラー言語プログラムの場合、オプションで独自の DFHEIENT マクロを指定できます。アセンブラー言語プログラムが AMODE(64) であることを宣言するには、z/OS アセンブラー SYSSTATE マクロを使用します。AMODE(64) アセンブラー言語プログラムの場合、DFHEIENT マクロを指定する必要があり、オプションで独自の DFHEIRET マクロを指定できます。

### AMODE(24) および AMODE(31) アセンブラー言語プログラム

#### DFHEIENT マクロ

AMODE(24) および AMODE(31) プログラムの場合、変換プログラムによって自動的に挿入される DFHEIENT マクロから提供される値は、4095 バイトより大きい変換出力を生成するアプリケーション・プログラムには不十分である可能性があります。この状態では、独自のバージョンの DFHEIENT マクロを提供できます。変換プログラムによってそのバージョンの DFHEIENT マクロが自動的に挿入されないようにするには、NOPROLOG 変換プログラム・オプションを指定します。

例えばデフォルトでは、変換プログラムによって、基底レジスター (レジスター 3) が 1 つだけセットアップされます。これにより、プログラムのアドレス可能性の問題が生じる可能性があります。ユーザーの DFHEIENT マクロに CODEREG オペランドを使用することができるため、複数の基底レジスターを指定することができます。CSECT ステートメントと同じラベルを持つ独自のバージョンの DFHEIENT マクロをコーディングすると、ソース・プログラム内の CSECT ステートメントが置き換えられることがあります。ラベルなしで DFHEIENT マクロをコーディングする場合は、CSECT ステートメントの直後に配置する必要があります。

基底レジスターを指定するには、以下のオペランドを使用します。

- CODEREG - 基底レジスター (レジスター 13、14、15、0、および 1 は許可されていません)
- DATAREG - 動的ストレージ・レジスター
- EIBREG - EIB をアドレッシングするためのレジスター

例えば、次の単純なアセンブラー言語アプリケーション・プログラムでは、BMS コマンドの SEND MAP を使用して、マップを端末に送ります。

```
INSTRUCT CSECT
 EXEC CICS SEND MAP('DFH$AGA') MAPONLY ERASE
 END
```

次のサンプル・コードでは、このプログラムの基底レジスターとデータ・レジスターの数を増やします。

```
INSTRUCT DFHEIENT CODEREG=(2,3,4),
 DATAREG=(13,5),
 EIBREG=6
 EXEC CICS SEND
 MAP('DFH$AGA')
 MAPONLY ERASE
 END
```

記号レジスター DFHEIPLR は、明示的に指定されたあるいはデフォルトに獲得された、最初の DATAREG と同等です。DFHECALL マクロにより、レジスター 13 は必ず、DFHEISTG によって動的ストレージに定義された保管域をアドレス指定するため、最初のデータの動的ストレージ・レジスターとしてレジスター 13 を使用することをお勧めします。このレジスターを使用しないと、DFHECALL によって生成されたコードにより、レジスター 13 を操作するための余分な命令が追加されます。

DFHEIPLR は、CICS コマンドの展開によって、DFHEIENT で設定された値を含んでいると見なされます。このレジスターは、専用にするか、各 CICS コマンドの前に必ず復元するかしてください。

また、DFHEIENT マクロを使用して、ユーザーのプログラムで相対アドレッシング命令を使用することを指定することもできます。相対アドレッシングを使用する場合は、プログラム命令をアドレス指定するために基底レジスターを使用する必要はありませんが、そのプログラム内の静的データをアドレス指定するときには 1 つ以上の基底レジスターを使用する必要があります。DFHEIENT マクロで以下のオペランドを指定します。

- CODEREG=0、プログラム命令をアドレス指定するのにレジスターを使用しないよう指定する場合。
- STATREG、プログラム内の静的データ域のアドレス指定のため 1 つ以上のレジスターを指定する場合。
- STATIC、プログラム内の静的データの開始アドレスを指定する場合。

相対アドレッシングを使用する場合は、(z/OS で提供される) IEABRCX DEFINE マクロを使ってアセンブラー・ニーモニックを再定義することにより、ブランチ命令で相対ブランチ命令が使われるようにしてください。また、すべての LTORG ステートメントと、EXECUTE ステートメントのターゲットである命令が、STATIC オペランドで指定されたラベルの後に配置されていることも確認します。以下に例を示します。

```

IEABRCX DEFINE Define relative branch mnemonics
RELATIVE DFHEIENT CODEREG=0,STATREG=(8,9),STATIC=MYSTATIC
....
EX R2,VARMOVE Execute instruction in static area
....

MYSTATIC DS 0D Static data area
MYCONST DC C'constant' Static data value
VARMOVE MVC WORKA(0),WORKB Executed instruction
LTORG , Literal pool

```

IEABRCX マクロの詳細については、「z/OS MVS プログラミング: アセンブラー・サービス解説書 IAR-XCT」の『IEABRCX - 相対ブランチ・マクロの拡張機能』を参照してください。

DLI オプションで変換したアセンブラー言語プログラムには、各 CSECT ステートメントの後に DLI 初期設定呼び出しが挿入されています。4095 バイトより大きいアセンブラー言語プログラムで複数の基底レジスターを設定するときに DFHEIENT マクロの CODEREG オペランドを使用しない場合は、LTORG ステートメントを組み込むことによって、DFHEIENT または DLI 初期設定呼び出しで生成されたリテラルが、基底レジスターの範囲内に入るようにしてください。

通常、LTORG ステートメントは、長さが 4095 バイトを超えるすべての CSECT が必要です。

#### DFHEIRET マクロ

AMODE(24) および AMODE(31) プログラムの場合は、DFHEIRET マクロが EPILOG プログラムを呼び出して、アプリケーション・プログラムの作業用ストレージを解放します。

DFHEIRET マクロには次のパラメーターを指定できます。

#### RCREG

戻りコードを配置するレジスターを指定します。値を指定しない場合は、戻りコードがプログラムの呼び出し元に返されます。デフォルト値はありません。

変換プログラムによって、パラメーターが指定されずに DFHEIRET マクロが END ステートメントの直前に挿入されます (ただし、挿入されないように NOEPILOG 変換プログラム・オプションを指定した場合を除きます)。END は、変換プログラムで認識されるように、大文字にする必要があります。この変換プログラムによって挿入された DFHEIRET マクロの前に DFHEIRET マクロが呼び出されると、変換プログラムによって挿入されたマクロはコードを生成しません。

### AMODE(64) アセンブラー言語プログラム

CICS Transaction Server は、64 ビット・アドレッシング・モードで実行される非 Language Environment アセンブラー言語プログラムをサポートします。

#### SYSSTATE マクロ

アプリケーションが AMODE(64) であることを宣言するには、z/OS アセンブラー SYSSTATE (システム状態の識別) マクロを使用します。

1. SYS1.MACLIB マクロ・ライブラリー (SYSSTATE マクロが含まれている) の SYSLIB ステートメントをコンパイル JCL に組み込みます。
2. 以下のいずれかを確認します。
  - パラメーター AMODE64=YES が指定された SYSSTATE マクロが、CICS 変換プログラム・オプション・ステートメントの直後にある。以下に例を示します。

```
*ASM XOPTS(NOPROLOG NOEPILOG)
 SYSSTATE AMODE64=YES
```
  - プログラムに CICS 変換プログラム・オプションが指定されていない場合は、パラメーター AMODE64=YES が指定された SYSSTATE マクロが最初のステートメントになります。

SYSSTATE ステートメントは 1 行である必要があります。CICS は、このステートメントの継続行をサポートしないためです。SYSSTATE マクロの詳細については、「z/OS MVS プログラミング: アセンブラー・サービス解説書 (IAR-XCT)」の『SYSSTATE - システム状態の識別』を参照してください。

アプリケーションがこの方法で AMODE(64) として宣言されると、次の CICS 提供マクロによって AMODE(64) コードが生成されます。

- DFHEIENT

- DFHEISTG
- DFHEIRET
- DFHECALL

CICS 提供マクロによって初めてコードが生成されるときに、SYSSTATE マクロが指定されていなかった場合、またはパラメーター AMODE64=YES が指定されずに SYSSTATE マクロが指定されていた場合は、これらのマクロによって AMODE(24) または AMODE(31) コードが生成されます。

#### **DFHEIENT マクロ**

AMODE(64) プログラムの場合は、DFHEIENT マクロが AMODE(64) PROLOG プログラムを呼び出します。これにより、ユーザー変数を保持するため、および CICS で使用できるように、作業用ストレージが割り振られます。PROLOG プログラムによって、このストレージの CICS 部分がセットアップされます。PROLOG プログラムが戻ると、DFHEIENT パラメーターで指定されたレジスターが、このコードによってセットアップされます。

相対アドレッシングのみがサポートされるため、DFHEIENT マクロ・パラメーターを指定して、プログラムが相対アドレッシング命令を使用するように指定する必要があります。相対アドレッシングの場合は、プログラム命令をアドレス指定するために基底レジスターは必要はありませんが、そのプログラム内の静的データをアドレス指定するときには 1 つ以上の基底レジスターを使用する必要があります。STATREG パラメーターと STATIC パラメーターを使用して、1 つ以上の静的レジスターをセットアップします。

DFHEIENT マクロには以下のパラメーターを指定できます。

#### **CODEREG**

値 0 (デフォルト) を指定して、相対アドレッシングを指定します。

#### **DATAREG**

アプリケーション・プログラム用の作業用ストレージ・レジスターを 1 つ以上指定します。デフォルトはレジスター 13 で、最初のデータ動的ストレージ・レジスターとしてレジスター 13 を使用することをお勧めします。このレジスターを使用しないと、DFHECALL マクロによって生成されたコードにより、レジスター 13 を操作するための余分な命令が追加されます。DFHECALL マクロにより、レジスター 13 は必ず、DFHEISTG によって動的ストレージに定義された保管域をアドレス指定します。

#### **EIBREG**

EXEC インターフェース・ブロック (EIB) をアドレス指定するために使用するレジスターを指定します。デフォルトはレジスター 11 です。

#### **STATREG**

アプリケーション・プログラムが使用する静的レジスターを 1 つ以上指定します。デフォルトはレジスター 3 です。

#### **STATIC**

静的区域の開始のアセンブラー・ラベルを指定します。値を指定する必要があります。このパラメーターのデフォルトはありません。

NOPROLOG 変換プログラム・オプションを使用し、相対アドレッシングに適したパラメーターとともに DFHEIENT マクロを指定します。  
DFHEIENT マクロを指定しなかった場合は、変換プログラムによって、必須パラメーターなしの DFHEIENT マクロが挿入され、次のエラーが発生します。

12,DFHEIENT - AMODE 64 - STATIC REQUIRED

次の 2 つの DFHEIENT ステートメントの例では、同じコードが生成されます。最初のステートメントでは、すべてのパラメーターがコーディングされます (デフォルト値が指定されます)。2 番目のステートメントでは、デフォルト値を持たないパラメーターのみがコーディングされます。

DFHEIENT CODEREG=0,DATAREG=13,EIBREG=11,STATREG=3,STATIC=STAT

DFHEIENT STATIC=STAT

#### DFHEIRET マクロ

AMODE(64) プログラムの場合は、DFHEIRET マクロが AMODE(64) EPILOG プログラムを呼び出して、アプリケーション・プログラムの作業用ストレージを解放します。

DFHEIRET マクロには次のパラメーターを指定できます。

#### RCREG

戻りコードを配置するレジスターを指定します。値を指定しない場合は、戻りコードがプログラムの呼び出し元に返されます。デフォルト値はありません。

変換プログラムによって、パラメーターが指定されずに DFHEIRET マクロが END ステートメントの直前に挿入されます (ただし、挿入されないように NOEPILOG 変換プログラム・オプションを指定した場合を除きます)。END は、変換プログラムで認識されるように、大文字にする必要があります。この変換プログラムによって挿入された DFHEIRET マクロの前に DFHEIRET マクロが呼び出されると、変換プログラムによって挿入されたマクロはコードを生成しません。

DFHEISTG ストレージの詳細については、590 ページの『動的ストレージの拡張』を参照してください。DFHECALL マクロおよび DFHEIRET マクロの詳細については、DFHECALL マクロ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

アセンブラー言語プログラムは、Language Environment 準拠と非準拠のいずれかに分類されます。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。

Language Environment 準拠のアセンブラー言語ルーチンは、CEEENTRY およびそれに関連する Language Environment 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、684 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。

64 ビット・アドレッシング・モードは、Language Environment 準拠のアセンブラー言語プログラムではサポートされていません。

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 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 は、(ストレージの) 次の使用可能バイト)。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、次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
- 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、Language Environment 準拠のアセンブラー言語ルーチンの実行時には常にそのルーチンの 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。プログラム内の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・プロセスでプログラムを RMODE (24) に設定しなければなりません。

## 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らない)。

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

アセンブラー言語アプリケーション・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。

### 64 ビット・アドレッシング・モード

コマンドを含んでいる AMODE (64) 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つことができます。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。

### 24 ビットおよび 31 ビット・アドレッシング・モード

コマンドを含んでいる AMODE(24) および AMODE(31) 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 プログラム定義をもつことができます。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。ただし、コマンドを含んでいる AMODE(24) プログラムおよび AMODE(31) 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したり、アセンブラー言語の CALL マクロによって呼び出したりすることもできます。

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は、684 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。異なるアドレッシング・モードを使用するプログラムについて詳しくは、169 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 が出力されますが、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 117. 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 118. ASMTEST ASSEMBLE

以下のように、CICS で提供される JCL プロシージャを使用して、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ていることを前提としています。

DFHYITPL プロシージャ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 という名前のエントリー・ポイントが含まれているので、リンケージ・エディターまたはバインダー・プログラムが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。

リンク・エディットからの出力で、プログラムの先頭に DFHEAI スタブが含まれる必要があります。そのためには、JCL のリンク・エディット・ステップ内に DFHEAI に関する ORDER および INCLUDE ステートメントが含まれる必要があります。アセンブラ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・エディットするために、SDFHPROC ライブラリー内の CICS 提供アセンブラ・プロシージャ DFHEITAL を使用するとき、このプロシージャの COPYLINK ステップは SDFHMAC(DFHEILIA) をコピーします。組み込まれる必要のある以下のステートメントが DFHEILIA に含まれます。

```
ORDER DFHEAI
INCLUDE SYSLIB(DFHEAI)
```

これらのステートメントは、プロシージャの LKED ステップでアセンブルされるアプリケーション・プログラムの前に連結される一時ファイルの中に配置されます。

ユーザー独自の JCL を作成する場合は、すべての言語に必要なエン트리・ポイントが DFHELII スタブに含まれているので、DFHELII スタブを組み込む必要があります。

アセンブラー言語アプリケーション・プログラムは、DFHEIENT マクロで開始し、DFHEIRET マクロで終了することができます。これらは CICS 変換プログラムによって自動的に挿入されるため、先の例のようにプログラムに EXEC CICS コマンドが存在し、それが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。CICS TS 2.2 では、DFHEIRET が相対アドレッシングに対応するように変更されたことに注意してください。ベース・レジスターがない場合、マクロ展開で LTORG が生成されなくなりました。アセンブラーは通常、END に達すると、残りの VCON を収集するために自動 LTORG を生成します。コード実行の結果として CSECT 連結の先頭に専用コード CSECT が生成された場合は、問題が発生する可能性があります。アセンブラーは END ステートメントに達すると、最初の CSECT に戻るため、そこからは VCON を解決できません。アセンブラー・リストを確認して、この種の専用制御セクションを削除することをお勧めします。詳しくは、『HLASM Language Reference』の『Unnamed section』の情報を参照してください。

---

## 動的ストレージの拡張

DFHEISTG という名前の DSECT 内にあるユーザーのソース・プログラムで変数を定義することによって、これらの変数に追加のストレージを提供するよう動的ストレージを拡張できます。

### 24 ビットおよび 31 ビット・アドレッシング・モード

AMODE(24) および AMODE(31) プログラムの場合、DFHEISTG DSECT を使用して取得できる最大の動的ストレージ量は 65 264 バイトです。DFHEISTG は予約名です。このストレージは X'00' に初期設定されます。変換プログラムは変換時に、DFHEISTG マクロを、ユーザーの DFHEISTG DSECT 命令の直後に挿入します。このようにして DSECT は、パラメーター・リスト、コマンド・レベルのインターフェース、およびユーザー変数に必要な動的ストレージを記述します。

DFHEISTG ストレージが必ず x'00' に初期化されるように、リンク・エディット時に CEEXOPT マクロの STORAGE オプションを使用します。例えば、CEEXOPT STORAGE=(,00) のようにします。アプリケーションが、ユーザー DFHEISTG 領域で定義されている任意の定数を伝搬または初期化することを確認してください。

591 ページの図 119 の例は、動的ストレージにあるそのような変数を使用する、簡単なアセンブラー言語アプリケーション・プログラムです。

Source program

```
INSTRUCT CSECT
 EXEC CICS SEND MAP('DFH$AGA') MAPONLY ERASE
END
```

This source program is translated to:

```
 DFHEIGBL , INSERTED BY TRANSLATOR
INSTRUCT CSECT
 DFHEIENT INSERTED BY TRANSLATOR
* EXEC CICS SEND MAP('DFH$AGA') MAPONLY ERASE
 DFHECALL =X'1804C00008000000000046204000020',
 (CHA7,=CL7'DFH$AGA*'),(____RF,DFHEIV00)
 DFHEIRET INSERTED BY TRANSLATOR
 DFHEISTG INSERTED BY TRANSLATOR
 DFHEIEND INSERTED BY TRANSLATOR
END
```

図 119. ユーザー変数の変換コード

## 64 ビット・アドレッシング・モード

非 Language Environment AMODE(64) アセンブラー言語プログラムの場合は、DFHEISTG マクロによって AMODE(64) DSECT が生成されます。DFHEISTG ストレージは、64 ビット・ストレージではなく 31 ビット・ストレージ (16 MB の境界より上、2 GB の境界より下) から取得されます。DFHEISTG DSECT を使用して取得可能な最大の動的ストレージ量は、65 264 バイトです。このストレージは X'00' に初期設定されます。

変換プログラムは変換時に、DFHEISTG マクロを、ユーザーの DFHEISTG DSECT 命令の直後に挿入します。このようにして DSECT は、パラメーター・リスト、コマンド・レベルのインターフェース、およびユーザー変数に必要な動的ストレージを記述します。

CICS は、DFHEISTG ストレージの前の部分を定義します。

DFHEIENT マクロからアプリケーション・プログラムに制御が返される前に、EIB ポインターである DFHEIBP および COMMAREA ポインターである DFHEICAP がセットアップされます。これらのポインターは、24 ビットまたは 31 ビット・ストレージでは 64 ビット・ポインターです。

---

## AMODE(64) アセンブラー言語プログラムの開発

CICS Transaction Server は、64 ビット・アドレッシング・モード (AMODE(64)) で実行される非 Language Environment (LE) アセンブラー言語プログラムをサポートしています。

### 始める前に

575 ページの『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』で該当のセクションを確認してください。

## このタスクについて

以下の手順を参照すると、AMODE(64) アセンブラー言語プログラムの開発に特に関係する情報の概要がわかります。

### 手順

1. z/OS アセンブラー SYSSTATE マクロを使用して、アプリケーションが AMODE(64) であることを宣言します。 580 ページの『EXEC CICS アセンブラー・インターフェースのコーディング』を参照してください。
2. 相対アドレッシングに関する該当のパラメーターを設定して DFHEIENT マクロを指定します。 580 ページの『EXEC CICS アセンブラー・インターフェースのコーディング』を参照してください。

DFHEISTG ストレージについて詳しくは、 590 ページの『動的ストレージの拡張』を参照してください。

3. EXEC CICS RETURN を使用しない場合は、DFHEIRET マクロを指定します。EXEC CICS RETURN を使用する場合は、変換プログラムによって挿入されるマクロを使用します。 580 ページの『EXEC CICS アセンブラー・インターフェースのコーディング』を参照してください。
4. プログラムを変換、アセンブル、およびリンク・エディットします。 1014 ページの『アセンブラー言語アプリケーション・プログラムの変換、アセンブル、およびリンク・エディット』を参照してください。

DFHECALL マクロについて詳しくは、DFHECALL マクロを参照してください。

スタブ・プログラムのリンク・エディットについて詳しくは、 915 ページの『AMODE(64) アプリケーションでの EXEC インターフェース・モジュールの使用』を参照してください。

## 第 7 章 C および C++ アプリケーションの開発

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。

アプリケーション・プログラミング言語に関する CICS サポートの変更点には、CICS Transaction Server for z/OS, バージョン 5 リリース 5 でサポートされている C コンパイラおよび C++ コンパイラと、z/OS におけるそれらの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 5 リリース 5 の資料において C および C++ に言及する場合、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を使用することを意味します。COBOL、PL/I、およびアセンブラ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サポートされていますが、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 ファウンデーション・クラスの使用法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、16 MB 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 16 MB 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 52. 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 |
| DFH\$DBRW ブラウズ (3270)      | DFH\$DGC | DFH\$DMC | DBRW           |
| DFH\$DREN 受注 (3270)        | DFH\$DGK | DFH\$DMK | DORD           |

表 52. 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\$DCOM 受注キュー印刷<br>(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ことができます。

```
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

## FLOAT コンパイラー・オプション

z/OS V1.11 XL C (または C++) 以降では、FLOAT(NOAFP) コンパイラー・オプション、または FLOAT(AFP(VOLATILE)) コンパイラー・オプションのいずれかを指定します。

- 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、C、および C++ で浮動小数点レジスターが保存されます。追加のコードが生成されるため、パフォーマンスが低下する場合があります。

## 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、いくつかの制約事項および要件が適用されます。

### 使用できない関数およびコマンド

以下の EXEC CICS コマンドは、非構造化例外処理に関連するものであり、C および C++ アプリケーションについてはサポートされていません。

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

- **POP HANDLE**

これらのコマンドが使用されているかどうか、変換プログラムによって診断されます。**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が設定されている 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++ ファイル操作では、`type=memory` を指定してオープンされるファイルのみについて操作が実行されます。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。機能する場合がありますが、CICS のパフォーマンスまたは実行に影響する可能性があります。

- **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。

- \$、#、および @ を含むフィールド名は、アセンブラで受け入れ可能であっても C または C++ コンパイラの異常終了の原因となるので、使用しないでください。
- C++ は、単一行のコメントについては「//」を使用します。そのようなコメントは、EXEC CICS コマンドの途中には置かないでください。例えば、以下のコードは無効です。

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

以下のコード例は有効です。

```
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かのように処理されます。そのため、条件が発生したのに処理されないで「システム処置」によってトランザクションが異常終了するというようなことは起こり得ません。制御は常に次の命令にフローするので、通常応答に関するテストはアプリケーションが担当します。

## 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、600 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。

## アドレッシング・モード

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、16 MB 境界より上に常駐できます。

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るプログラムにパラメーターを渡す場合には、以下のいずれかを行う必要があります。

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

64 ビット・アドレッシング・モード (AMODE(64)) は、C および C++ 言語プログラムについてはサポートされていません。

## 64 ビット常駐モード

CICS では、64 ビット常駐モード (RMODE(64)) はサポートされておらず、すべての RMODE(64) プログラムが RMODE(31) として処理されます。つまり、RMODE(64) プログラムは、64 ビット (2 GB 境界より上) ストレージではなく、31 ビット (16 MB 境界より上) ストレージにロードされます。

## 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 (Language Environment) 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。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

iscics() 関数は、既存のプログラムを適応させる場合や、CICS でだけでなく CICS 外部でも稼働するように設計されている新規プログラムを作成する場合に役立ちます。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、実行されていない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++ での引数の受け渡し

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、引数のあるデータ・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** 型に変換される場合があります。

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 53 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 53. EXEC CICS コマンド内の引数として値を渡す場合の規則

| データ型         | 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 <b>&amp;</b> 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。 |

表 53. EXEC CICS コマンド内の引数として値を渡す場合の規則 (続き)

| データ型                         | 使用法                      | 引数のコーディング                                                                                                           |
|------------------------------|--------------------------|---------------------------------------------------------------------------------------------------------------------|
| 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LEMENT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<br>(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 <b>&amp;</b> 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<br>(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<br>(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 <b>&amp;</b> 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。

EIB にアクセスしたい場合は、各アプリケーションのはじめに ADDRESS EIB ステートメントをコーディングしなければなりません。RESP または RESP2 オプションを含むコマンドを使用する場合も同じです。

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、外部プロシーチャーを呼び出す 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、687 ページの『言語環境プログラム (Language Environment) 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。考えられる解決方法の 1 つは、z/OS Language Environment カスタマイズのユーザー出口のカスタマイズに関する情報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むことです。

---

## CICS ファウンデーション・クラスの使用

このセクションでは、CICS ファウンデーション・クラスについて、およびそれらの使用方法について説明します。ユーザー・インターフェースの公式なリストについては、ファウンデーション・クラス: リファレンスを参照してください。

### C++ オブジェクト

このセクションでは、オブジェクトを作成、使用、および削除する方法について説明します。

このセクションでは、オブジェクトを作成、使用、および削除する方法について説明します。このコンテキストでは、オブジェクトはクラスのインスタンスです。オブジェクトを、基本クラスまたは抽象基本クラスのインスタンスにすることはできません。本書のリファレンス部分で説明しているすべての具象 (非基本) クラスのオブジェクトを作成できます。

#### オブジェクトの作成

クラスにコンストラクターがある場合は、そのクラスのオブジェクトの作成時にコンストラクターが実行されます。このコンストラクターは一般的に、オブジェクトの状態を初期化します。ファウンデーション・クラスのコンストラクターには多くの場合、プログラマーがオブジェクト作成時に指定する必要のある必須の定位置パラメーターがあります。

C++ オブジェクトは、以下のいずれかの方法で作成できます。

1. オブジェクトが C++ スタック上に作成される場合は、自動。以下に例を示します。

```
{
ClassX objX
ClassY objY(parameter1);
} //objects deleted here
```

ここで、objX および objY が自動的にスタック上に作成されます。オブジェクトの存続期間は、オブジェクトが作成されたコンテキストによって制限されます。オブジェクトがその有効期間を超えると、自動的に削除されます (つまり、デストラクターが実行され、ストレージが解放されます)。

2. オブジェクトが C++ ヒープ上に作成される場合は、動的。以下に例を示します。

```
{
ClassX* pObjX = new ClassX;
ClassY* pObjY = new ClassY(parameter1);
} //objects NOT deleted here
```

ここで、オブジェクトそのものではなく、オブジェクトのポインターを操作します。オブジェクトの存続期間は、それが作成されたときの有効期間を超えます。前のサンプルでは、オブジェクトが有効期間を超えると、ポインター (pObjX および pObjY) は「失われます」が、ポインターが指していたオブジェクトは存在し続けます。オブジェクトは、以下のように明示的に削除しない限り、存在しています。

```
{
ClassX* pObjX = new ClassX;
ClassY* pObjY = new ClassY(parameter1);
:
pObjX->method1();
pObjY->method2();
:
delete pObjX;
delete pObjY;
}
```

本書のほとんどのサンプルでは、自動ストレージが使用されています。オブジェクトを明示的に削除することを覚えている必要がないため、自動ストレージを使用することが推奨されますが、CICS C++ ファウンデーション・クラス・プログラムにはどちらのスタイルを使用しても構いません。ファウンデーション・クラスおよびストレージ管理の詳細については、『650 ページの『ストレージ管理』』を参照してください。

## オブジェクトを使用

クラスの public メソッドは、そのクラスのオブジェクトに対して呼び出すことができます。

クラスの public メソッドは、そのクラスのオブジェクトに対して呼び出すことができます。次の例では、オブジェクト *obj* を作成してから、そのオブジェクトに対してメソッド **doSomething** を呼び出します。

```
ClassY obj("TEMP1234");
obj.doSomething();
```

また、動的オブジェクト作成の使用時に、これを実行することもできます。

```
ClassY* pObj = new ClassY("parameter1");
pObj->doSomething();
```

## オブジェクトの削除

オブジェクトが破棄されたら、そのデストラクター関数 (~ (チルド) が先頭に付いたクラスと同じ名前を持つ) が自動的に呼び出されます。(デストラクターを明示的に呼び出すことはできません。)

オブジェクトが自動的に作成された場合は、有効範囲を超えたときに、自動的に破棄されます。

オブジェクトが動的に作成された場合は、明示的な **delete** 演算子を使用されるまで、存在し続けます。

## ファウンデーション・クラスの概要

このトピックでは、ファウンデーション・クラスで実行できる内容について概説します。

単純な開始方法については、『ICC\$HEL: C++ Hello World サンプル』を参照してください。このセクションでは、CICS C++ ファウンデーション・クラス・ライブラリーについて概説するため、カテゴリーを順に示していきます。

ファウンデーション・クラスの詳細については、『ファウンデーション・クラス・リファレンス』を参照してください。

CICS ファウンデーション・クラスに属するすべてのクラスに接頭部 **Icc** が付きます。

### 基本クラス

すべてのクラスは、**IccBase** から直接または間接的に継承します。

**IccBase**  
**IccRecordIndex**  
**IccResource**  
**IccControl**  
**IccTime**  
**IccResourceId**

図 120. 基本クラス

すべてのリソース識別クラス (**IccTermId**、**IccTransId** など) が、**IccResourceId** クラスから継承します。これらは、通常、CICS テーブル・エントリーです。

すべての CICS リソース (実際には、**IccResource** から継承し、CICS サービスへのアクセスを必要とするすべてのクラス)。

基本クラスにより、クラスのカテゴリーの共通インターフェースを定義できます。これらのインターフェースは、IBM 提供のファウンデーション・クラスを作成するために使用され、アプリケーション・プログラマーが独自で派生させたクラスを作成するために使用できます。

### **IccBase**

他のすべてのファウンデーション・クラスの基本。これにより、メモリ管理が可能になり、オブジェクトのタイプを検出するためにオブジェクトを調べることができます。

### **IccControl**

アプリケーション・プログラムがサブクラスを設定し、**run** メソッドの実装を提供する必要がある、抽象基本クラス。

### **IccResource**

CICS リソースまたはサービスにアクセスするすべてのクラスの基本クラス。608 ページの『リソース・クラス』を参照してください。

### **IccResourceId**

**IccFileId** や **IccTempStoreId** などの、すべてのテーブル・エントリー (リソース名) クラスの基本クラス。

### **IccTime**

時間情報を保管するクラス (**IccAbsTime**、**IccTimeInterval**、および **IccTimeOfDay**) の基本クラス。

## リソース識別クラス

リソース識別クラスは以下のとおりです。

**IccBase**  
**IccResourceId**  
**IccConvId**  
**IccDataQueueId**  
**IccFileId**  
**IccGroupId**  
**IccJournalId**  
**IccJournalTypeId**  
**IccLockId**  
**IccPartnerId**  
**IccProgramId**  
**IccRequestId**  
**IccAlarmRequestId**  
**IccSysId**  
**IccTempStoreId**  
**IccTermId**  
**IccTPNameId**  
**IccTransId**  
**IccUserId**

図 121. リソース識別クラス

CICS リソース識別クラスは、CICS リソース ID (一般的には、RDO リソース定義に指定されているリソースの名前) を定義します。例えば、**IccFileId** オブジェクトは CICS ファイル名を表します。すべての具象リソース識別クラスに、以下のプロパティーが適用されます。

- クラスの名前は **Id** で終わります。

- クラスは、**IccResourceId** クラスのサブクラスです。
- コンストラクターは、指定されたリソース ID が CICS 標準に従っているかどうかを検査します。例えば、**IccFileId** オブジェクトには、1 から 8 バイトの文字フィールドが含まれている必要があります。9 バイトのフィールドを指定しても、許可されません。

リソース識別クラスは、型検査を改善します。パラメーターとして **IccFileId** オブジェクトを予期するメソッドは、代わりに **IccProgramId** オブジェクトが提供されると、それを受け入れません。リソース名を表す文字ストリングが代わりに使用されると、コンパイラーが妥当性を検査できません。つまり、ストリングがファイル名であるか、プログラム名であるかを検査できません。

『『リソース・クラス』』で説明しているリソース・クラスの多くには、リソース識別クラスが含まれています。例えば、**IccFile** オブジェクトには **IccFileId** オブジェクトが含まれています。CICS リソース上で機能させるためには、リソース識別オブジェクトではなく、リソース・オブジェクトを使用する必要があります。例えば、ファイルからレコードを読み取る場合は、**IccFileId** ではなく **IccFile** を使用する必要があります。

| クラス               | CICS リソース         |
|-------------------|-------------------|
| IccAlarmRequestId | アラーム要求            |
| IccConvId         | 会話 (conversation) |
| IccDataQueueId    | 一時データ・キュー         |
| IccFileId         | ファイル              |
| IccGroupId        | グループ              |
| IccJournalId      | ジャーナル             |
| IccJournalTypeId  | ジャーナル・タイプ         |
| IccLockId         | (適用されない)          |
| IccPartnerId      | APPC パートナー定義ファイル  |
| IccProgramId      | プログラム             |
| IccRequestId      | 要求                |
| IccSysId          | リモート・システム         |
| IccTempStoreId    | 一時記憶域キュー          |
| IccTermId         | 端末                |
| IccTPNameId       | リモート APPC TP 名    |
| IccTransId        | トランザクション          |
| IccUserId         | user              |

## リソース・クラス

CICS リソース・クラスはすべて、**IccResource** 基本クラスから継承します。

IccBase  
IccResource  
IccAbendData  
IccClock  
IccConsole  
IccControl  
IccDataQueue  
IccFile  
IccFileIterator  
IccJournal  
IccProgram  
IccSemaphore  
IccSession  
IccStartRequestQ  
IccSystem  
IccTask  
IccTempStore  
IccTerminal  
IccTerminalData  
IccUser

図 122. リソース・クラス

これらのクラスは、以下のように、主要な CICS リソースの動作をモデル化します。

- 端末は **IccTerminal** によってモデル化されます。
- プログラムは **IccProgram** によってモデル化されます。
- 一時記憶域キューは **IccTempStore** によってモデル化されます。
- 一時データ・キューは **IccDataQueue** によってモデル化されます。

CICS リソースに対する操作によって、CICS 条件が発生することがあります。**IccResource** の **condition** メソッド (『IccResource method: condition』ページを参照) がリソースの問い合わせを行うことがあります。

(CICS サービスにアクセスするクラスは、すべて、**IccResource** から派生したものである必要があります)。

| クラス              | CICS リソース               |
|------------------|-------------------------|
| IccAbendData     | タスク異常終了データ              |
| IccClock         | CICS 日時サービス             |
| IccConsole       | CICS コンソール              |
| IccControl       | 実行中のプログラムの制御            |
| IccDataQueue     | 一時データ・キュー               |
| IccFile          | ファイル                    |
| IccFileIterator  | ファイル・イテレーター (ファイルの参照)   |
| IccJournal       | ユーザーまたはシステム・ジャーナル       |
| IccProgram       | プログラム (実行中のプログラムの外部)    |
| IccSemaphore     | セマフォ (サービスのロック)         |
| IccSession       | セッション                   |
| IccStartRequestQ | 開始要求キュー (非同期トランザクション開始) |

| クラス             | CICS リソース              |
|-----------------|------------------------|
| IccSystem       | CICS システム              |
| IccTask         | 現行タスク                  |
| IccTempStore    | 一時記憶域キュー               |
| IccTerminal     | 現行タスクに属する端末            |
| IccTerminalData | <b>IccTerminal</b> の属性 |
| IccTime         | 時間指定                   |
| IccUser         | ユーザー (セキュリティ属性)        |

## サポート・クラス

サポート・クラスは以下のとおりです。

**IccBase**  
**IccBuf**  
**IccEvent**  
**IccException**  
**IccMessage**  
**IccRecordIndex**  
**IccKey**  
**IccRBA**  
**IccRRN**  
**IccResource**  
**IccTime**  
**IccAbsTime**  
**IccTimeInterval**  
**IccTimeOfDay**

図 123. サポート・クラス

これらのクラスは、リソース・クラスを補完するツールです。これによりアプリケーション・プログラマーの負担が軽減され、オブジェクト・モデルに値が追加されます。

### Resource class (リソース・クラス) 説明

|                 |                                      |
|-----------------|--------------------------------------|
| IccAbsTime      | 絶対時刻 (1900 年 1 月 1 日以降のミリ秒)          |
| IccBuf          | データ・バッファー (データ域の操作を容易にする)            |
| IccEvent        | イベント (CICS コマンドの結果)                  |
| IccException    | ファウンデーション・クラス例外 (C++ 例外処理モデルをサポートする) |
| IccTimeInterval | 時間間隔 (例: 5 分)                        |
| IccTimeOfDay    | 時刻 (例: 6 時 5 分)                      |

**IccAbsTime**、**IccTimeInterval**、および **IccTimeOfDay** クラスにより、アプリケーション・プログラマーが、アプリケーション・プログラム内のオブジェクトとして時間測定を簡単に指定できるようになります。**IccTime** は基本クラスです。

**IccAbsTime**、**IccTimeInterval**、および **IccTimeOfDay** は **IccTime** から派生します。

以下のようなシグニチャーを持つ **IccTask** クラスのメソッド **delay** について考えてみます。

```
void delay(const IccTime& time, const IccRequestId*
reqId = 0);
```

1 分 7 秒の遅延 (つまり、時間間隔) を要求するには、アプリケーション・プログラマーは以下を実行します。

```
IccTimeInterval time(0, 1, 7);
task()->delay(time);
```

注: **IccControl** クラスにはタスク・メソッドが用意されており、アプリケーションのタスク・オブジェクトのポインターを返します。

また、12 時 10 分 (ランチタイム?) までの遅延を要求するには、アプリケーション・プログラマーが以下を実行します。

```
IccTimeOfDay lunchtime(12, 10);
task()->delay(lunchtime);
```

**IccBuf** クラスにより、バッファー (ファイル・レコード・バッファー、一時データ・レコード・バッファー、COMMAREA など) を容易に操作できます (**IccBuf** クラスの詳細については、『613 ページの『バッファー・オブジェクト』』を参照してください)。

**IccMessage** クラスは主に **IccException** クラスによって使用され、例外がスローされた理由を示す説明をカプセル化します。また、アプリケーション・プログラマーは **IccMessage** を使用すると、独自のメッセージ・オブジェクトを作成することもできます。

エラーが検出されると、ファウンデーション・クラス内の多くのメソッドから **IccException** オブジェクトがスローされます。

**IccEvent** クラスを使用すると、プログラマーは、特定の CICS イベント (コマンド) に関連する情報にアクセスできます。

## CICS リソースの使用

CICS リソース (ファイル、プログラムなど) を使用するには、まず適切なオブジェクトを作成してから、オブジェクトに対するメソッドを呼び出す必要があります。

リソース・オブジェクトの作成:

リソース・オブジェクトを作成する場合は、実際の CICS リソース (ファイルやプログラムなど) の表現を作成します。CICS リソースを作成するわけではありません。このオブジェクトは、アプリケーションから見たリソースです。オブジェクトの破棄にも同じことが当てはまります。

リソース・オブジェクトを作成するときは、付随するリソース識別オブジェクトを使用します。以下に例を示します。

```
IccFileId id("XYZ123");
IccFile file(id);
```

これにより、以下のような誤った処理が実行されないように C++ コンパイラーで防止できるようになります。

```
IccDataQueueId id("WXYZ");
IccFile file(id); //gives error at compile time
```

オブジェクト作成時に、リソースのテキスト名を代わりに使用することも可能です。

```
IccFile file("XYZ123");
```

*singleton* クラス:

**IccFile** など、多くのリソース・クラスは、単一プログラム内に複数のリソース・オブジェクトを作成するために使用できます。

```
IccFileId id1("File1");
IccFileId id2("File2");
IccFile file1(id1);
IccFile file2(id2);
```

しかし、一部のリソース・クラスは、プログラマーがそのクラスのインスタンスを **1** つ だけ作成できるように設計されており、それらのクラスは *singleton* クラスと呼ばれています。以下のファウンデーション・クラスは *singleton* です。

- **IccAbendData** は、タスクの異常終了に関する情報を提供します。
- **IccConsole** またはその派生クラスは、オペレーター・メッセージ用のシステム・コンソールを表します。
- **IccControl** またはその派生クラス (**IccUserControl** など) は、実行中のプログラムを制御します。
- **IccStartRequestQ** またはその派生クラスは、アプリケーション・プログラムが CICS トランザクション (タスク) を非同期に開始できるようにします。
- **IccSystem** またはその派生クラスは、そのアプリケーションを実行している CICS システムのアプリケーション・ビューです。
- **IccTask** またはその派生クラスは、実行中のプログラムを実行している CICS タスクを表します。
- **IccTerminal** またはその派生クラスは、使用している基本機能が 3270 端末であれば、タスクの端末を表します。

*singleton* クラスの複数のオブジェクトを作成しようとすると、エラーになり、C++ 例外がスローされます。

これらの各 *singleton* クラスには、**instance** というクラス・メソッドが用意されています。これは、要求されたオブジェクトへのポインターを返し、まだそのポインターが存在しない場合は作成します。以下に例を示します。

```
IccControl* pControl = IccControl::instance();
```

リソース・オブジェクトでのメソッドの呼び出し:

`public` メソッドはいずれも、リソース・クラスのオブジェクトで呼び出すことができます。

以下に例を示します。

```
IccTempStoreId id("TEMP1234");
IccTempStore temp(id);
temp.writeItem("Hello TEMP1234");
```

メソッド **writeItem** は、渡されたストリングの内容 ("Hello TEMP1234") を CICS 一時記憶域キュー「TEMP1234」に書き込みます。

## バッファ・オブジェクト

ファウンデーション・クラスは、**IccBuf** オブジェクトを広範囲で使用します。このバッファ・オブジェクトは、データまたはレコードを処理するタスクを単純化します。

本書の以降の説明を読み進めるための前提条件として、これらのオブジェクトの使用方法について理解しておく必要があります。

データを CICS に渡す処理 (例えば、データ・レコードの書き込み) およびデータを CICS から受け取る処理 (例えば、データ・レコードの読み取り) に関する各 CICS リソース・クラスで、**IccBuf** クラスが使用されます。このようなクラスの例は、**IccConsole**、**IccDataQueue**、**IccFile**、**IccFileIterator**、**IccJournal**、**IccProgram**、**IccSession**、**IccStartRequestQ**、**IccTempStore**、および **IccTerminal** です。

### **IccBuf** クラス

本書のリファレンス部分で詳しく説明している **IccBuf** では、データ域の一般的な操作を実行できます。

このクラスは多くの方法で利用できるため、オブジェクトの動作に影響する **IccBuf** コンストラクターがいくつかあります。これから、**IccBuf** オブジェクトの 2 つの重要な属性について説明します。

データ域の所有権:

**IccBuf** には、データ域がオブジェクトの内部と外部のどちらに割り振られているかを示す属性があります。

この属性の値は 'internal' と 'external' のいずれかです。これを問い合わせるには、**dataAreaOwner** メソッドを使用します。

バッファの内部/外部所有権:

**DataAreaOwner** = external である場合は、アプリケーション・プログラマーが、**IccBuf** オブジェクトの基となっているストレージの妥当性を確認する必要があります。ストレージが、オブジェクトに適用された特定のメソッドに対して無効または不適切である場合は、予測不能な結果が生じます。

データ域の拡張性:

この属性は、**IccBuf** オブジェクトが作成されたら、そのオブジェクト内のデータ域の長さを伸ばすことができるかどうかを定義します。

この属性の値は 'fixed' と 'extensible' のいずれかです。これを問い合わせるには、**dataAreaType** メソッドを使用します。

'fixed' であるオブジェクトのデータ域サイズを大きくすることはできないため、**IccBuf** オブジェクトに割り当てられたデータの長さ (例えば、ファイル・レコード) は、データ域の長さを超えてはなりません。データ域の長さを超えると、C++ 例外がスローされます。

注: 定義では、'extensible' バッファは 'internal' でもある 必要があります。

**IccBuf** コンストラクター:

**IccBuf** コンストラクターには、**IccBuf** オブジェクトの作成に使用される形式がいくつかあります。

以下に例を示します。

```
IccBuf buffer;
```

これにより、初期の長さがゼロである 'internal' および 'extensible' データ域が作成されます。データがオブジェクトに割り当てられると、割り当てられているデータが収まるように、データ域長が自動的に延長されます。

```
IccBuf buffer(50);
```

これにより、初期の長さが 50 バイトである 'internal' および 'extensible' データ域が作成されます。データ長は、データがオブジェクトに割り当てられるまで、ゼロのままです。50 バイトのデータがオブジェクトに割り当てられると、データ長とデータ域長の両方が値 50 を返します。50 バイトを超えるデータがオブジェクトに割り当てられると、データが収まるように、データ域長が自動的に (つまり、ユーザーが操作しなくても) 延長されます。

```
IccBuf buffer(50, IccBuf::fixed);
```

これにより、長さが 50 バイトである 'internal' および 'fixed' データ域が作成されます。50 バイトを超えるデータをオブジェクトに割り当てようとする、データが切り捨てられ、例外がスローされて、エラー状態がアプリケーションに通知されます。

```
struct MyRecordStruct
{
 short id;
 short code;
 char data(30);
 char rating;
};
MyRecordStruct myRecord;
IccBuf buffer(sizeof(MyRecordStruct), &myRecord);
```

これにより、`myRecord` という名前の 'external' データ域を使用する **IccBuf** オブジェクトが作成されます。定義上は、'external' データ域は 'fixed' でもあります。データを割り当てるには、**IccBuf** オブジェクトのメソッドを使用するか、または `myRecord` 構造を直接使用します。

```
IccBuf buffer("Hello World");
```

これにより、長さがストリング "Hello World" の長さと等しい 'internal' および 'extensible' データ域が作成されます。このストリングは、オブジェクトのデータ域にコピーされます。この初期データ割り当ては、用意されているいずれかの操作メソッド (**insert**、**cut**、または **replace**) を使用して変更できます。

```
IccBuf buffer("Hello World");
buffer << " out there";
IccBuf buffer2(buffer);
```

ここでは、コピー・コンストラクターによって、最初のバッファとほとんど同じ属性が適用された 2 番目のバッファが作成されます。異なるのは、データ域所有権属性です。2 番目のオブジェクトには必ず、最初のオブジェクト内のデータ域のコピーである 'internal' データ域が含まれます。上記の例では、`buffer2` には "Hello World out there" が含まれ、データ域長とデータ長はいずれも 21 となります。

#### **IccBuf** メソッド:

**IccBuf** オブジェクトを操作するには、用意されている多数のメソッドを使用します。例えば、データをバッファに付加したり、バッファ内のデータを変更したり、バッファからデータを切り取ったり、データをバッファの中央へ挿入したりすることができます。

演算子 **const char\***、**=**、**+=**、**==**、**!=**、および **<<** は、クラス **IccBuf** で多重定義されています。**IccBuf** 属性を照会できるメソッドもあります。詳細については、リファレンス・セクションを参照してください。

#### **IccResource** サブクラスの処理:

**IccResource** サブクラスの処理を示すには、**IccTempstore** クラスを使用して CICS 一時記憶域にキュー項目を書き込むことを考えてみてください。

```
IccTempStore store("TEMP1234");
IccBuf buffer(50);
```

作成される **IccTempStore** オブジェクトは、"TEMP1234" という名前の CICS 一時記憶域キューのアプリケーション・ビューです。作成される **IccBuf** オブジェクトのデータ域は 50 バイトです ('extensible' となる場合もあります)。

```
buffer = "Hello Temporary Storage Queue";
store.writeItem(buffer);
```

文字ストリング "Hello Temporary Storage Queue" がバッファにコピーされます。これが可能なのは、**operator=** メソッドが **IccBuf** クラスで多重定義されているためです。

**IccTempStore** オブジェクトはその **writeItem** メソッドを呼び出し、最初のパラメーターとして **IccBuf** オブジェクトに対する参照を渡します。 **IccBuf** オブジェクトの内容が、CICS 一時記憶域キューに書き込まれます。

ここで、CICS リソースからアプリケーション・プログラムの **IccBuf** オブジェクトヘレコードを読み取るという反対の操作について考えてみましょう。

```
buffer = store.readItem(5);
```

**readItem** メソッドは、CICS 一時記憶域キュー内の 5 番目の項目の内容を読み取り、**IccBuf** 参照としてデータを返します。

C++ コンパイラーは、上記のコード行を 2 つのメソッド呼び出し (**IccTempStore** クラスに定義されている **readItem**、および **IccBuf** クラスに多重定義された **operator=**) に解決します。この 2 番目のメソッドは、返された **IccBuf** 参照の内容を取得し、そのデータをバッファーにコピーします。

ファウンデーション・クラスを使用してレコードを読み書きする上記のスタイルは一般的なスタイルです。最後の例では、上記の例と同様のスタイルを使用してコードを記述する方法を示します。ただし今回は、CICS 一時データ・キューにアクセスします。

```
IccDataQueue queue("DATQ");
IccBuf buffer(50);
buffer = queue.readItem();
buffer << "Some extra data";
queue.writeItem(buffer);
```

**IccDataQueue** オブジェクトの **readItem** メソッドが呼び出され、**IccBuf** に対する参照が返されます。これは、**buffer** オブジェクトに割り当てられます (**IccBuf** クラスで多重定義されている **operator=** メソッドを使用)。文字ストリング "Some extra data" がバッファーに付加されます (**IccBuf** クラスで多重定義されている **operator chevron** « メソッドを使用)。その後、**writeItem** メソッドによって、変更されたこのバッファーが CICS 一時データ・キューに書き込まれます。

以下のセクションのサンプルでは、この構文の例をさらに示し、ファウンデーション・クラスを使用して CICS サービスにアクセスする方法について説明します。

**IccBuf** クラスの詳細については、リファレンス・セクションを参照してください。また、用意されているサンプル **ICC\$BUF** も役立ちます。

## CICS Service の使用

このセクションでは、CICS サービスの使用方法について説明します。サービスについて順に考えてみましょう。

### ファイル制御

ファイル制御クラス **IccFile**、**IccFileId**、**IccKey**、**IccRBA**、および **IccRRN** を使用すると、ファイル内のレコードを読み取り、書き込み、更新、および削除できます。

さらに、**IccFileIterator** クラスを使用すると、ファイル内のすべてのレコードをブラウズできます。

**IccFile** オブジェクトは、ファイルを表すために使用されます。**IccFileId** オブジェクトを使用して、ファイルを名前で識別する方法は便利ですが、必須ではありません。

アプリケーション・プログラムは、個々のレコードの形式でそのデータを読み書きします。読み取りまたは書き込みの各要求は、メソッド呼び出しから出されます。レコードにアクセスするには、ファイルと特定のレコードの両方をプログラムが識別する必要があります。

VSAM (または VSAM に類似した) ファイルは、以下のタイプです。

#### **KSDS**

キー順: 各レコードがキー (レコード内の事前定義位置にあるフィールド) で識別されます。各キーは、ファイル内で固有である必要があります。

ファイル内のレコードの論理順序は、キーによって決まります。物理位置は、VSAM が保持する索引に組み込まれます。

レコードをブラウズすると、それらの論理順序で見つかります。

#### **ESDS**

入力順: 各レコードが相対バイト・アドレス (RBA) で識別されます。

レコードは、最初にファイルにロードされた順序で ESDS に保持されます。新しいレコードは必ず末尾に追加されます。レコードは、削除したり、その長さを変更したりすることはできません。

レコードをブラウズすると、最初に書き込まれた順序で見つかります。

#### **RRDS** ファイル

相対レコード: レコードが固定長スロットに書き込まれます。レコードは、それを保持するスロットの相対レコード番号 (RRN) で識別されます。

レコードの読み取り:

読み取り操作には、2 つのクラスを使用します。操作を実行するための **IccFile** と、ファイル・アクセス・タイプが KSDS、ESDS、RRDS のいずれかに応じて、特定のレコードを識別するための **IccKey**、**IccRBA**、**IccRRN** のいずれかです。

**IccFile** クラスの **readRecord** メソッドは、レコードを読み取ります。

**KSDS** レコードの読み取り:

レコードを読み取る前に、**IccFile** の **registerRecordIndex** メソッドを使って、**IccKey** クラスのオブジェクトとファイルを関連付ける必要があります。

レコードにアクセスするには、**IccKey** オブジェクトに保持されているキーを使用する必要があります。「完全」キーは、物理ファイルのキーの長さと同じ長さの文字ストリングです。各レコードは、それぞれの完全なキーにより個別に特定されます。

また、キーを「総称」にすることもできます。総称キーは完全キーより短く、一連のレコードを検索するために使用されます。**IccKey** クラスには、キーの設定や変更を行うことができるメソッドがあります。

**IccFile** クラスには、**isReadable**、**keyLength**、**keyPosition**、**recordIndex**、および **recordLength** の各メソッドがあり、これらは KSDS レコードの読み取りに役立ちます。

**ESDS** レコードの読み取り:

レコードの先頭にアクセスするには、**IccRBA** オブジェクト内に保持されている相対バイト・アドレス (RBA) を使用する必要があります。

レコードを読み取る前に、**IccFile** の **registerRecordIndex** メソッドを使って、**IccRBA** クラスのオブジェクトとファイルを関連付ける必要があります。

**IccFile** クラスには、**isReadable**、**recordFormat**、**recordIndex**、および **recordLength** の各メソッドがあり、これらは ESDS レコードの読み取りに役立ちます。

**RRDS** レコードの読み取り:

レコードにアクセスするには、**IccRRN** オブジェクト内に保持されている相対レコード番号 (RRN) を使用する必要があります。

レコードを読み取る前に、**IccFile** の **registerRecordIndex** メソッドを使って、**IccRRN** クラスのオブジェクトとファイルを関連付ける必要があります。

**IccFile** クラスには、**isReadable**、**recordFormat**、**recordIndex**、および **recordLength** の各メソッドがあり、これらは RRDS レコードの読み取りに役立ちます。

レコードの書き込み:

レコードの書き込みは、「レコードの追加」とも呼ばれます。

このトピックでは、これまで書き込まれたことがないレコードの書き込みについて説明します。既に存在するレコードの書き込みは、そのレコードがそれまでに「更新」モードになっている場合を除いて許可されません。詳しくは、619 ページの『レコードの更新』を参照してください。

レコードを書き込む前に、**IccFile** の **registerRecordIndex** メソッドを使って、**IccKey**、**IccRBA**、または **IccRRN** クラスのオブジェクトとファイルを関連付ける必要があります。 **IccFile** クラスの **writeRecord** メソッドは、レコードを書き込みます。

書き込み操作には、2 つのクラスを使用します。操作を実行するための **IccFile** と、ファイル・アクセス・タイプが KSDS、ESDS、RRDS のいずれかに応じて、特定のレコードを識別するための **IccKey**、**IccRBA**、**IccRRN** のいずれかです。

書き込むレコードが複数あるときは、データの大量挿入を使用して、書き込み速度を向上させることができます。この大量挿入の開始と終了は、**IccFile** の **beginInsert** および **endInsert** メソッドを呼び出して行います。

### **KSDS** レコードの書き込み:

レコードにアクセスするには、**IccKey** オブジェクトに保持されているキーを使用する必要があります。

「完全」キーは、レコードを一意的に識別する文字ストリングです。各レコードは、それぞれの完全なキーにより個別に特定されます。

**IccFile** クラスの **writeRecord** メソッドは、レコードを書き込みます。

**IccFile** クラスには、**isAddable**、**keyLength**、**keyPosition**、**recordIndex**、**recordLength**、および **registerRecordIndex** の各メソッドがあり、これらは KSDS レコードの書き込みに役立ちます。

### **ESDS** レコードの書き込み:

レコードの先頭にアクセスするには、**IccRBA** オブジェクト内に保持されている相対バイト・アドレス (RBA) を使用する必要があります。

**IccFile** クラスには、**isAddable**、**recordFormat**、**recordIndex**、**recordLength**、および **registerRecordIndex** の各メソッドがあり、これらは ESDS レコードの書き込みに役立ちます。

### **RRDS** レコードの書き込み:

新しい ESDS レコードを追加するには、**writeRecord** メソッドを使用します。

**IccFile** クラスには、**isAddable**、**recordFormat**、**recordIndex**、**recordLength**、および **registerRecordIndex** の各メソッドがあり、これらは RRDS レコードの書き込みに役立ちます。

### レコードの更新:

レコードの更新は、「レコードの再書き込み」とも呼ばれます。

レコードを更新する前に、まず 'update' モードで **readRecord** メソッドを使用して読み取る必要があります。これによりレコードがロックされるため、どのユーザーも変更できなくなります。

レコードを更新するには、**rewriteRecord** メソッドを使用します。**IccFile** オブジェクトは、処理中のレコードを記憶しますが、この情報が再び渡されることはありません。

例については、『コード・フラグメント:「更新用にレコードを読み取る」』を参照してください。

レコードを変更するときに、KSDS ファイル内の基本キーを変更してはなりません。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。

ESDS、RRDS、または固定長の KSDS ファイル内のレコードの長さを、更新時に変更してはなりません。

固定長レコードを含むファイルとして CICS に定義されているファイルの場合は、更新するレコードの長さが、元の長さと同じである必要があります。更新されたレコードの長さは、VSAM に定義されている最大以下である必要があります。

レコードの削除:

ESDS ファイルからレコードを削除することは絶対にできません。

通常レコードの削除:

**IccFile** クラスの **deleteRecord** メソッドは、レコードが「更新」モードの状態であるという効果によってロックされていない場合に、1 つ以上のレコードを削除します。

削除するレコードは、**IccKey** または **IccRRN** オブジェクトによって定義されます。

ロックされたレコードの削除:

**IccFile** クラスの **deleteLockedRecord** メソッドは、**readRecord** メソッドによって「更新」モードの状態にあるという効果により、それまでにロックされているレコードを削除します。

レコードのブラウズ:

ブラウズ、つまりファイルの順次読み取りでは、別のクラス **IccFileIterator** を使用します。

このクラスのオブジェクトは、**IccFile** オブジェクトのほか、**IccKey**、**IccRBA**、**IccRRN** のいずれかのオブジェクトに関連付けられている必要があります。この関連付けを行った後は、その他のオブジェクトをさらに参照することなく、**IccFileIterator** オブジェクトを使用できます。

**readNextRecord** メソッドを使用して順方向へ、または **readPreviousRecord** メソッドを使用して逆方向へ、いずれかのブラウズを実行できます。 **reset** メソッドは、**IccKey** オブジェクトまたは **IccRBA** オブジェクトで指定されたレコードを指すように **IccFileIterator** オブジェクトをリセットします。

ファイルのブラウズの例は、コード・フラグメント「すべてのレコードをキーの昇順にリストする (List all records in assending order of key)」のページに示されています。

ファイル制御の例:

このサンプル・プログラムは、**IccFile** クラスと **IccFileIterator** クラスの使用法を示しています。

このサンプルのソースは、C++ sample programsのファイル ICC\$FIL にあります。ここでは、ソース・ファイルには見られる端末の入出力を省略して、コードを示しています。

```
#include "icceh.hpp"
#include "iccmmain.hpp"
```

最初の 2 行は、ファウンデーション・クラスのヘッダー・ファイルと、アプリケーション・プログラムの稼働環境をセットアップする標準 **main** 関数を組み込んでいます。

```
const char* fileRecords[] =
{
//NAME KEY PHONE USERID
"BACH, J S 003 00-1234 BACH ",
"BEETHOVEN, L 007 00-2244 BEET ",
"CHOPIN, F 004 00-3355 CHOPIN ",
"HANDEL, G F 005 00-4466 HANDEL ",
"MOZART, W A 008 00-5577 WOLFGANG "
};
```

これは、サンプル・プログラムによって使用される複数行のデータを定義します。

```
void IccUserControl::run()
{
```

**IccUserControl** クラスの **run** メソッドに、この例のユーザー・コードが含まれています。端末を使用するため、このサンプルは端末オブジェクトの作成と関連する画面の消去から始まります。

```
 short recordsDeleted = 0;
 IccFileId id("ICCKFILE");
 IccKey key(3,IccKey::generic);
 IccFile file(id);
 file.registerRecordIndex(&key);
 key = "00";
 recordsDeleted = file.deleteRecord();
```

最初に *key* オブジェクトと *file* オブジェクトが作成され、これらを使用して、KSDS ファイル「ICCKFILE」内の「00」で始まるキーを持つレコードがすべて削除されます。 *key* は 3 バイトの総称キーとして定義されますが、このインスタンスでは、先頭 2 バイトのみが使用されます。

```
 IccBuf buffer(40);
 key.setKind(IccKey::complete);
 for (short j = 0; j < 5; j++)
 {
 buffer = fileRecords[j];
 key.assign(3, fileRecords[j]+15);
 file.writeRecord(buffer);
 }
```

この次のフラグメントは、指定されたすべてのデータを、ファイルのレコードに書き込みます。このデータは、この目的のために作成される **IccBuf** オブジェクトによって渡されます。**setKind** メソッドを使用して、*key* を「generic」から「complete」に変更します。

これらの呼び出し間の **for** ループでは、すべてのデータをループしながら、**IccBuf** の **operator=** メソッドを使用してバッファにデータを渡してから、**writeRecord** を使用してファイル内のレコードに渡します。途中で、**assign** を使用して、各レコードのキーがデータ内で出現する文字ストリング (15 文字目から 3 文字) になるよ

うに設定します。

```
IccFileIterator fIterator(&file,
 &key);
key = "000";
buffer = fIterator.readNextRecord();
while (fIterator.condition() == IccCondition::NORMAL)
{
 term->sendLine("- record read: [%s]",(const char*) buffer);
 buffer = fIterator.readNextRecord();
}
```

ここに示すループは、**sendLine** を使用して、すべてのレコードをキーの昇順で端末にリストします。ここでは **IccFileIterator** オブジェクトを使用してレコードをブラウズします。この例にはありませんが、キーの最小値が設定された場合は、CICS に依存して、キー・シーケンスで最初のレコードを検索して、このループが開始されます。

ループは、NORMAL 以外の条件が返されるまで続行されます。

```
key = "¥xFF¥xFF¥xFF";
fIterator.reset(&key);
buffer = fIterator.readPreviousRecord();
while (fIterator.condition() == IccCondition::NORMAL)
{
 buffer = fIterator.readPreviousRecord();
}
```

次のループは最後のループとほぼ同じですが、レコードがキーの逆順でリストされます。

```
key = "008";
buffer = file.readRecord(IccFile::update);
buffer.replace(4, "5678", 23);
file.rewriteRecord(buffer);
```

このフラグメントは、更新のためにレコードを読み取り、他のプログラムが変更できないようにそのレコードをロックします。次に、バッファ内のレコードを変更し、更新したレコードをファイルに書き込みます。

```
buffer = file.readRecord();
```

実際に更新したことを示すため、同じレコードを再び読み取り、端末に送信します。

```
return;
}
```

**run** が終了し、制御が CICS に戻ります。

このサンプルからの予期される出力については、C++ sample programsを参照してください。

## プログラム制御

このセクションでは、現在実行されているプログラム以外のプログラムにアクセスし、それを使用する方法について説明します。

プログラム制御では、リソース・クラスの 1 つである **IccProgram** クラスを使用します。

プログラムは、**IccProgram** オブジェクトを使用して、ロード、アンロード、およびリンクできます。**IccProgram** オブジェクトを問い合わせると、プログラムに関する情報を取得できます。詳しくは、IccProgram クラスを参照してください。

以下に示す例では、1 つのプログラムがほかの 2 つのプログラムを順に呼び出します。これらの間でデータは COMMAREA 経由で受け渡されます。1 つのプログラムがローカルであることが想定され、2 番目のプログラムはリモート CICS システム上にあります。プログラムは 2 つのファイル ICC\$PRG1 および ICC\$PRG2 に含まれています。これらのファイルの位置と、これらのサンプル・プログラムから予想される出力については、C++ sample programsを参照してください。

これらのサンプル内の端末入出力のほとんどが、以下のコードでは省略されています。

```
#include "icceh.hpp"
#include "iccmmain.hpp"
void IccUserControl::run()
{
```

両方のプログラムのコードが、まずファウンデーション・クラスのヘッダー・ファイルと **main** メソッドのスタブから始まります。ユーザー・コードは、各プログラムの **IccUserControl** クラスの **run** メソッド内にあります。

```
IccSysId sysId("ICC2");
IccProgram icc$prg2("ICC$PRG2");
IccProgram remoteProg("ICC$PRG3");
IccBuf commArea(100, IccBuf::fixed);
```

最初のプログラム (ICC\$PRG1) では、リモート領域を表す **IccSysId** オブジェクト、およびこのプログラムから呼び出されるローカル・プログラムとリモート・プログラムを表す 2 つの **IccProgram** オブジェクトが作成されます。100 バイトの固定長バッファ・オブジェクトも作成され、プログラム間の通信領域として使用されます。

```
icc$prg2.load();
if (icc$prg2.condition() == IccCondition::NORMAL)
{
 term->sendLine("Loaded program: %s <%s> Length=%ld Address=%x",
 icc$prg2.name(),
 icc$prg2.conditionText(),
 icc$prg2.length(),
 icc$prg2.address());
 icc$prg2.unload();
}
```

その後プログラムは、プログラム ICC\$PRG2 のプロパティをロードし問い合わせようとしています。

```
commArea = "DATA SET BY ICC$PRG1";
icc$prg2.link(&commArea);
```

通信域バッファは、ICC\$PRG1 のリンク先の最初のプログラム (ICC\$PRG2) に渡されるデータを格納するように設定されます。ICC\$PRG1 は、ICC\$PRG2 の実行中には中断状態になります。

呼び出されるプログラム ICC\$PRG2 は単純なプログラムで、その骨子は以下のとおりです。

```
IccBuf& commArea = IccControl::commArea();
commArea = "DATA RETURNED BY ICC$PRG2";
return;
```

ICC\$PRG2 は、渡された通信域へのアクセスを取得します。その後、この通信域のデータを変更し、呼び出し側のプログラムに制御を返します。

この時点で最初のプログラム (ICC\$PRG1) が、以下のように別のシステムで別のプログラムを呼び出します。

```
remoteProg.setRouteOption(sysId);
commArea = "DATA SET BY ICC$PRG1";
remoteProg.link(&commArea);
```

**setRouteOption** は、このオブジェクトの呼び出しがリモート・システムにルーティングされるように要求します。通信域が再設定され (ICC\$PRG2 によって変更されているため)、リモート・プログラム (システム ICC2 上の ICC\$PRG3) にリンクされます。

呼び出されたプログラムは、CICS 一時記憶域を使用しますが、以下の 3 行について考えてみましょう。

```
IccBuf& commArea = IccControl::commArea();
commArea = "DATA RETURNED BY ICC$PRG3";
return;
```

リモート・プログラム (ICC\$PRG3) が再度、渡された通信域へのアクセスを取得します。この通信域のデータを変更し、呼び出し側のプログラムに制御を返します。

```
return;
};
```

最後に、呼び出し側プログラムそのものが終了し、制御を CICS に返します。

## トランザクションの非同期の開始

**IccStartRequestQ** クラスを使用すると、プログラムは別の CICS トランザクション・インスタンスを非同期で開始できます (さらに、オプションでデータを開始済みトランザクションに渡すことができます)。

開始済みトランザクションでは同じクラスを使用して、開始要求を発行したタスクから渡されたデータへのアクセスを取得します。最後に、開始要求を後から取り消すことができます。

### トランザクションの開始:

以下のいずれかのメソッドを使用して、どのデータを、開始されるトランザクションへ送信するかを設定できます。

- **registerData** または **setData**
- **setQueueName**
- **setReturnTermId**
- **setReturnTransId**

実際の開始は、**start** メソッドを使用して要求します。

開始データへのアクセス:

開始されるトランザクションは、**retrieveData** メソッドを呼び出すことにより、自己の開始データにアクセスできます。

このメソッドはすべての開始データ属性を **IccStartRequestQ** オブジェクトに保管します。個々の属性には、以下のメソッドを使用してアクセスできます。

- **data**
- **queueName**
- **returnTermId**
- **returnTransId**

満了していない開始要求のキャンセル:

満了していない開始要求 (つまり、まだ到達していない将来の時点の開始要求) は、**cancel** メソッドを使用してキャンセルすることができます。

トランザクション開始の例:

システム ICC1 の端末 PEO1 でトランザクション ISR1 を開始します。

| CICS システム | ICC1               | ICC2      |
|-----------|--------------------|-----------|
| トランザクション  | ISR1/ITMP          | ISR2      |
| プログラム     | ICC\$SRQ1/ICC\$TMP | ICC\$SRQ2 |
| 端末        | PEO1               | PEO2      |

これは、2 つの開始要求を発行し、最初の要求は満了前に取り消されます。2 番目の要求は、システム ICC2 の端末 PEO2 でトランザクション ISR2 を開始します。このトランザクションは、その開始データにアクセスし、元の端末 (システム ICC1 の PEO1) でトランザクション ITMP を開始して、トランザクションが終了します。

プログラムとその予期される出力は、C++ sample programsに、ファイル ICC\$SRQ1 および ICC\$SRQ2 として用意されています。ここでは、端末の入出力要求を省略してコードを示しています。

トランザクション ISR1 は、システム ICC1 でプログラム ICC\$SRQ1 を実行します。最初にこのプログラムについて考えてみます。

```
#include "icceh.hpp"
#include "iccmmain.hpp"
void IccUserControl::run()
{
```

これらの行では、ファウンデーション・クラスのヘッダー・ファイルと、アプリケーション・プログラムのクラス・ライブラリーをセットアップするために必要な **main** 関数を組み込んでいます。**IccUserControl** クラスの **run** メソッドに、この例のユーザー・コードが含まれています。

```
IccRequestId req1;
IccRequestId req2("REQUEST1");
IccTimeInterval ti(0,0,5);
IccTermId remoteTermId("PE02");
IccTransId ISR2("ISR2");
IccTransId ITMP("ITMP");
IccBuf buffer;
IccStartRequestQ* startQ = startRequestQ();
```

ここでは、以下のようないくつかのオブジェクトを作成しています。

**req1** 特定の開始要求を識別するために作動可能な、空の **IccRequestId** オブジェクト。

**req2** ユーザーが提供する ID「REQUEST1」を格納する **IccRequestId** オブジェクト。

**ti** 0 時 0 分 5 秒を表す **IccTimeInterval** オブジェクト。

**remoteTermId**

**IccTermId** オブジェクト。トランザクションを開始するリモート・システムの端末を表します。

**ISR2** **IccTransId** オブジェクト。リモート・システムで開始するトランザクションを表します。

**ITMP**

**IccTransId** オブジェクト。開始されたトランザクションが、このプログラムの端末で開始するトランザクションを表します。

バッファ

開始データを保持する **IccBuf** オブジェクト。

最後に、**IccControl** クラスの **startRequestQ** メソッドが、単一インスタンス (singleton) クラス **IccStartRequestQ** にポインターを返します。

```
startQ->setRouteOption("ICC2");
startQ->registerData(&buffer);
startQ->setReturnTermId(terminal()->name());
startQ->setReturnTransId(ITMP);
startQ->setQueueName("startqnm");
```

このコード・フラグメントでは、開始要求を発行するときに渡される開始データを作成します。**setRouteOption** は、リモート・システム **ICC2** で開始要求を発行することを指示します。**registerData** メソッドは、開始データを含む **IccBuf** オブジェクトを関連付けます (**IccBuf** オブジェクトの内容は、開始要求を発行するまで抽出されません)。**setReturnTermId** メソッドと **setReturnTransId** メソッドにより、開始要求側がトランザクションと端末名を、開始されるトランザクションに渡すこ

とができます。開始されるトランザクションが別の端末、この例の場合は発信元の端末で、指定されたとおりに 別の トランザクションを開始できるよう、これらのフィールドが一般的に使用されます。

**setQueueName** は、開始されるトランザクションに渡すことができる、もう 1 つの情報です。

```
buffer = "This is a greeting from program
'icc$srq1'!!";
req1 = startQ->start(ISR2, &remoteTermId, &ti);
startQ->cancel(req1);
```

ここで、開始要求で渡すデータを設定します。時間間隔 *ti* (5 秒) の経過後、トランザクション **ISR2** を開始します。要求 ID は、*req1* に保管されます。5 秒経過する前に (つまり、即時に) 開始要求を取り消します。

```
req1 = startQ->start(ISR2, &remoteTermID,
&ti, &req2);
return;
}
```

時間間隔 *ti* (5 秒) の経過後、トランザクション **ISR2** を再度開始します。今度はその要求を満了させるため、トランザクション **ISR2** はリモート・システムで開始されます。それと同時に、**CICS** に制御を返して終了します。

次に、開始されるプログラム **ICC\$SRQ2** について考えてみます。

```
IccBuf buffer;
IccRequestId req("REQUESTX");
IccTimeInterval ti(0,0,5);
IccStartRequestQ* startQ = startRequestQ();
```

ここでは、**ICC\$SRQ1** の場合と同様に、いくつかのオブジェクトを作成します。

**バッファ**

呼び出し元 (**ICC\$SRQ1**) から渡された開始データを保持する **IccBuf** オブジェクト。

**req** 呼び出し元の端末で発行する開始を識別する **IccRequestId** オブジェクト。

**ti** 0 時 0 分 5 秒を表す **IccTimeInterval** オブジェクト。

**IccControl** クラスの **startRequestQ** メソッドは、**singleton** クラス **IccStartRequestQ** を指すポインターを返します。

```
if (task()->startType() != IccTask::startRequest)
{
term->sendLine(
"This program should only be started via the StartRequestQ");
task()->abend("OOPS");
}
```

ここでは、**IccTask** クラスの **startType** メソッドを使用して、**ICC\$SRQ2** が **start** メソッドによって開始されていて、他の方法 (端末でのトランザクション名の入力など) では開始されていないことを確認します。意図したとおりに開始されていない場合は、異常終了し、「OOPS」という異常終了コードが返されます。

```
startQ->retrieveData();
```

ICC\$SRQ1 によって渡された開始データを取得し、以降のアクセス用に **IccStartRequestQ** オブジェクト内に保管します。

```
buffer = startQ->data();
term->sendLine("Start buffer contents = [%s]", buffer.dataArea());
term->sendLine("Start queue= [%s]", startQ->queueName());
term->sendLine("Start rtn = [%s]",
startQ->returnTransId().name());
term->sendLine("Start rtrm = [%s]", startQ->returnTermId().name());
```

この開始データ・バッファは、**IccBuf** オブジェクトにコピーされます。その他の開始データ項目 (キュー、**returnTransId**、および **returnTermId**) は端末に表示されます。

```
task()->delay(ti);
```

5 秒間遅延します (つまり、スリープして何も実行しません)。

```
startQ->setRouteOption("ICC1");
```

**setRouteOption** は、呼び出し元のシステム (ICC1) での開始をシグナル通知します。

```
startQ->start(
startQ->returnTransId(),startQ->returnTermId());
return;
```

ITMP (ICC\$SRQ1 によって **returnTransId** 開始情報で渡されたトランザクションの名前) というトランザクションを発信元の (このトランザクションを開始したときに ICC\$SRQ1 が完了した) 端末で開始します。開始要求を発行したら、ICC\$SRQ1 は、制御を CICS に戻して、終了します。

最後に、トランザクション ITMP が最初の端末で実行されます。これで、トランザクションを非同期で開始する、このデモンストレーションは終了です。

## 一時データ

一時データ・クラス **IccDataQueue** および **IccDataQueueId** を使用すると、データを後続の処理のために一時データ・キューに保管できます。

以下のことができます。

- 一時データ・キューからデータを読み取る (**readItem** メソッド)
- 一時データ・キューにデータを書き込む (**writeItem** メソッド)
- 一時データ・キューを削除する (**empty** メソッド)

**IccDataQueue** オブジェクトは、一時記憶域キューを表すために使用します。

**IccDataQueueId** オブジェクトは、キューを名前で識別するために使用します。

**IccDataQueueId** オブジェクトが初期化されたら、キューの名前を使用する代わりに、このオブジェクトを使用してキューを識別できます。これにより、C++ コンパイラでさらにエラーを検出できます。

**IccDataQueue** クラスで使えるメソッドは、**IccTempStore** クラスのメソッドに類似しています。これらの詳細については、『 630 ページの『一時記憶』』を参照してください。

データの読み取り:

キューから項目を読み取るには、**readItem** メソッドを使用します。

このメソッドは、情報を含んでいる **IccBuf** オブジェクトへの参照を返します。

データの書き込み:

**IccDataQueue** の **writeItem** メソッドは、データの新しい項目をキューに追加し、指定されたバッファからデータを取得します。

キューの削除:

**empty** メソッドは、キューに入っているすべての項目を削除します。

一時データ管理の例:

このサンプル・プログラムは、**IccDataQueue** クラスと **IccDataQueueId** クラスの使用法を示しています。

このプログラムのソースと、このプログラムからの予期される出力は、ファイル **ICC\$DAT** として C++ sample programs にあります。ここでは、端末の入出力要求を省略してコードを示しています。

```
#include "icceh.hpp"
#include "iccmmain.hpp"
```

最初の 2 行は、ファウンデーション・クラスのヘッダー・ファイルと、アプリケーション・プログラムの稼働環境をセットアップする標準 **main** 関数を組み込んでいます。

```
const char* queueItems[] =
{
 "Hello World - item 1",
 "Hello World - item 2",
 "Hello World - item 3"
};
```

これは、サンプル・プログラム用のバッファを定義しています。

```
void IccUserControl::run()
{
```

**IccUserControl** クラスの **run** メソッドに、この例のユーザー・コードが含まれています。

```
 short itemNum =1;
 IccBuf buffer(50);
 IccDataQueueId id("ICCQ");
 IccDataQueue queue(id);
 queue.empty();
```

このフラグメントは最初に、「ICCQ」を含む、IccDataQueueId のタイプの識別オブジェクトを作成します。次に、一時データ・キュー「ICCQ」を表す **IccDataQueue** オブジェクトを作成し、データを空にします。

```
for (short i=0 ; i<3 ; i++)
{
buffer = queueItems[i];
queue.writeItem(buffer);
}
```

このループでは、3 つのデータ項目を一時データ・オブジェクトに書き込みます。このデータは、この目的のために作成された **IccBuf** オブジェクトによって渡されます。

```
buffer = queue.readItem();
while (queue.condition() == IccCondition::NORMAL)
{
buffer = queue.readItem();
}
```

3 つのレコードの書き出しが完了したら、今度はこれらを読み戻し、正常に書き込まれたことを示します。

```
return;
}
```

**run** が終了し、制御が **CICS** に戻ります。

### 一時記憶

一時記憶域クラス **IccTempStore** および **IccTempStoreId** を使用すると、データを一時記憶域キューに保管できます。

以下のことができます。

- 一時記憶域キューから項目を読み取る (**readItem** メソッド)
- 一時記憶域キューの終わりに新規項目を書き込む (**writeItem** メソッド)
- 一時記憶域キュー内の項目を更新する (**rewriteItem** メソッド)
- 一時記憶域キュー内の次の項目を読み取る (**readNextItem** メソッド)
- 一時データをすべて削除する (**empty** メソッド)

**IccTempStore** オブジェクトは、一時記憶域キューを表すために使用します。

**IccTempStoreId** オブジェクトは、キューを名前で識別するために使用します。

**IccTempStoreId** オブジェクトが初期化されたら、キューの名前を使用する代わりに、このオブジェクトを使用してキューを識別できます。これにより、C++ コンパイラでさらにエラーを検出できます。

**IccTempStore** クラスで使えるメソッドは、**IccDataQueue** クラスのメソッドに類似しています。これらの詳細については、『628 ページの『一時データ』』を参照してください。

項目の読み取り:

**IccTempStore** の **readItem** メソッドは、指定された項目を一時記憶域キューから読み取ります。

このメソッドは、情報を含んでいる **IccBuf** オブジェクトへの参照を返します。

項目の書き込み:

項目の書き込みは、項目の「追加」とも呼ばれます。

このセクションでは、これまで書き込まれたことのない項目の書き込みについて説明します。**rewriteItem** メソッドを使用して、既に存在する項目の書き込みを行うことができます。詳しくは、『項目の更新』を参照してください。

**IccTempStore** の **writeItem** メソッドは、新しい項目をキューの末尾に追加し、指定されたバッファーからデータを取得します。これが正常に実行された場合は、追加されたレコードの項目番号が返されます。

項目の更新:

項目の更新は、項目の「再書き込み」とも呼ばれます。

一時記憶域キュー内の指定した項目を更新するには、**IccTempStore** クラスの **rewriteItem** メソッドを使用します。

項目の削除:

一時記憶域キュー内の個々の項目を削除することはできません。

**IccTempStore** オブジェクトに関連付けられているすべての一時データを削除するには、**IccTempStore** クラスの **empty** メソッドを使用します。

一時記憶域の例:

このサンプル・プログラムは、**IccTempStore** クラスと **IccTempStoreId** クラスの使用法を示しています。

このプログラムと、このプログラムからの予期される出力は、ファイル **ICC\$TMP** として **C++ sample programs** にあります。このサンプルは、ここでは、端末の入出力要求を省略して示しています。

```
#include "icceh.hpp"
#include "iccmmain.hpp"
#include <stdlib.h>
```

最初の 3 行は、ファウンデーション・クラスのヘッダー・ファイル、アプリケーション・プログラムの稼働環境をセットアップする標準 **main** 関数、および標準ライブラリーを組み込んでいます。

```
const char* bufferItems[] =
{
 "Hello World - item 1",
 "Hello World - item 2",
 "Hello World - item 3"
};
```

これは、サンプル・プログラム用のバッファーを定義しています。

```
void IccUserControl::run()
{
```

**IccUserControl** クラスの **run** メソッドに、この例のユーザー・コードが含まれています。

```
 short itemNum = 1;
 IccTempStoreId id("ICCSTORE");
 IccTempStore store(id);
 IccBuf buffer(50);
 store.empty();
```

このフラグメントは最初に、フィールド「ICCSTORE」を含む識別オブジェクト **IccTempStoreId** 作成します。次に、一時記憶域キュー「ICCSTORE」を表す **IccTempStore** オブジェクトを作成し、レコードを空にします。

```
 for (short j=1 ; j <= 3 ; j++)
 {
 buffer = bufferItems[j-1];
 store.writeItem(buffer);
 }
```

このループでは、3 つのデータ項目を一時記憶域オブジェクトに書き込みます。このデータは、この目的のために作成された **IccBuf** オブジェクトによって渡されます。

```
 buffer = store.readItem(itemNum);
 while (store.condition() == IccCondition::NORMAL)
 {
 buffer.insert(9, "Modified ");
 store.rewriteItem(itemNum, buffer);
 itemNum++;
 buffer = store.readItem(itemNum);
 }
```

この次のフラグメントでは、項目を読み戻して変更し、一時記憶域キューに再書き込みします。最初に、**readItem** メソッドを使用して、一時記憶域オブジェクトからバッファータを読み取ります。**IccBuf** クラスの **insert** メソッドを使用してバッファータ・オブジェクトのデータが変更された後、**rewriteItem** メソッドによってバッファータが上書きされます。ループでは、次のバッファータ項目の読み取りが続行されます。

```
 itemNum = 1;
 buffer = store.readItem(itemNum);
 while (store.condition() == IccCondition::NORMAL)
 {
 term->sendLine(" - record #%d = [%s]", itemNum,
 (const char*)buffer);
 buffer = store.readNextItem();
 }
```

このループでは、一時記憶域キュー項目を再び読み取り、これらの項目が更新されたことを表示します。

```
 return;
}
```

**run** が終了し、制御が CICS に戻ります。

## 端末管理

端末管理クラス **IccTerminal**、**IccTermId**、および **IccTerminalData** を使用すると、CICS タスクに属する端末へのデータの送信、端末からのデータの受信、および端末に関する情報の検索を実行できます。

**IccTerminal** オブジェクトは、CICS タスクに属する端末を表すために使用されます。トランザクションの基本装置が 3270 端末である場合にのみ、このオブジェクトを作成できます。**IccTermId** クラスは、端末を識別するために使用します。

**IccTerminal** が所有する **IccTerminalData** には、端末特性に関する情報が含まれます。

端末へのデータの送信:

**IccTerminal** クラスの **send** および **sendLine** メソッドは、画面へのデータの書き込みに使用されます。

**set...** メソッドを使用して、これを実行できます。また、**erase** メソッドを使用して現在端末で表示されているデータを消去し、**freeKeyboard** メソッドを使用してキーボードを解放し、入力を受け取れるように準備します。

端末からのデータの受信:

**IccTerminal** クラスの **receive** メソッドと **receive3270data** メソッドは、端末からデータを受信するために使用されます。

端末に関する情報の検索:

端末の特性と端末の現在の状態に関する情報を検索できます。

**data** オブジェクトは、端末の特性に関する情報を含む **IccTerminalData** オブジェクトを指し示します。**IccTerminalData** 内のメソッドを使用すると、例えば、画面の縦の長さや、端末が消去書き込み代替をサポートしているかどうかなどを検出できます。**IccTerminal** のメソッドの中には、ある画面が保持する行数などの特性に関する情報を取得するメソッドがあります。

その他に、端末の現在の状態に関する情報を取得できるメソッドもあります。これらのメソッドには、現在の行番号を返す **line** や、現行カーソル位置を返す **cursor** があります。

端末管理の例:

このサンプル・プログラムは、**IccTerminal**、**IccTermId**、および **IccTerminalData** の各クラスの使用法を示しています。

このプログラムと、このプログラムからの予期される出力は、ファイル **ICC\$TRM** として C++ sample programs にあります。

```
#include "icceh.hpp"
#include "iccmmain.hpp"
```

最初の 2 行は、ファウンデーション・クラスのヘッダー・ファイルと、アプリケーション・プログラムの稼働環境をセットアップする標準 **main** 関数を組み込んでい

ます。

```
void IccUserControl::run()
{
IccTerminal& term = *terminal();
term.erase();
```

**IccUserControl** クラスの **run** メソッドに、この例のユーザー・コードが含まれています。端末を使用するため、このサンプルは端末オブジェクトの作成と関連する画面の消去から始まります。

```
term.sendLine("First part of the line...");
term.send("... a continuation of the line.");
term.sendLine("Start this on the next line");
term.sendLine(40, "Send this to column 40 of current line");
term.send(5, 10, "Send this to row 5, column 10");
term.send(6, 40, "Send this to row 6, column 40");
```

このフラグメントは、**send** および **sendLine** メソッドを使用して、データを端末に送信する方法を示します。これらのメソッドはすべて、ストリング・リテラル (`const char*`) の代わりに、**IccBuf** 参照 (`const IccBuf&`) を取ることができます。

```
term.setNewLine();
```

これで、画面にブランク行が送信されます。

```
term.setColor(IccTerminal::red);
term.sendLine("A Red line of text.");
term.setColor(IccTerminal::blue);
term.setHighlight(IccTerminal::reverse);
term.sendLine("A Blue, Reverse video line of text.");
```

**setColor** メソッドは、画面のテキストの色を設定するため、**setHighlight** メソッドは、強調表示を設定するために使用されます。

```
term << "A cout style interface... " <<
endl;
term << "you can " << "chain input together; "
<< "use different types, eg numbers: " << (short)123 <<
" "
<< (long)4567890 << " " << (double)123456.7891234
<< endl;
term << "... and everything is buffered till you issue a flush."
<< flush;
```

このフラグメントは、**iostream** に似たインターフェース **endl** を使用して、次の行でデータを開始する方法を示します。パフォーマンスを高めるために、画面にデータを送信する **flush** が発行されるまで、データを端末のバッファに入れることができます。

```
term.send(24,1, "Program 'icc$trm' complete: Hit PF12
to End");
term.waitForAID(IccTerminal::PF12);
term.erase();
```

**waitForAID** メソッドは、**erase** メソッドを呼び出して表示をクリアする前に、指定されたキーが押されるまで端末を待機させます。

```

return;
}

```

**run** が終了し、制御が CICS に戻ります。

## 日時サービス

**IccClock** クラスは、CICS の日時サービスへのアクセスを制御します。

**IccAbsTime** は、絶対時刻 (1900 年の年初から経過したミリ秒単位の時間) に関する情報を保持します。これは、他の形式に日時に変換できます。**IccClock** オブジェクトと **IccAbsTime** オブジェクトで使用可能なメソッドは、非常によく似ています。

日時サービスの例:

このサンプル・プログラムは、**IccClock** クラスの使用法を示しています。

このプログラムのソースと、このプログラムからの予期される出力は、ファイル **ICC\$CLK** として C++ sample programs にあります。このサンプルは、ここでは、端末の入出力要求を省略して示しています。

```

#include "icceh.hpp"
#include "iccmmain.hpp"
void IccUserControl::run()
{

```

最初の 2 行は、ファウンデーション・クラスのヘッダー・ファイルと、アプリケーション・プログラムの稼働環境をセットアップする標準 **main** 関数を組み込んでいます。

**IccUserControl** クラスの **run** メソッドに、この例のユーザー・コードが含まれています。

```

IccClock clock;

```

これで、**clock** オブジェクトが作成されます。

```

term->sendLine("date() = [%s]",
clock.date());
term->sendLine("date(DDMMYY) = [%s]",
clock.date(IccClock::DDMMYY));
term->sendLine("date(DDMMYY,':') = [%s]",
clock.date(IccClock::DDMMYY,':'));
term->sendLine("date(MMDDYY) = [%s]",
clock.date(IccClock::MMDDYY));
term->sendLine("date(YYDDD) = [%s]",
clock.date(IccClock::YYDDD));

```

ここでは、**date** メソッドを使用して、*format* 列挙型で指定されている形式で日付を返します。形式は順に、システム、DDMMYY、DD:MM:YY、MMDDYY、および YYDDD です。フィールド区切りに使用する文字は、*dateSeparator* 文字で指定されます (指定されていない場合、デフォルトは何も設定されません)。

```

term->sendLine("daysSince1900() = %ld",
clock.daysSince1900());
term->sendLine("dayOfWeek() = %d",
clock.dayOfWeek());
if (clock.dayOfWeek() == IccClock::Friday)
term->sendLine(40, "Today IS Friday");
else
term->sendLine(40, "Today is NOT Friday");

```

このフラグメントは、**daysSince1900** および **dayOfWeek** メソッドの使用を示します。**dayOfWeek** は、曜日を示す列挙型を返します。その日が金曜日であれば、画面に「本日は金曜日です (Today IS Friday)」というメッセージが送信され、それ以外の場合は、「本日は金曜日ではありません (Today is NOT Friday)」のメッセージが送信されます。

```

term->sendLine("dayOfMonth() = %d",
clock.dayOfMonth());
term->sendLine("monthOfYear() = %d",
clock.monthOfYear());

```

これは、**IccClock** クラスの **dayOfMonth** メソッドと **monthOfYear** メソッドを示しています。

```

term->sendLine("time() = [%s]",
clock.time());
term->sendLine("time('-') = [%s]",
clock.time('-'));
term->sendLine("year() = [%ld]",
clock.year());

```

現在時刻は端末へ送信されますが、最初は分離文字を付けずに (つまり、HHMMSS 形式で) 送信され、次に桁を分離する「-」を付けて (つまり、HH-MM-SS 形式で) 送信されます。年が送信されます (例えば、1996)。

```

return;
};

```

**run** が終了し、制御が CICS に戻ります。

## コンパイル、実行、およびデバッグ

このセクションでは、CICS ファウンデーション・クラス・プログラムをコンパイル、実行、およびデバッグする方法について説明します。

### CICS ファウンデーション・クラス・プログラムのコンパイル

CICS ファウンデーション・クラス・プログラムをコンパイルし、リンクするには、プログラムのソース、コンパイラ、ヘッダー・ファイル、およびダイナミック・リンク・ライブラリーにアクセスする必要があります。

以下のアイテムにアクセスする必要があります。

- コンパイルするプログラムのソース

以下に示すように、ご使用の C++ プログラムのソース・コードには、ファウンデーション・クラスのヘッダーとファウンデーション・クラスの **main()** プログラム・スタブに、**#include** ステートメントが必要です。

```
#include "icceh.hpp"
#include "iccmmain.hpp"
```

- IBM C++ コンパイラー
- ファウンデーション・クラスのヘッダー・ファイル (『ヘッダー・ファイル』を参照)
- ファウンデーション・クラスのダイナミック・リンク・ライブラリー (DLL)。ICCFCDLL モジュールは、CICSTS55.CICS.SDFHLOAD にあります。

ファウンデーション・クラスを使用するときは、コンパイルの前に「EXEC CICS」API を変換する必要はないことに注意してください。

以下の JOB ステートメントのサンプルでは、ICC\$HEL という名前のプログラムをコンパイル、プリリンク、およびリンクする方法を示しています。

```
//ICC$HEL JOB 1,user_name,MSGCLASS=A,CLASS=A,NOTIFY=userid
//PROCLIB JCLLIB ORDER=(
CICSTS55.CICS
.SDFHPROC)
//ICC$HEL EXEC ICCFCCL,INFILE=
indatasetname
(ICC$HEL),OUTFILE=
outdatasetname
(ICC$HEL)
//
```

ヘッダー・ファイル:

ヘッダー・ファイルは、CICS C++ ファウンデーション・クラス・プログラムをコンパイルするために必要な C++ クラス定義です。

| C++ ヘッダー・ファイル | このヘッダーに定義されているクラス                               |
|---------------|-------------------------------------------------|
| ICCABDEH      | IccAbendData                                    |
| ICCBASEH      | IccBase                                         |
| ICCBUFEH      | IccBuf                                          |
| ICCCLKEH      | IccClock                                        |
| ICCCNDEH      | IccCondition (struct)                           |
| ICCCONEH      | IccConsole                                      |
| ICCCTLEH      | IccControl                                      |
| ICCDATEH      | IccDataQueue                                    |
| ICCEH         | 1 (638 ページ) を参照してください。                          |
| ICCEVTEH      | IccEvent                                        |
| ICCEXCEH      | IccException                                    |
| ICCFIEH       | IccFile                                         |
| ICCFLIEH      | IccFileIterator                                 |
| ICCGLBEH      | Icc (struct) (グローバル関数)                          |
| ICJRNEH       | IccJournal                                      |
| ICCMSGEH      | IccMessage                                      |
| ICCPRGEH      | IccProgram                                      |
| ICCRECEH      | IccRecordIndex、IccKey、IccRBA、および IccRRN         |
| ICCRESEH      | IccResource                                     |
| ICCRIDEH      | IccResourceId + サブクラス (IccConvId など)            |
| ICCSEMEH      | IccSemaphore                                    |
| ICCSESEH      | IccSession                                      |
| ICCSRQEH      | IccStartRequestQ                                |
| ICCSYSEH      | IccSystem                                       |
| ICCTIMEH      | IccTime、IccAbsTime、IccTimeInterval、IccTimeOfDay |

| C++ ヘッダー・ファイル | このヘッダーに定義されているクラス |
|---------------|-------------------|
| ICCTMDEH      | IccTerminalData   |
| ICCTMPEH      | IccTempStore      |
| ICCTRMEH      | IccTerminal       |
| ICCTSKEH      | IccTask           |
| ICCUSREH      | IccUser           |
| ICCVALEH      | IccValue (struct) |

注:

1. リストされているすべてのヘッダー・ファイルを含む単一のヘッダーは、ICCEH という名前です。
2. ICCMAIN ファイルには、C++ ヘッダー・ファイルも提供されます。これには、ファウンデーション・クラス・プログラムの作成時に使用する **main** 関数スタブが含まれています。
3. ヘッダー・ファイルは、CICSTS55.CICS.SDFHC370 にあります。

## プログラムの実行

コンパイルされリンクされた (つまり、実行可能な) ファウンデーション・クラス・プログラムを実行するには、以下を実行する必要があります。

1. 実行可能プログラムを CICS で使用できるようにします。このとき、プログラムが適切なディレクトリーまたはロード・ライブラリー内にあることも確認します。また、サーバーによっては、CICS プログラム定義を作成してからでないと (CICS リソース定義機能を使用)、プログラムを実行できない場合があります。
2. CICS 端末にログオンします。
3. プログラムを実行します。

## プログラムのデバッグ

ファウンデーション・クラス・プログラムを正常にコンパイル、リンク、および実行試行した後、デバッグが必要な場合があります。

CICS ファウンデーション・クラス・プログラムをデバッグするために使用できるオプションは、以下のように 3 つあります。

- シンボリック・デバッガーを使用する
- トレースをアクティブにしてファウンデーション・クラス・プログラムを実行する
- CICS 実行診断機能を使用してファウンデーション・クラス・プログラムを実行する

### シンボリック・デバッガー

シンボリック・デバッガーを使用すると、CICS ファウンデーション・クラス・プログラムのソースをステップスルー・デバッグできます。デバッグ・ツールは、IBM C/C++ の機能として組み込まれています。シンボリック・デバッガーを使用して CICS ファウンデーション・クラス・プログラムをデバッグするには、デバッグ情報を実行可能プログラムに追加するフラグを指定してプログラムをコンパイルします。CICS Transaction Server for z/OS の場合、このフラグは TEST(ALL) です。

詳細については、Debug Tool for z/OSを参照してください。

## Tracing (トレース)

デバッグ目的でトレース・ファイルを記述するように、CICS ファウンデーション・クラスを構成できます。

例外のトレースは常にアクティブです。CETR トランザクションは、すべてのCICS プログラム (C++ クラスを使用して開発されたプログラムを含む) の補助トレースおよび内部トレースを制御します。

## 実行診断機能

実行診断機能 (EDF) を使用すると、各 **EXEC CICS** 呼び出しで停止しながら、CICS プログラムをステップスルーできます。表示画面には、CICS ファウンデーション・クラス・タイプのインターフェースではなく、**EXEC CICS** プロシージャ呼び出しインターフェースが表示されます。

EDF を有効にするには、ソース・コード内でプリプロセッサ・マクロ `ICC_EDF` を使用してから、`ICCMAIN` ファイルを組み込みます。

```
#define ICC_EDF //switch EDF on
#include "iccmain.hpp"
```

または、コンパイラ CPARM で適切なフラグを使用して、`ICC_EDF` を宣言します。

## 条件、エラー、および例外

このセクションでは、検出される可能性のあるさまざまなエラー状態に対応するために、ファウンデーション・クラスがどのように設計されているかについて説明します。

### ファウンデーション・クラスの異常終了コード

重大なエラー (オブジェクトを作成するためのストレージが不足している場合など) が発生した場合は、ファウンデーション・クラスが CICS タスクを即時に終了します。

CICS ファウンデーション・クラスの異常終了コードはすべて、`ACLx` という形式です。アプリケーションが終了し、「ACL」で始まる異常終了コードが出された場合は、『』を参照してください。

### C++ 例外およびファウンデーション・クラス

C++ 例外は、予約語 **try**、**throw**、および **catch** を使用して管理されます。

詳細については、コンパイラの資料または参考文献に記載されているいずれかの C++ 資料を参照してください。

以下に、サンプル `ICC$EXC1` を示します (C++ sample programsを参照)。

```

#include "icceh.hpp"
#include "iccmmain.hpp"
class Test {
public:
void tryNumber(short num) {
IccTerminal* term = IccTerminal::instance();
*term << "Number passed = " << num << endl <<
flush;
if (num > 10) {
*term << ">>Out of Range - throwing exception" << endl
<< flush;
throw "!!Number is out of range!!";
}
}
};

```

最初の 2 行は、ファウンデーション・クラスのヘッダー・ファイルと、アプリケーション・プログラムの稼働環境をセットアップする標準 **main** 関数を組み込んでいます。

クラス **Test** を宣言します。これには、**public** メソッド **tryNumber** が 1 つ含まれています。このメソッドは、10 より大きい整数が渡された場合に例外がスローされるように、インラインで実装されています。また、情報を CICS 端末に書き込みます。

```

void IccUserControl::run()
{
IccTerminal* term = IccTerminal::instance();
term->erase();
*term << "This is program 'icc$excl' ..." << endl;
try {
Test test;
test.tryNumber(1);
test.tryNumber(7);
test.tryNumber(11);
test.tryNumber(6);
}
catch(const char* exception) {
term->setLine(22);
*term << "Exception caught: " << exception << endl
<< flush;
}
term->send(24,1,"Program 'icc$excl' complete: Hit PF12 to End");
term->waitForAID(IccTerminal::PF12);
term->erase();
return;
}

```

**IccUserControl** クラスの **run** メソッドに、この例のユーザー・コードが含まれています。

端末表示を消去し、テキストを書き込んだ後、**try** ブロックを開始します。**try** ブロックは、任意の行数の C++ コードに有効範囲を設定できます。

ここで、**Test** オブジェクトを作成し、唯一のメソッドである **tryNumber** を、さまざまなパラメーターを指定して呼び出します。最初の 2 回の呼び出し (1、7) は成功したものの、3 回目 (11) では **tryNumber** によって例外がスローされます。例外によって、プログラムの実行フローが現在の **try** ブロックから外れるため、4 番目の **tryNumber** の呼び出し (6) は実行されません。

その後、**try** ブロックから離れ、適切な **catch** ブロックを検索します。適切な **catch** ブロックは、スローされている例外のタイプと互換性のある引数を含むブロック (ここでは **char\***) です。**catch** ブロックによって、CICS 端末にメッセージが書き込まれ、**catch** ブロックの後の行で実行が再開されます。

この CICS プログラムの出力は以下のとおりです。

```
This is program 'icc$excl' ...
Number passed = 1
Number passed = 7
Number passed = 11
>>Out of Range - throwing exception
Exception caught: !!Number is out of range!!
Program 'icc$excl' complete: Hit PF12 to End
```

前のサンプルのように、CICS C++ ファウンデーション・クラスは **char\*** 例外をスローしませんが、代わりに **IccException** オブジェクトをスローします。

**IccException** にはいくつかのタイプがあります。**type** メソッドは、タイプを示す列挙型を返します。各タイプについて以下で説明します。

#### **objectCreationError**

オブジェクトを作成する試みが無効でした。これは例えば、**IccTask** などの **singleton** クラスの 2 番目のインスタンスを作成しようとした場合に発生します。

#### **invalidArgument**

無効な引数を指定してメソッドが呼び出されました。これは例えば、含まれているデータが多すぎる **IccBuf** オブジェクトがアプリケーション・プログラムによって **IccTempStore** クラスの **writelnItem** メソッドに渡される場合に発生します。

また、長すぎるストリングを使用して **IccResourceId** のサブクラス (例えば、**IccTermId**) を作成しようとした場合にも発生します。

次のサンプルは、C++ sample programsにあります (ファイル ICC\$EXC2)。ここに示すサンプルでは、端末 IO 要求の多くを除いてあります。

```
#include "icceh.hpp"
#include "iccmmain.hpp"
void IccUserControl::run()
{
 try
 {
 IccTermId id1("1234");
 IccTermId id2("12345");
 }
 catch(IccException& exception)
 {
 terminal()->send(21, 1, exception.summary());
 }
 return;
}
```

前の例では、最初の **IccTermId** オブジェクトは正常に作成されますが、2 番目のオブジェクトでは、4 バイトしか許可されないのに 5 バイトのストリング "12345" が使用されているため、**IccException** がスローされています。

す。このサンプル・プログラムからの予期される出力については、C++ sample programsを参照してください。

#### **invalidMethodCall**

メソッドを呼び出すことはできません。これは一般的に、オブジェクトが現在の状態では呼び出しを受け入れられないことが原因です。例えば、読み取るレコードを指定するための **IccRecordIndex** オブジェクトが既にファイルに関連付けられている場合は、**IccFile** オブジェクト上の **readRecord** 呼び出しのみが受け入れられます。

#### **CICSCondition**

**IccCondition** 構造体にリストされている CICS 条件がオブジェクトで発生しました。このオブジェクトは例外をスローするように構成されていました。

#### **familyConformanceError**

このプログラムではファミリー・サブセットの制約が有効になっており、サポートされているすべてのプラットフォームで無効な操作が試行されました。

#### **internalError**

CICS ファウンデーション・クラスによって、内部エラーが検出されました。サービス担当者に連絡してください。

### **CICS 条件**

CICS ファウンデーション・クラスは、アプリケーション実行時に発生する条件を処理するための高性能なフレームワークを提供します。

CICS リソースにアクセスすると、ファウンデーション・クラス・リファレンスに記載されているように、多数の CICS 条件が発生する可能性があります。

条件は、呼び出し側アプリケーションに返されるエラーまたは情報を表します。決定要因は、多くの場合、条件が発生したコンテキストです。

アプリケーション・プログラムは、さまざまな方法で CICS 条件を処理できます。各 CICS リソース・オブジェクト (プログラム、ファイル、データ・キューなど) は、必要に応じてさまざまな方法で CICS 条件を処理できます。

リソース・オブジェクトは、検出可能な各条件に対して、以下のいずれかのアクションを実行するように構成できます。

#### **noAction**

手動条件処理

#### **callHandleEvent**

自動条件処理

#### **throwException**

例外処理

#### **abendTask**

重大エラー処理

### 手動条件処理 (**noAction**):

これは、(任意のリソース・オブジェクト用の) すべての CICS 条件のデフォルトの処理です。

つまり、**condition** メソッドを使用して、条件を手動で処理する必要があるということです。以下に例を示します。

```
IccTempStore temp("TEMP1234");
IccBuf buf(40);
temp.setActionOnCondition(IccResource::noAction,
IccCondition::QIDERR);
buf = temp.readNextItem();
switch (temp.condition())
{
case IccCondition::QIDERR:
//do whatever here
:
default:
//do something else here
}
```

### 自動条件処理 (**callHandleEvent**):

以下のように、QIDERR など任意の CICS 条件に対してこれを活動化します。

```
IccTempStore temp("TEMP1234");
temp.setActionOnCondition(IccResource::callHandleEvent,
IccCondition::QIDERR);
```

オブジェクト「temp」の任意のメソッドに対する呼び出しにより、CICS で QIDERR 状態が発生した場合、**handleEvent** メソッドが自動的に呼び出されます。**handleEvent** メソッドは仮想メソッドにすぎないため、オブジェクトが **IccTempStore** のサブクラスに属していて **handleEvent** メソッドがオーバーライドされている場合のみ、この呼び出しが役立ちます。

**IccTempStore** のサブクラスを作成し、コンストラクターを宣言して、**handleEvent** メソッドをオーバーライドします。

```
class MyTempStore : public IccTempStore
{
public:
MyTempStore(const char* storeName) : IccTempStore(storeName) {}
HandleEventReturnOpt handleEvent(IccEvent& event);
};
```

ここで、**handleEvent** メソッドを実装します。

```

IccResource::HandleEventReturnOpt
MyTempStore::handleEvent(IccEvent& event)
{
switch (event.condition())
{
case ...
:
case IccCondition::QIDERR:
//Handle QIDERR condition here.
:
//
default:
return rAbendTask;
}
}

```

このコードは、特定の CICS 条件の「callHandleEvent」に対して構成されている任意の **MyTempStore** オブジェクトで呼び出されます。

#### 例外処理 (**throwException**):

以下のように、QIDERR など任意の CICS 条件に対してこれを活動化します。

```

IccTempStore temp("TEMP1234");
temp.setActionOnCondition(IccResource::throwException,
IccCondition::QIDERR);

```

例外処理は、**try**、**throw**、および **catch** を使用する C++ 例外処理モデルによるものです。以下に例を示します。

```

try
{
buf = temp.readNextItem();
:
}
catch (IccException& exception)
{
//Exception handling code
:
}

```

**try** ブロックの内部にあるメソッドのいずれかでオブジェクト「temp」の QIDERR 状態を発生させた場合に例外がスローされます。例外がスローされると、C++ では、スタックをアンwindし、適切な **catch** ブロックで実行を再開します。**try** ブロック内で再開することはできません。さらに詳細な例については、サンプル ICC\$EXC3 を参照してください。

注: この例以外に多くの理由で、ファウンデーション・クラスから例外がスローされることがあります。詳しくは、639 ページの『C++ 例外およびファウンデーション・クラス』を参照してください。

#### 重大エラー処理 (**abendTask**):

このオプションにより、CICS では、特定の条件が発生したときにタスクを終了することができます。

以下のように、QIDERR など任意の CICS 条件に対してこれを活動化します。

```
IccTempStore temp("TEMP1234");
temp.setActionOnCondition(IccResource::abendTask,
IccCondition::QIDERR);
```

CICS がオブジェクト「temp」に対して QIDERR 状態を発生させた場合、CICS タスクが ACL3 異常終了で終了します。

## プラットフォームの相違

CICS ファウンデーション・クラスは、以下で説明するように、それが実行される特定の CICS プラットフォームから独立するように設計されています。ただし、プラットフォームによっていくつかの相違点があります。これらの相違点、およびそれぞれのコピー方法について、以下で説明します。

注: このセクションでは、詳しく説明するために、他の CICS プラットフォームに言及しています。それらのプラットフォームにおける CICS ファウンデーション・クラスのテクノロジー・リリースがありました。

アプリケーションは以下のいずれかのモードで実行できます。

### fsAllowPlatformVariance

CICS ファウンデーション・クラスを使用して記述されたアプリケーションは、ターゲット CICS サーバーで使用可能なすべての機能にアクセスできます。

### fsEnforce

アプリケーションが使用できる機能は、すべての CICS サーバー (z/OS と UNIX) 間で使用できる CICS 機能に制限されます。

デフォルトではプラットフォームの差異が認められますが、代わりにすべての CICS プラットフォームに共通の機能のみをアプリケーションで使用するよう強制することもできます。

クラス・ヘッダーはすべてのプラットフォームで同じで、あらゆる CICS プラットフォーム上のファウンデーション・クラスで使用可能なすべての CICS 機能を「サポート」(つまり定義) します。各プラットフォームの制約事項については、『ファウンデーション・クラス・リファレンス』で説明しています。プラットフォームの差異は以下のレベルで存在します。

- オブジェクト・レベル
- メソッド・レベル
- パラメーター・レベル

オブジェクト・レベル:

一部のオブジェクトは特定のプラットフォームでサポートされません。

例えば、**IccConsole** オブジェクトは、CICS(r) for AIX® でコンソール・サービスをサポートしていない場合、CICS(r) for AIX(r) では作成できません。

CICS(r) for AIX(r) で **IccConsole** オブジェクトを作成しようとする、タイプ「platformError」の **IccException** オブジェクトがスローされますが、他のプラットフォームでは許容されます。

```
IccConsole* cons = console(); //No good on CICS for AIX
```

「fsEnforce」を選択してアプリケーションを初期化した場合 (`initializeEnvironment` を参照)、前のどちらの例でも、タイプ「familyConformanceError」の **IccException** オブジェクトがすべてのプラットフォームでスローされます。

**IccConsole** クラスや **IccJournal** クラスのオブジェクトとは異なり、ほとんどのオブジェクトはどの CICS サーバー・プラットフォーム上でも作成できます。ただし、メソッドの使用が制限される場合があります。ファウンデーション・クラス：リファレンスに、すべてのプラットフォームの制約事項が記載されています。

メソッド・レベル:

あるプラットフォームで正常に実行されるメソッドが、別のプラットフォームでは問題を引き起こす場合があります。

例えば、**IccControl** クラスのメソッド **programId** について考えてみます。

```
void IccUserControl::run()
{
if (strcmp(programId.name(), "PROG1234") == 0)
//do something
}
```

ここで、メソッド **programId** は CICS TS for z/OS では正しく実行されますが、CICS(r) for AIX(r) ではタイプ「platformError」の **IccException** オブジェクトがスローされます。

また、ファミリー・サブセットの適用によってアプリケーションを初期化した場合 (**Icc** 構造体の `initializeEnvironment` 機能を参照)、どのような CICS サーバー・プラットフォームでも、メソッド **programId** でタイプ「familyConformanceError」の **IccException** オブジェクトをスローします。

パラメーター・レベル:

このレベルでは、メソッドはすべてのプラットフォームでサポートされますが、特定の定位置パラメーターはプラットフォームの制限を受けます。

**IccTask** クラスのメソッド **abend** について考えます。

```

task()->abend();
1

task()->abend("WXYZ");
2

task()->abend("WXYZ", IccTask::respectAbendHandler);
3

task()->abend("WXYZ", IccTask::ignoreAbendHandler);
4

task()->abend("WXYZ", IccTask::ignoreAbendHandler,
5
IccTask::suppressDump);

```

異常終了 **1** から **4** は、すべての CICS サーバー・プラットフォームで正常に実行されます。

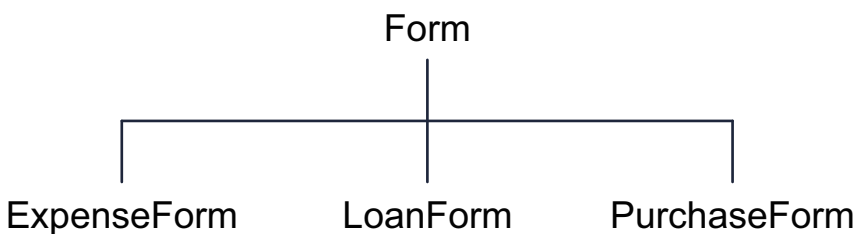
ファミリー・サブセットの適用がオフの場合は、CICS(r) for AIX(r) プラットフォームでは abend **5** がタイプ「platformError」の **IccException** オブジェクトをスローしますが、CICS Transaction Server for z/OS プラットフォームではスローしません。

ファミリー・サブセットの適用がオンの場合は、ターゲット CICS プラットフォームにかかわらず、abend **5** がタイプ「familyConformanceError」の **IccException** オブジェクトをスローします。

## ポリモアフィック動作

ポリモアフィズム (ポリ = 多数、モアフィ = フォーム) は、多くの異なるフォームのオブジェクトを同じフォームとして扱う機能です。

ポリモアフィズムは、継承および仮想関数を使用して C++ で実現します。一般的な Form を特殊化した 3 つのフォーム (ExpenseForm、LoanForm、PurchaseForm) について考えてみましょう。



各フォームは、何らかの時点で印刷する必要があります。プロシージャ型プログラミングでは、`print` 関数をコーディングして異なる 3 つのフォームを処理するか、または 3 つの異なる関数 (`printExpenseForm`、`printLoanForm`、`printPurchaseForm`) を記述します。

C++ では、以下のように、はるかに簡潔にこれを行えます。

```

class Form {
public:
virtual void print();
};
class ExpenseForm : public Form {
public:
virtual void print();
};
class LoanForm : public Form {
public:
virtual void print();
};
class PurchaseForm : public Form {
public:
virtual void print();
};

```

これらの指定変更された各関数は、各フォームが正しく印刷されるように実装されます。この時点で、フォーム・オブジェクトを使用するアプリケーションでは、以下を実行できます。

```

Form* pForm[10]
//create Expense/Loan/Purchase Forms...
for (short i=0 ; i < 9 ; i++)
pForm->print();

```

ここで、Expense、Loan、および Purchase Form を組み合わせて 10 個のオブジェクトを作成します。ただし、基本クラス **Form** のポインターを扱うため、所有しているフォーム・オブジェクトの種類を認識する必要はありません。正しい **print** メソッドが自動的に呼び出されます。

ファウンデーション・クラスでは、制限付きのポリモフィック動作を使用できます。基本クラス **IccResource** には、以下の 3 つの仮想関数が定義されています。

```

virtual void clear();
virtual const IccBuf& get();
virtual void put(const IccBuf&
buffer
);

```

これらのメソッドは、可能な場合は **IccResource** のサブクラス内に実装されています。

| クラス          | clear | get | put |
|--------------|-------|-----|-----|
| IccConsole   | ×     | ×   | ✓   |
| IccDataQueue | ✓     | ✓   | ✓   |
| IccJournal   | ×     | ×   | ✓   |
| IccSession   | ×     | ✓   | ✓   |
| IccTempStore | ✓     | ✓   | ✓   |
| IccTerminal  | ✓     | ✓   | ✓   |

これらの仮想メソッドは、テーブル内のメソッドを除き、**IccResource** のサブクラスではサポートされていません。

注: 基本クラス **IccResource** 内の **clear**、**get**、および **put** のデフォルト実装から、例外がスローされるため、ユーザーはサポートされていないメソッドを呼び出さずに済みます。

## ポリモアフィック動作の例

以下のサンプルは、ICC\$RES2 ファイルとして samples ディレクトリーにあります。

ここでは、端末の入出力要求を省略して示しています。C++ sample programsを参照してください。

```
#include "icceh.hpp"
#include "iccmmain.hpp"
char* dataItems[] =
{
 "Hello World - item 1",
 "Hello World - item 2",
 "Hello World - item 3"
};
void IccUserControl::run()
{
```

ここでは、ファウンデーション・クラスのヘッダーと **main** 関数を組み込みます。**dataItems** には、いくつかのサンプル・データ項目が含まれています。**IccUserControl** クラスの **run** メソッドのアプリケーション・コードを作成します。

```
 IccBuf buffer(50);
 IccResource* pObj[2];
```

データ項目を保持するために、**IccBuf** オブジェクト (最初に 50 バイト) を作成します。**IccResource** オブジェクトへの 2 つのポインターの配列を宣言します。

```
 pObj[0] = new IccDataQueue("ICCQ");
 pObj[1] = new IccTempStore("ICCTEMPS");
```

**IccResource** から派生するクラスを持つ 2 つのオブジェクト、**IccDataQueue** と **IccTempStore** を作成します。

```
 for (short index=0; index <= 1 ; index++)
 {
 pObj[index]->clear();
 }
```

両方のオブジェクトに対して、**clear** メソッドを呼び出します。これは、アプリケーション・プログラムに対して透過的な方法で、オブジェクト別にそれぞれ処理されます。これがポリモアフィック動作です。

```
 for (index=0; index <= 1 ; index++)
 {
 for (short j=1 ; j <= 3 ; j++)
 {
 buffer = dataItems[j-1];
 pObj[index]->put(buffer);
 }
 }
```

ここで、各リソース・オブジェクトに 3 つのデータ項目を **put** します。さらに、**put** メソッドがオブジェクト・タイプに適切な方法で要求に応答します。

```
for (index=0; index <= 1 ; index++)
{
buffer = pObj[index]->get();
while (pObj[index]->condition() == IccCondition::NORMAL)
{
buffer = pObj[index]->get();
}
delete pObj[index];
}
return;
}
```

**get** メソッドを使用して、各リソース・オブジェクトからデータ項目を再読み込みします。リソース・オブジェクトを削除し、CICS に制御を戻します。

## ストレージ管理

C++ オブジェクトは通常、スタックまたはヒープに保管されます。

スタック上のオブジェクトは、有効範囲を超えると、自動的に破棄されますが、ヒープ上のオブジェクトは破棄されません。

CICS ファウンデーション・クラスによって内部的に作成されるオブジェクトの多くは、スタック上ではなくヒープ上に作成されます。これにより、一部の CICS サーバー環境では問題が発生することがあります。

CICS Transaction Server for z/OS では、CICS および Language Environment によって、すべてのタスク・ストレージが管理されるため、タスク終了 (正常終了または異常終了) 時にストレージが解放されます。

CICS for AIX 環境では、ヒープに割り振られたストレージは、タスク終了時に自動的に解放されません。これは、アプリケーション・プログラマーがヒープ上のオブジェクトを明示的に削除することを忘れた場合や、さらに深刻な状況として、タスクが異常終了した場合などに、「メモリー・リーク」につながるおそれがあります。

この問題は、CICS ファウンデーション・クラスでは解消されています。基本ファウンデーション・クラス **IccBase** で、演算子 **new** および **delete** を指定します。これらを構成するには、動的ストレージ割り振り要求を CICS タスク・ストレージにマップします。これにより、タスク終了時に すべてのストレージが自動的に解放されます。この方法の欠点は、パフォーマンス上の問題です。ファウンデーション・クラスは一般的に、単一の大規模な割り振り要求ではなく、小規模のストレージ割り振り要求を大量に発行します。

この機能は、ファウンデーション・クラスの使用前に発行する必要がある **Icc::initializeEnvironment** 呼び出しの影響を受けます。(この関数は、デフォルトの **main** 関数から呼び出されます。main 関数を参照してください。)

**initializeEnvironment** 関数に渡される最初のパラメーターは、以下の 3 つの値のいずれかを取る列挙です。

## cmmDefault

デフォルト・アクションは、以下のようにプラットフォームに依存します。

**z/OS** 「cmmNonCICS」と同じです (『cmmNonCICS』セクションを参照)。

## UNIX

「cmmCICS」と同じです (『cmmCICS』セクションを参照)。

## cmmNonCICS

**IccBase** クラス内の **new** 演算子および **delete** 演算子は、動的ストレージ割り振り要求を CICS タスク・ストレージにマップしません。代わりに、C++ のデフォルトの **new** 演算子と **delete** 演算子が呼び出されます。

## cmmCICS

**IccBase** クラス内の **new** 演算子および **delete** 演算子は、動的ストレージ割り振り要求を CICS タスク・ストレージ (タスクの正常終了または異常終了時に自動的に解放される) にマップします。

ファウンデーション・クラスに提供されるデフォルトの **main** 関数は、「cmmDefault」の **enum** を使用して **initializeEnvironment** を呼び出します。これは、プログラム内で以下のように変更できます。このとき、提供された「ヘッダー・ファイル」**ICCMAN** を変更する必要はありません。

```
#define ICC_CLASS_MEMORY_MGMT Icc::cmmNonCICS
#include "iccmmain.hpp"
```

または、コンパイル時に **DEV(ICC\_CLASS\_MEMORY\_MGMT)** オプションを設定します。

## パラメーター受け渡し規則

ファウンデーション・クラス・メソッド呼び出しでオブジェクトの受け渡しに使用される規則は、オブジェクトが必須である場合は参照による受け渡し、オブジェクトがオプションである場合はポインターによる受け渡しです。

例えば、**IccStartRequestQ** クラスのメソッド **start** を確認してみましょう。このメソッドには、以下のシグニチャーがあります。

```
const IccRequestId& start(const IccTransId&
transId,
const IccTime* time=0,
const IccRequestId* reqId=0);
```

上記の規則から、**IccTransId** オブジェクトが必須である一方、**IccTime** オブジェクトと **IccRequestId** オブジェクトはどちらもオプションであることがわかります。これによりアプリケーションは、以下のいずれかの方法でこのメソッドを使用できます。

```

IccTransId trn("ABCD");
IccTimeInterval int(0,0,5);
IccRequestId req("MYREQ");
IccStartRequestQ* startQ = startRequestQ();
startQ->start(trn);
startQ->start(trn, &int);
startQ->start(trn, &int, &req);
startQ->start(trn, 0, &req);

```

## 「read」メソッドから返される **IccBuf** 参照内のデータの有効範囲

**IccResource** の多くのサブクラスには、**const IccBuf** 参照を返す「read」メソッド (例えば、**IccFile::readRecord**、**IccTempStore::readItem**、および **IccTerminal::receive**) があります。

**IccBuf** 参照から独自の **IccBuf** オブジェクトへデータをコピーするのではなく、**IccBuf** オブジェクトの参照を保持する場合は、注意が必要です。例えば、以下について考えてみます。

```

IccBuf buf(50);
IccTempStore store("TEMPSTOR");
buf = store.readNextItem();

```

ここでは、**IccTempStore::readNextItem** から返された **IccBuf** 参照内のデータが、アプリケーション独自の **IccBuf** オブジェクトに 即時 にコピーされるため、データが後から無効になっても影響はありません。ただし、アプリケーションは次のようになる可能性があります。

```

IccTempStore store("TEMPSTOR");
const IccBuf& buf = store.readNextItem();

```

ここでは、**IccTempStore::readNextItem** から返された **IccBuf** 参照が、アプリケーション独自のストレージに コピーされない ため、注意が必要です。

注: 有効なデータが含まれていない **IccBuf** オブジェクトの参照の使用を回避するために、このスタイルのプログラミングを使用することは、お勧めしません。

返された **IccBuf** 参照には一般的に有効なデータが含まれていますが、以下のいずれかの条件に該当する場合は除きます。

- **IccResource** オブジェクトで別の「read」メソッド (上記の例では、別の **readNextItem** または **readItem** メソッド) が呼び出される。
- リソース更新がコミットされる (メソッド **IccTask::commitUOW** を参照)。
- タスクが (正常または異常) 終了する。

---

## 第 8 章 COBOL アプリケーションの開発

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。

アプリケーション・プログラミング言語に関する CICS サポートの変更点には、CICS Transaction Server for z/OS, バージョン 5 リリース 5 でサポートされている COBOL コンパイラと、z/OS におけるそれらの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 5 リリース 5 の資料において COBOL に言及している場合は、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、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 5 リリース 5 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 コンパイラは、VS COBOL II コンパイラです。

COBOL コンパイラ間のマイグレーションについて詳しくは、Enterprise COBOL for z/OS 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 5 リリース 5 では、VS COBOL II コンパイラ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 (Language Environment) 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 で提供されるランタイム・ライブラリーは、サポートされていません。

660 ページの『VS COBOL II プログラム』で、VS COBOL II コンパイラ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。

VS COBOL II コンパイラは、言語環境プログラム (Language Environment) のランタイム・オプションを調整して、これらのアプリケーションが正しく実行されるようにします。詳しくは、「Enterprise COBOL for z/OS 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 5 リリース 5 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 5 リリース 5 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。Enterprise COBOL で使用されている新しい Java ベースの OO COBOL は、CICS 変換プログラムではサポートされていません。

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します。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ありません。

## 作業用ストレージ

コンパイラー・オプション DATA(24) では、作業用ストレージは 16 MB 境界より下に割り振られます。コンパイラー・オプション DATA(31) では、作業用ストレージは 16 MB 境界より上に割り振られます。

```
Export-Package: com.ibm.jzos,
 com.ibm.jzos.fields
```

---

## 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適用されます。

デフォルトでは、CICS 変換プログラムおよび COBOL コンパイラーは、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) を指定すると、コンパイラーは、IGYCCICS テーブルを使用し、また、CICS でサポートされていない COBOL ワードにエラー・メッセージ付きでフラグを立てます。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。制限され、IGYCCICS テーブルに含ま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 (Language Environment) 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、制約事項が適用されます。「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。
- 以下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。これらの項目は、前述の制限付き SORT 機能に関連付けられ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。

## Enterprise COBOL V5 以上に関する考慮事項

CICS で COBOL V5 以上を使用する場合は、Enterprise COBOL for z/OS Knowledge Center で説明されている CICS 移行における考慮事項に注意してください。

### 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。
- 領域 B (12 から 71 カラム) で、EXEC CICS と END-EXEC の両方のステートメントを開始します。キーワード EXEC は SQL ステートメントなどその他のステートメントの一部として領域 A で使用される可能性があるため、EXEC CICS ステートメントが領域 A (8 から 11 カラム) で開始された場合に、統合された CICS 変換プログラムはエラー・メッセージを生成しないことに注意してください。
- 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 16 MB 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 128 MB です。

ストレージ・アカウンティングおよび保管域に 80 バイトが必要であるため、その分も含めて制限内に収めなければなりません。

- DLI オプションが指定され、ENTRY ステートメントが既存のプログラムの PROCEDURE DIVISION ヘッダーのすぐ後に続く場合は、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してから CICS のプログラムを呼び出します。

- 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で 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。Language Environment ランタイム・オプションについては、「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

16 MB 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。

### 64 ビット・アドレッシング

64 ビット・アドレッシング・モードは、COBOL プログラムについてはサポートされていません。

### 64 ビット常駐

CICS では、64 ビット常駐モード (RMODE(64)) はサポートされておらず、すべての RMODE(64) プログラムが RMODE(31) として処理されます。つまり、RMODE(64) プログラムは、64 ビット (2 GB 境界より上) ストレージではなく、31 ビット (16 MB 境界より上) ストレージにロードされます。

## 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。
  - 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
- DLL コンパイラー・オプションを使用できます。
- 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・オプションの詳細については、「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)。

アプリケーションが EIB のフィールドを使用する場合、DFHEIBLK コピーブックは、EIBCALEN などのフィールドを PICTURE S9(4) USAGE COMPUTATIONAL として定義します。DFHEIBLK コピーブックとともに TRUNC(OPT) コンパイラー・オプションを使用すると、2 進数フィールドの 9999 より大きい値が切り捨てられる場合があります。切り捨ての問題を回避するには、更新されたバージョンの DFHEIBLK コピーブックを使用する、統合された変換プログラムを使用することをお勧めします。統合された変換プログラムによって使用されるバージョンの DFHEIBLK は、TRUNC(OPT) または TRUNC(BIN) コンパイル・オプションによって影響されるすべてのフィールドを USAGE COMP-5 として定義します。

TRUNC オプションの詳細については、「Enterprise COBOL for z/OS カスタマイズ・ガイド」を参照してください。

- 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 16 MB 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。

## Language Environment の 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 (Language Environment) 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、687 ページの『言語環境プログラム (Language Environment) 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。

図 124 はこ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 を定義しており、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 124. DLI 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 (Language Environment) は、VS COBOL II コンパイラ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、このコンパイラは、言語環境プログラムに準拠していない (言語環境プログラム以前のコンパイラである) ため、使用に関連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 (Language Environment) 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は、「Enterprise COBOL for z/OS コンパイラおよびランタイム 移行ガイド」を参照してください。

### 言語環境プログラム (Language Environment) の呼び出し可能サービス

言語環境プログラム (Language Environment) に準拠した COBOL コンパイラ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 (Language Environment) 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、静的にしる動的にしる、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 (Language Environment) 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 IGSZWLKA 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 (Language Environment) の下で稼働しますが、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 (Language Environment) 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 (Language Environment) 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。次のように機能します。

- ランタイム時に検出されるエラーが発生したときに 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 で実行する既存の VS COBOL II アプリケーションとの互換性を提供する。
- ランタイム時に検出したすべての未処理エラーを、VS COBOL II が発行する対応するユーザー 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)。

サンプルのユーザー条件処理ルーチン CEEWUCHA が、実行時に使用可能であることを (例えば、STEPLIB 連結または LPA を使用して) 確認します。プログラムの自動インストールを使用するのではなく、ご使用の CICS 領域の CICS システム定義データ・セット (CSD) に条件処理ルーチンを定義します。

---

## 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。

662 ページの図 125 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 125. 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、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は、「Enterprise COBOL for z/OS プログラミング・ガイド」および「Enterprise COBOL バージョン 4 リリース 2 パフォーマンス・チューニング」(<http://www.ibm.com/support/docview.wss?uid=swg27018287>) を参照してください。

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、684 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)。664 ページの図 126 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。

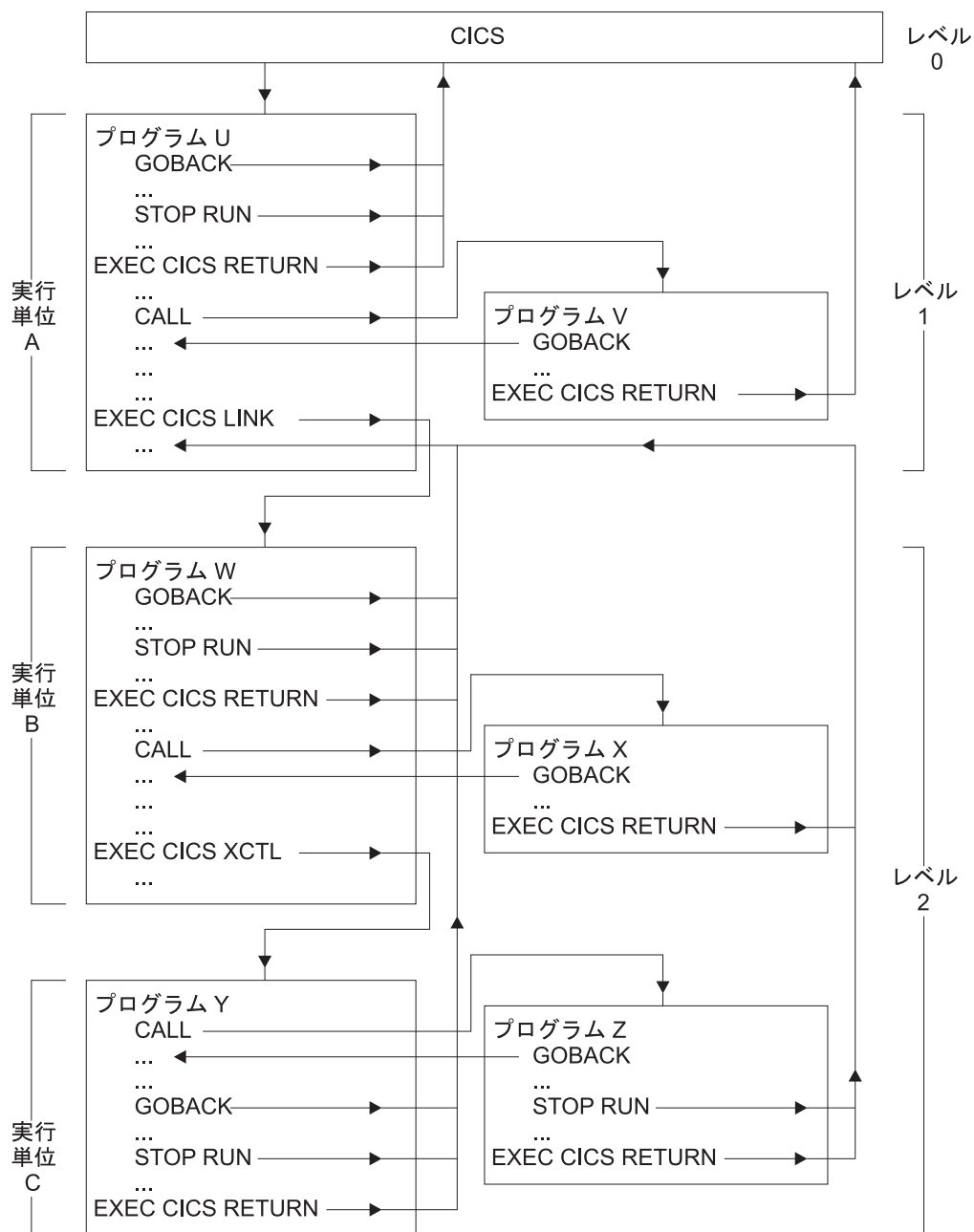


図 126. COBOL プログラム、実行単位、および CICS 間の制御のフロー

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、163 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。

### Translation

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・バック域または、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。

### COBOL の再帰呼び出し

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、言語環境プログラムはメッセージ IGZ0064S を CEEMSG に発行し、異常終了 (4038) します。

プログラム A およびプログラム B が PROGRAM-ID に RECURSIVE キーワードを持っている場合は、再帰呼び出しが許可されます。

### 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 16 MB 境界より上のアドレスにある場合には、CICS はそれを 16 MB 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、**EXEC CICS RETURN** コマンドまたはネイティブ言語戻りコマンド (COBOL ステートメントの **GOBACK** など) を使用して戻る必要があります。

### 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** や **EXIT PROGRAM** など) を使用して戻る必要があります。呼び出されたサブプログラムで **EXEC CICS RETURN** を使用すると、呼び出し側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)。

リンクされるサブプログラムへの各入り口で、新しく初期設定された **LOCAL-STORAGE** セクションのコピーが提供されます。

### 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 論理レベル内の呼び出し先サブプログラムへの各入り口で、新しく初期設定された LOCAL-STORAGE SECTION のコピーが提供されます。

## 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

- 言語環境プログラム (Language Environment) および CBLPSHPOP ON を使用した、動的に呼び出された COBOL プログラムが異常終了した場合、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。
- 動的に呼び出された COBOL プログラムが CBLPSHPOP OFF の設定で異常終了し、呼び出し側プログラムの条件、AID、または異常終了処理がアクティブである場合、プログラムは異常終了コード APC2 を出して終了します。
- 静的に呼び出された COBOL プログラムの場合は、CBLPSHPOP の設定に関係なく、条件、AID、および異常終了処理は引き続き有効になります。

---

## 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 5 リリース 5 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。

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 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。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。

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。

プログラムの変換とその実行準備について詳しくは、881 ページの『第 13 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/小文字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)
```

*data-name* は、添字付きまたは修飾、あるいはその両方にすることができます。*leftmost-character-position* および *length* は、算術式に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 節がサポートされます。最上位レベルのプログラム ( 674 ページの『ネストされた COBOL プログラム』を参照) 内で、GLOBAL 節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え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 オプションで使うことができます。

## 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、989 ページの『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。詳細については、673 ページの図 127 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おかまいませんが、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。674 ページの図 128 に、このように編集した後の 673 ページの図 127 からの出力を示します。

バッチ・コンパイルについては、989 ページの『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 DFHYITVL を 2 つのプロシージャに分割する。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 127. 編集前のコンパイラ出力

5. 674 ページの図 128 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、CICSTS55.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。674 ページの図 128 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としています。推奨するステートメントは、次の通りです。

```
//SYSLIB DD DSN=
CICSTS55.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 128. 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、最上位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上位のプログラム (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、入力ソースが最上位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、最上位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は、DFHEIBLK および DFHCOMMAREA を最上位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. 前述のいずれかの理由で、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には、そのプログラムへの呼び出しの 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。

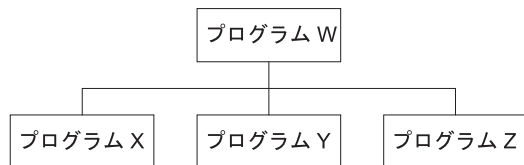


図 129. ネストされたプログラムの例 - ネスト構造

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。

図 130 に、これらのポイントを示します。

```

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 130. ネストされたプログラムの例 - コーディング



---

## 第 9 章 プログラミング言語と言語環境プログラム (Language Environment)

z/OS の要素として提供される言語環境プログラム (Language Environment) 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を使用すると、プログラミング言語やシステム・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 1 つだけ使用できます。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。

Language Environment 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していました。言語環境プログラム (Language Environment) 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、従来のコンパイラで提供されるランタイム・ライブラリーを置き換えます。共通環境には、2 つの重要な利点があります。

- 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。
- 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。以下に例を示します。
  - 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 は、さまざまなコンパイラによっ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。このリリースの CICS Transaction Server for z/OS でサポートされるコンパイラのリストについては、アプリケーション・プログラミング言語に関する CICS サポートの変更点を参照してください。

Language Environment では、AMODE(64) アセンブラー言語プログラムはサポートされていません。

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のほとんどは、言語環境プログラム準拠のコンパイラです。つまり、これらのコンパイラ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。CICS と言語環境プログラム (Language Environment) は、言語環境プログラム (Language Environment) に準拠していない一部の言語環境プログラム以前のコンパイラでコンパイルされるプログラムも

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でコンパイルおよびリンクされたアプリケーションは、Language Environment が提供するランタイム・サポートを使用して正常に実行されることもあります。これらのアプリケーションでは、再コンパイルや再リンク・エディットが必要とされない場合があります。一部の環境では、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る場合があります。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・ガイドおよび使用中の言語の「*Compiler and Run-Time Migration Guide*」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で提供されるランタイム・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。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 で、言語環境プログラムに準拠するアセンブラの MAIN プログラムを作成することもできます。アセンブラ・プログラムについて詳しくは、575 ページの『第 6 章 アセンブラ言語アプリケーションの開発』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。

### 一般

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、静的にしる動的にしる、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 (Language Environment) の異常終了と条件処理

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 がアクティブではない場合に、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 (Language Environment) 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 (Language Environment) の下で実行されている場合、異常終了に対してタスクがスケジュールされている場合に取りられる処置は、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、CICS HANDLE ABEND で定義した処置が行われます。言語環境プログラム (Language Environment) の条件処理は、異常終了やプログラム割り込みを制御せず、CEEHDLR で設定したすべてのユーザー作成条件ハンドラーは無視されます。
- CICS HANDLE ABEND がアクティブではない場合、ランタイム・オプション TRAP(ON) が指定されていると、言語環境プログラム (Language Environment) の条件処理は、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ず、異常終了が続行します。通常の言語環境プログラム (Language Environment) の条件処理について詳しくは、「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 (Language Environment) のユーザー作成の条件処理ルーチン

言語環境プログラム (Language Environment) 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、そのレベルの条件処理ルーチン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。CEEHDLU サービスを使用してそれを明示的に登録抹消することができます。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。

ユーザー作成の言語環境プログラム (Language Environment) 条件処理ルーチンを (COBOL 以外で) 作成する場合、大部分の CICS コマンド (NOHANDLE、RESP、または RESP2 オプションを指定してコーディングされているもの) を使用して、条件処理ルーチンの実行中にさらなる条件が発生することを防ぐことができます。使用できないコマンドは次のとおりです。これらのコマンドは、条件処理ルーチン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、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。呼び出されるサブルーチンは、他の CICS コマンドを実行する前に、EXEC CICS ADDRESS EIB(DFHEIPTR) コマンドを発行する必要があります。

ユーザー作成の言語環境プログラム (Language Environment) 条件処理ルーチンをアプリケーションで使用するには、その条件処理ルーチンが (例えば、STEPLIB 連結か LPA を使用して) 実行時に使用可能になっている必要があります。プログラムの自動インストールを使用するのではなく、ご使用の CICS 領域の CICS システム定義データ・セット (CSD) にそのような条件処理ルーチンを定義します。これには、サンプルのユーザー作成条件ハンドラー CEEWUCHA が含まれます。

言語環境プログラム (Language Environment) 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場合、言語環境プログラム (Language Environment) は、CICS に対して、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 RUWAPOL の設定値によって決まります。

RUWAPOL=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)。

RUWAPOL=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 RUWAPOL=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 CICS DATAKEY または USER DATAKEY ストレージを取得することができます。ただし、USER の EXEC KEY を備えたプログラムでは、CICS DATAKEY 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

Language Environment を使用して、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ている複数のプログラムから構成されるアプリケーションを作成することができます。

アセンブラー言語サブルーチンを高水準言語 (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 を使用してサブルーチンを呼び出すがそのサブルーチンが別の言語で書かれているというような場合には、はるかに慎重に考慮しなければならず、言語間通信 (ILC) が必要になります。

Language Environment では、ILC アプリケーションは、複数の HLL (場合によっては、アセンブラー言語も) を使用して作成されているアプリケーションと定義されています。詳しくは、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。

Language Environment では、CICS で実行単位内に ILC がある場合は、それぞれのコンパイル単位を Language Environment 準拠のコンパイラでコンパイルする必要があります。CICS は次の HLL をサポートしています。

- C/C++
- COBOL
- PL/I

以降のセクションでは、各 HLL ペアの条件について説明します。ご使用のアプリケーションに含まれている HLL が 2 つのみの場合は、該当するセクションを参照

してください。アプリケーションに 3 種類の HLL がすべて含まれる場合は、そのアプリケーション内の各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

C/C++ および COBOL によって書かれたルーチン間の ILC が Language Environment でサポートされるための条件は、以下の要素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 プログラムにある `#pragma` リンケージ・ステートメント (ある場合)。
- C プログラムがエクスポートするのは関数か変数か。
- C++ プログラムにある `extern` ステートメント (ある場合)。

これらの要素の影響については、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。

## 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、C/C++ および PL/I でコンパイルされるルーチン間の ILC は、Language Environment で以下のようにサポートされます。

- 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 ルーチンが取り出せるのは、CICS 変換プログラムによって処理されていない C/C++ ルーチンだけである。なぜならば、動的呼び出しの間は、変換プログラムによって作成された特定の静的フィールドが正しく設定できないからです。

## COBOL および PL/I

CICS では、Language Environment 準拠の COBOL および PL/I コンパイラーによってコンパイルされるルーチン間の ILC は、Language Environment で以下のようにサポートされます。

- 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。

## アセンブラ言語

- Language Environment 準拠の任意の HLL プログラムから、Language Environment 準拠のアセンブラ言語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。反対に、Language Environment 準拠のアセンブラ言語ルーチンは、Language Environment のマクロである CEEFETCH と CEELOAD のいずれかを使用して、Language Environment 準拠の任意のルーチンを静的に呼び出したり、別の (アセンブラ言語または HLL) ルーチン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 によって取り出した 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。これは、解放されるモジュールに PL/I を使用した ILC がない場合にのみ、C/C++、COBOL、および PL/I から行うことができます。

さらに、Language Environment 準拠の任意の HLL プログラムまたはアセンブラ言語サブルーチンから、準拠していないアセンブラ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、非準拠のアセンブラ言語ルーチンは、Language Environment のマクロを使用できないので、Language Environment 準拠の任意のルーチンを静的に呼び出すことや、準拠ルーチンを取り出したりロードしたりすることはできません。

C または C++ を呼び出すアセンブラ言語では、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 では ILC アプリケーションで DL/I を使用する場合、EXEC DLI ステートメントと CALL xxxTDLI のいずれかによる DL/I 呼び出しは、メインプログラムと同じ言語で作成されたプログラムでのみ行うことができます。

Language Environment 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、その情報を参照してください。

---

## 言語環境プログラム (Language Environment) 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 から開始されるプログラムでは、Language Environment の事前初期設定モジュール CEEPIPI を使用します。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。Language Environment のランタイム・オプションの優先順位の完全な順序について詳しくは、z/OS Language Environment プログラミング・ガイドを参照してください。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. CEECCICS にビルドされた CEEDOPT CSECT には、IBM 言語環境プログラムのデフォルトのランタイム・オプションが含まれます。SCEESAMP に配置された CEEWCOPT サンプル・ジョブを使用して、これらのデフォルトのランタイム・オプションを変更できます。この方法はサポートされていますが、CEEPRMxx parmlib メンバーを使用してランタイム・オプションを指定するのが、推奨される最も簡単な方法です。
2. CEEPRMxx parmlib メンバーは、CICS のデフォルトの言語環境プログラムのランタイム・オプションを設定するための推奨される方法である、CEEEOPT オプション・グループをサポートしています。
3. 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。
4. 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)。

5. 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。
6. 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。以下に例を示します。

```
#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。以下に例を示します。

```
DECLARE PLIXOPT CHARACTER(18) VARYING STATIC EXTERNAL
INIT('RPTOPTS(ON) NOSTAE');
```
7. 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。詳細については、977 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 つの方式のみを選択して、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 (Language Environment) 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 (Language Environment) 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るには、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と、子エンクレーブと呼ばれる言語環境プログラム (Language Environment) 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 コマンドが実行される場合も、同様の結果になります。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。この環境変数は、Language Environment のランタイム・オプション ENVAR を使用して指定することができます。

CICSVAR を CEEDOPT CSECT 内で使用してインストールのデフォルトを設定することができますが、個々のプログラムとリンク・エディットされる CEEUOPT CSECT 内で設定するか、C または C++ プログラムのソース内の `#pragma` ステートメントによって設定するか、あるいは PL/I プログラムの `PLIXOPT` ステートメントによって設定するのが最も有効です。例えば、プログラムがスレッド・セーフ標準に従ってコーディングされている場合は、PROGRAM リソース定義を変更せずにそのように定義することができます。または、インストール時に定義される命名標準に従って、プログラム自動インストール出口で適切な属性を使用してインストールされるようにすることができます。

プログラムが Language Environment 準拠のコンパイラを使用してコンパイルされている場合、CICSVAR を、Language Environment 準拠アセンブラ言語、PL/I、COBOL、ならびに C および C++ プログラム (XPLINK オプションが使用されているか否かにかかわらずコンパイルされているもの) に使用することができます。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一度実行すると、CICSVAR による指定変更が適用された設定が `INQUIRE PROGRAM` コマンドによって表示されます。

CICSVAR の有効値は、QUASIRENT、THREADSAFE、REQUIRED、および OPENAPI です。

#### **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=REQUIRED**

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

以下に、CEEUOPT CSECT でコーディングされた Language Environment のランタイム・オプション ENVAR の例を示します。

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

このコードをアセンブルしてロード・モジュールにリンク・エディットしてから、CEEUOPT ロード・モジュールを、Language Environment でサポートされている任意の言語のプログラムとリンク・エディットすることができます。

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・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 (Language Environment) 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、684 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 の詳細情報については、「z/OS Language Environment プログラミング・ガイド」を参照してください。

---

## 第 10 章 PL/I アプリケーションの開発

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。

アプリケーション・プログラミング言語に関する CICS サポートの変更点には、CICS Transaction Server for z/OS, バージョン 5 リリース 5 でサポートされている PL/I コンパイラーと、z/OS におけるそれらの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 5 リリース 5 の資料において PL/I に言及する場合はすべて、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) オプションを指定した 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) オプション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。結果として、追加のコードが生成され、パフォーマンスが影響を受ける場合があります。

---

## 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される PL/I プログラムには、いくつかの制約事項および要件が適用されます。

## 使用できない関数およびステートメント

- 以下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

- 以下のマルチタスク・オプションを使用することはできません。

EVENT

PRIORITY

TASK

- 以下の 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、ならびに端末との通信のため、EXEC CICS コマンドが提供されています。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 キーワードを使用する際には、そのような変数名が暗黙に生成されないように注意する必要があります。

- 小文字を使用できる PROCEDURE 名を除き、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(...,...))
```

この呼び出しは、コンパイラーによって拒否されます。

## 64 ビット・アドレッシング

64 ビット・アドレッシング・モードは、PL/I プログラムについてはサポートされていません。

## 64 ビット常駐

CICS では、64 ビット常駐モード (RMODE(64)) はサポートされておらず、すべての RMODE(64) プログラムが RMODE(31) として処理されます。つまり、RMODE(64) プログラムは、64 ビット (2 GB 境界より上) ストレージではなく、31 ビット (16 MB 境界より上) ストレージにロードされます。

---

## PL/I アプリケーションでの言語環境プログラム (Language Environment) 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、687 ページの『言語環境プログラム (Language Environment) 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 (Language Environment) 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** ストリングに NOSTAE も NOSPIE も指定されていないことを確認する必要があります。これらのいずれかを指定すると、言語環境プログラム (Language Environment) が TRAP (OFF) に設定されてしまうためです。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を使用してコンパイルされた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、Enterprise PL/I for z/OS の製品情報で、関連のバージョンの「*Compiler and Runtime Migration Guide*」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。

サンプルのユーザー条件処理ルーチン CEEWUCHA が、実行時に使用可能であることを (例えば、STEPLIB 連結または LPA を使用して) 確認します。プログラムの自動インストールを使用するのではなく、ご使用の CICS 領域の CICS システム定義データ・セット (CSD) に条件処理ルーチンを定義します。

---

## 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。VisualAge® PL/I では、その制約事項の多くが除去されています。Enterprise PL/I for z/OS の製品情報で、関連バージョンの「*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の AMODE にアドレッシング可能でなければならない。

---

## 第 11 章 Java アプリケーションの開発

CICS サービスを使用し、CICS 制御下で実行される Java アプリケーション・プログラムを作成できます。IBM CICS SDK for Java を使用すると、JCICS クラス・ライブラリーを使用して CICS リソースにアクセスし、他の言語で作成されるプログラムと対話するアプリケーションを開発できます。さまざまなプロトコルおよびテクノロジー (Web サービスや CICS Transaction Gateway など) を使用して、Java プログラムに接続することもできます。

CICS は、Java アプリケーションをサポートするためのツールおよびランタイム環境を提供します。IBM CICS SDK for Java は、Eclipse ベースのツールであり、Java アプリケーションを開発し、CICS にデプロイするためのサポートを提供します。CICS リソースやサービスにアクセスするアプリケーションを開発するための JCICS クラス・ライブラリーが含まれています。例えば、VSAM ファイル、一時データ・キュー、および一時記憶にアクセスできます。また、JCICS を使用して、他の言語 (COBOL や C など) で作成された CICS アプリケーションにリンクすることもできます。IBM CICS SDK for Java には、CICS 用の Java アプリケーションの開発の初心者が始めるようにする一連のサンプルが含まれています。

---

### CICS について必要な知識

CICS は、ユーザーが要求によってアプリケーションを実行するためのサービスを提供するトランザクション処理のサブシステムです。多数のユーザーが、同じファイルとプログラムを使用する同じアプリケーションを同時に実行する要求を実行依頼することが可能です。CICS は、リソースの共用、データの保全性、実行の優先順位付けを管理する一方で、短い応答時間を維持します。

CICS アプリケーションは、製品オーダーの処理や会社の給与計算の準備などの業務を連携して実行する、関連したプログラムの集合です。CICS アプリケーションは、CICS 制御下で実行され、CICS サービスとインターフェースを使用してプログラムとファイルにアクセスします。

CICS アプリケーションを実行するには、トランザクション 要求を実行依頼します。CICS では、トランザクションという用語に特別な意味があります。CICS での意味と、業界でより一般的に使用される意味との違いについては、700 ページの『CICS トランザクション』を参照してください。トランザクションの実行は、必要な機能を実装する 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 用の Java アプリケーションを開発するには、CICS プログラム、トランザクション、およびタスク間の関係を理解する必要があります。これらの用語は、CICS 資料全体で使用され、多くのプログラミング・コマンドで表示されます。また、ランタイム環境において CICS が Java アプリケーションを処理する方法も理解しておく必要があります。

## CICS 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。

要求は通常、ユーザーによって端末で行われます。ただし、Web ページから、リモート・ワークステーション・プログラムから、または別の CICS 領域のアプリケーションから行われる場合があります。もしくは、事前定義された時点に自動的にトリガーされる場合もあります。CICS Web サポートの概念と構造および CICS 外部インターフェースの概要で、CICS トランザクションのさまざまな実行方法を説明します。

単一トランザクションは、1 つ以上のアプリケーション・プログラム で構成されます。これらのアプリケーション・プログラムが実行されると、必要な処理を実行します。

ただし、CICS ではトランザクション という用語は、単一イベントと、同じタイプの他のすべてのトランザクションの両方を意味するのに使用されます。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。この定義により、トランザクション・タイプに名前 (トランザクション ID、すなわち TRANSID) が指定され、実行される作業に関する複数の項目が CICS に指示されます。例えば、最初にどのプログラムを呼び出すか、トランザクションの実行全体でどの種類の認証が必要であるかなどです。

トランザクションを実行するには、その TRANSID を CICS に対して送信します。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開始します。

トランザクション という用語は、リカバリー単位、または CICS で作業単位 と呼ばれるものを記述するために、IT 業界で広く使用されています。一般に、これはリカバリー可能な完全な論理オペレーションです。プログラムされたコマンド、またはシステム障害の発生によって、トランザクション全体をコミットまたはバックアウトすることができます。多くの場合、CICS トランザクションの有効範囲は単一の作業単位でもありますが、CICS 資料を読むときは、意味の違いを認識する必要があります。

## CICS タスク

タスクは、トランザクションを実行する単一のインスタンスです。

CICS では、タスク という用語に特別な意味があります。CICS はトランザクションの実行要求を受け取ると、トランザクション・タイプの実行のこの 1 つのインスタンスに関連した新規タスクを開始します。すなわち、CICS タスクは、通常は特定のユーザーのために、データの独自の専用セットを使用した、トランザクションの 1 つの実行です。また、タスクをスレッド と見なすこともできます。タスクは、優先順位と準備度にしたがって CICS によってディスパッチ されます。トランザクションが完了すると、タスクは終了します。

## CICS アプリケーション・プログラム

Java プログラムでは、CICS 用の Java クラス・ライブラリー (JCICS) を使用して、CICS サービスにアクセスし、他の言語で作成されたアプリケーション・プログラムにリンクすることができます。

CICS アプリケーション・プログラムは、COBOL、C、C++、Java、PL/I、またはアセンブラ言語で作成できます。処理ロジックのほとんどは、標準的な言語ステートメントで表されますが、CICS サービスを要求するには、用意されてい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アプリケーションで使用します。COBOL、C、C++、PL/I、またはアセンブラ・プログラムは、**EXEC CICS** アプリケーション・プログラミング・インターフェースまたは C++ クラス・ライブラリーを使用できます。Java プログラムは、JCICS クラス・ライブラリーを使用します。JCICS については、715 ページの『CICS 用 Java クラス・ライブラリー (JCICS)』で説明しています。

## CICS サービス

Java プログラムは、JCICS プログラミング・インターフェースを介して、データ管理サービス、通信サービス、作業単位サービス、プログラム・サービス、および診断サービスの各 CICS サービスにアクセスできます。

CICS サービス・マネージャーの名称には、通常は「管理」または「制御」という語が含まれています (例えば、「端末管理」、「プログラム制御」など)。これらの用語は、CICS 資料で幅広く使用されています。

### データ管理サービス

CICS が提供するデータ管理サービスは、次のとおりです。

- 仮想記憶アクセス方式 (VSAM) データ・セットにアクセスする際の、保全性のあるレコード・レベル共用。データのバックアウト (トランザクション障害またはシステム障害の場合)、または順方向リカバリー (メディア障害の場合) をサポートするために、CICS はアクティビティをログに記録します。CICS ファイル制御は、VSAM データを管理します。

また、CICS は 2 つの専有ファイル構造も実装し、それらを操作するためのコマンドを提供します。

#### 一時記憶

一時記憶 (TS) は、複数のトランザクションからデータを容易に使用可能にする手段です。データは、プログラムからの要求に応じて作成されるキューに保持されます。キューには順次にアクセスするか、または項目番号でアクセスすることができます。

一時記憶域キューは、メインメモリーに常駐することも、ストレージ・デバイスに書き込むこともできます。

一時記憶域キューは、名前付きのスクラッチパッドと見なすことができます。

#### 一時データ

一時データ (TD) も複数のトランザクションから使用可能であり、キューに保持されます。ただし、TS キューとは異なり、TD キューは事前定義する必要があり、順次にしか読み取れません。各項目は、読み取られるとキューから除去されます。

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。一時データ・キューは、特定数の項目が書き込まれると、特定トランザクション

を開始するトリガーの役目をするように定義できます。例えば、起動したトランザクションによってそのキューを処理できます。

- データベース製品とのインターフェースを使用した、他のデータベース (Db2 を含む) 内のデータへのアクセス。

## 通信サービス

CICS は、SNA と TCP/IP プロトコルを使用して、さまざまな端末 (ディスプレイ、プリンター、およびワークステーション) へのアクセスを可能にするコマンドを備えています。CICS 端末管理により、SNA ネットワークおよび TCP/IP ネットワークを管理できます。

拡張プログラム間通信機能 (APPC) コマンドを使用して、SNA プロトコルを使用してリモート・システム内の他のプログラムを開始し、通信するプログラムを作成できます。CICS APPC は、ピアツーピア分散アプリケーション・モデルを実装します。

次の CICS 専有通信サービスが提供されます。

### 機能シップ

リモート CICS 領域で既存のものとして定義されるリソース (ファイル、キュー、およびプログラム) にアクセスするプログラム要求は、自動的に CICS によって専有領域に転送されます。

### 分散プログラム・リンク (DPL)

リモート CICS 領域で既存のものとして定義されるプログラムに対するプログラム・リンク要求は、自動的に専有領域に転送されます。CICS は、分散アプリケーションの保全性を維持するためのコマンドを提供します。

### 非同期処理

CICS は、プログラムが同じ CICS 領域またはリモート CICS 領域内の別のトランザクションを開始し、オプションとしてデータをそのトランザクションに渡すことを可能にするコマンドを提供します。新しいトランザクションは、新しいタスク内で独立してスケジュールされます。この機能は、他のソフトウェア製品によって提供される *fork* 操作に似ています。

### トランザクション・ルーティング

リモート CICS 領域で既存のものとして定義されるトランザクションを実行する要求は、自動的に専有領域に転送されます。ユーザーへの応答は、要求を受け取った領域に返されます。

## 作業単位サービス

CICS がトランザクションを実行する新しいタスクを作成すると、新しい作業単位 (UOW) が自動的に開始されます。BEGIN コマンドは必要ないため、CICS はこのコマンドを提供しません。CICS トランザクションは常にトランザクション内で実行されます。

CICS は、実行されたりカバリー可能な作業をコミットまたはロールバックするために SYNCPOINT コマンドを提供します。同期点が完了すると、CICS は自動的に別

の作業単位を開始します。SYNCPOINT コマンドを発行せずにプログラムを終了すると、CICS は暗黙的な同期点を取り、トランザクションをコミットしようとしません。

コミットの有効範囲には、リカバリー可能として定義されたすべての CICS リソース、および CICS によって提供されたインターフェースを使用してインタレストに登録した他のすべてのリソース・マネージャーが含まれます。

## プログラム・サービス

CICS は、プログラムが別のプログラムにリンクするか、制御を転送してから戻ることを可能にするコマンドを提供します。

## 診断サービス

CICS が提供するコマンドを使用して、プログラムをトレースし、ダンプを作成できます。

## CICS での Java ランタイム環境

CICS は、スレッド・セーフ Java アプリケーションを実行するための JVM サーバー環境を提供します。スレッド・セーフではないアプリケーションは、JVM サーバーを利用することはできません。

JVM サーバーは、単一の JVM で複数のタスクを実行できるランタイム環境です。この環境はそれぞれの Java タスクで必要な仮想ストレージの量を削減し、CICS が多数のタスクを同時に実行できるようにします。

CICS タスクは、同じ JVM サーバー・プロセス内のスレッドとして並列で実行されます。JVM は、複数のアプリケーションを同時に実行している可能性のあるすべての CICS タスクにより共用されます。すべての静的データおよび静的クラスも共用されます。このため、CICS で JVM サーバーを使用するには、Java アプリケーションがスレッド・セーフである必要があります。各スレッドは T8 TCB で実行され、JCICS API を使用して CICS サービスにアクセスできます。

アプリケーション内で `System.exit()` メソッドを使用しないでください。このメソッドにより、JVM サーバーと CICS の両方がシャットダウンし、アプリケーションの状態と可用性に影響を与えます。

## マルチスレッド・アプリケーション

新しいスレッドを開始したり、スレッドを開始するライブラリーを呼び出すためにアプリケーション・コードを作成できます。アプリケーションにスレッドを作成する場合、OSGi レジストリーから汎用 `ExecutorService` を使用する方式が推奨されています。アプリケーションが JVM サーバーで実行されている場合には、`ExecutorService` は自動的に `CICSExecutorService` を使用して CICS スレッドを作成します。この方式により、アプリケーションを他の環境に移植することが容易になり、特定の JCICS API メソッドを使用する必要がなくなります。

しかし、CICS に固有のアプリケーションを作成している場合は、JCICS API 内の `CICSExecutorService` クラスを使用して、新しいスレッドを要求することができます。

どちらの方法を選択した場合にも、新しく作成されたスレッドは CICS タスクとして実行され、CICS サービスにアクセスできます。JVM サーバーが使用不可にされると、CICS は、JVM で実行中の CICS タスクすべてが終了するまで待機します。ExecutorService クラスまたは CICSExecutorService クラスを使用することにより CICS は実行中のタスクを認識するので、アプリケーションの作業完了後に JVM サーバーをシャットダウンさせることが可能になります。

JCICS オブジェクトは、そのオブジェクトを作成したタスクでのみ使用してください。それらのオブジェクトをタスク間で共用しようとすると、予測不能な結果をもたらす可能性があります。

CICS ExecutorService の使用の詳細については、718 ページの『スレッド』を参照してください。

## JVM サーバーの始動とシャットダウン

静的データは JVM サーバーで実行中のすべてのスレッドで共用されるため、静的データを初期化して JVM のシャットダウン時に適切な状態にするための OSGi バンドル・アクティベーター・クラスを作成できます。JVM サーバーは、例えば、JVM の構成を変更したり、問題を修正したりするために、管理者が使用不可にするまで実行されます。バンドル・アクティベーター・クラスを提供することにより、アプリケーションにとって適切な状態を確実に設定できます。CICS にはタイムアウトがあります。このタイムアウトは、JVM サーバーの開始または停止を続行するまでにこれらのクラスが完了するのを待機する時間を指定します。開始クラスと終了クラスで JCICS を直接使用することはできません。ただし、開発者が CICSExecutorService.runAsCICS() API を使用して、アクティベーターから新しい JCICS 対応スレッドを開始することは可能です。JCICS コマンドはすべて、インストール・コマンドを実行したユーザー ID の権限で実行されます。そのため、管理者は、バンドル・アクティベーターをインストールする前に、その中で使用されるリソースを把握しておく必要があります。

---

## IBM CICS SDK for Java を使用したアプリケーションの開発

CICS Explorer には、IBM CICS SDK for Java およびオプションで IBM CICS SDK for Java EE and Liberty が含まれています。この IBM CICS SDK for Java は、OSGi および Web プロジェクトのサポートを含め、Java アプリケーションを開発して CICS にデプロイするための環境を提供します。

IBM CICS SDK for Java を使用して、OSGi 仕様に準拠するように、新しいアプリケーションを作成したり、既存の Java アプリケーションを再パッケージ化したりできます。OSGi には、コンポーネント・モデルを使用してアプリケーションを開発し、それらのアプリケーションを OSGi バンドルとしてフレームワークにデプロイするためのメカニズムが用意されています。OSGi バンドルは、アプリケーションのデプロイメントの単位であり、バージョン情報、依存関係、およびアプリケーション・コードが入っています。OSGi の主な利点は、Java パッケージと呼ばれる明確に定義されたインターフェースを介してのみアクセスされる再使用可能コンポーネントから、アプリケーションを作成できることです。そのため、OSGi サービスを使用して、Java パッケージにアクセスできます。また、Java アプリケーションのライフサイクルと依存関係をきめ細かく管理することもできます。OSGi を使用したアプリケーションの開発については、OSGi Allianceを参照してください。

IBM CICS SDK for Java を使用すると、サポートされる任意の CICS リリースで動作する Java アプリケーションを開発できます。CICS のリリースが異なると、サポートされる Java のバージョンも異なります。また、後のリリースでは、より多くの CICS 機能をサポートするように JCICS API が拡張されています。誤ったクラスの使用を防ぐために、IBM CICS SDK for Java には、ターゲット・プラットフォームまたはプロジェクト・ライブラリーをセットアップする機能が用意されています。どのリリースの CICS 用に開発するかを定義できます。定義すると、IBM CICS SDK for Java は、使用できない Java クラスを自動的に非表示にします。

Liberty JVM サーバーを使用している場合、IBM CICS SDK for Java は、動的 Web プロジェクトおよび OSGi アプリケーション・プロジェクトの処理に役立ちます。JCICS を使用して CICS サービスにアクセスする、最新の Web レイヤーおよびビジネス・ロジックを備えたアプリケーションを作成できます。別の OSGi バンドルからのコードに Web アプリケーションでアクセスする必要がある場合、OSGi アプリケーション・プロジェクト (EBA ファイル) としてそれをデプロイする必要があります。アプリケーション・マニフェストの中にもう一方の OSGi バンドルを含めるか、共通ライブラリーとして Liberty bundle\_repository の中にもう一方のバンドルをインストールする必要があります。アプリケーションへの入り口点を提供し、それを URL として Web ブラウザーに公開するには、Web 使用可能 OSGi バンドル (WAB ファイル) を EBA ファイルに含める必要があります。

## ターゲット・プラットフォームの更新

サード・パーティーの Java クラスを Eclipse 開発環境のターゲット・プラットフォームに追加できます。

### 始める前に

サード・パーティーの Java クラスが入っている JAR ファイルを OSGi プラグインとして使用できることと、そのファイルがローカル・ワークステーションにコピーされていることを確認します。CICS TS V5.5 と Java EE および Liberty ターゲット・プラットフォームが既に構成されていることを確認します。

### このタスクについて

CICS Explorer Software Development Kit (SDK) には、CICS API や Web API を使用するために必要な Java クラスしか用意されていません。その他のインターフェースのサポートを追加するには、サード・パーティーの JAR が入っている OSGi プラグインを Eclipse ターゲット・プラットフォームに追加する必要があります。そうすれば、そのターゲット・プラットフォームを使用するすべてのアプリケーションで、エクスポートしたパッケージを使用できるようになります。

### 手順

1. Eclipse で「ウィンドウ」 > 「設定」 > 「ターゲット・プラットフォーム」を選択します。
2. ターゲット定義から CICS TS V5.5 と Java EE および Liberty ターゲット・プラットフォームを選択します。

3. 「編集」をクリックし、「ロケーション」タブにある「追加」をクリックします。サード・パーティーのバンドル JAR が入っているディレクトリを表示します。
4. 「次へ」をクリックします。OSGi プラグインの内容が表示されます。
5. 「終了」>「終了」>「OK」をクリックします。

## タスクの結果

OSGi 環境を正常に更新して、Java アプリケーション開発に必要なサード・パーティーの OSGi バンドルと CICS の OSGi バンドルを組み込むことができました。

## 次のタスク

Java アプリケーションを CICS JVM サーバーにデプロイして、サード・パーティーの JAR を、OSGi ミドルウェア・バンドルとして追加するか、Liberty 共用バンドル・リポジトリに追加します。詳しくは、OSGi ミドルウェア・バンドルの更新および server.xml の手動調整を参照してください。

## プラグイン・プロジェクトの作成

CICS Java アプリケーションを、OSGi 仕様に準拠する Eclipse プラグイン・プロジェクトとして作成します。OSGi サービス・プラットフォームは、コンポーネント・モデルを使ってアプリケーションを開発し、それらのアプリケーションを OSGi バンドルとしてフレームワークに配置するためのメカニズムを提供します。

プラグイン・プロジェクトは OSGi バンドルであり、CICS Java アプリケーションで必要とされるすべてのファイルと成果物が含まれます。プラグイン・プロジェクトは、CICS バンドル・プロジェクトに組み込まれてから、ホスト・システムにエクスポートされます。

## 始める前に

ターゲット・プラットフォームを設定する必要があります。詳しくは、ターゲット・プラットフォームの更新を参照してください。

## このタスクについて

このタスクでは、新しいプラグイン・プロジェクトを作成します。特に記載されている場合を除き、設定はデフォルト値のままにしておくことができます。プロジェクトを作成する場合、マニフェストを編集して、JCICS API の依存関係を追加する必要があります。

## 手順

1. Eclipse メニュー・バーで、「ファイル」>「新規」>「プロジェクト (Project)」をクリックして、「新規プロジェクト」ウィザードを開きます。
2. 表示されるリストから、「プラグイン・プロジェクト (Plug-in Project)」を選択し、「次へ」をクリックして「新規プラグイン・プロジェクト (New Plug-in Project)」ウィザードを開きます。
3. 「プロジェクト名」フィールドに、プロジェクトの名前 (com.ibm.cics.example.accounting など) を入力します。「ターゲット・プラ

ットフォーム」セクションで、「**OSGi フレームワーク (an OSGi framework)**」を選択して、メニューから「標準」を選択します。「次へ」をクリックします。「コンテンツ (Content)」ペインが表示されます。

4. 「バージョン」フィールドで、バージョン番号の最後から『.qualifier』を削除します。
5. 「実行環境」フィールドで、CICS ランタイム・ターゲット・プラットフォームの実行環境に一致する Java のレベル (**JavaSE-1.7** など) を選択します。
6. 「アクティベーターを生成 (**Generate an activator**)」チェック・ボックスのチェック・マークを外して、「終了」をクリックします。「パッケージ・エクスプローラー (Package Explorer)」ビューで、新しいプラグイン・プロジェクトが作成されます。
7. 必須: ここでプラグイン・マニフェスト・ファイルを編集して、JCICS および `com.ibm.record` API の依存関係を追加する必要があります。これらのステップを実行しない場合、バンドルをエクスポートしてインストールできますが、動作しません。
  - a. 「パッケージ・エクスプローラー」ビューで、プロジェクト名を右クリックして、「プラグイン・ツール」 > 「マニフェストを開く」をクリックします。マニフェスト・ファイルがマニフェスト・エディターで開きます。
  - b. 「依存関係」タブを選択し、「インポート済みパッケージ」セクションで、「追加」をクリックします。「パッケージの選択 (Package Selection)」ダイアログが開きます。
  - c. パッケージ `com.ibm.cics.server` を選択して、「OK」をクリックします。パッケージが「インポートされたパッケージ (Imported Packages)」リストに表示されます。
  - d. オプション: アプリケーションで必要な場合には、前述のステップを繰り返して、以下のパッケージをインストールします。

#### **com.ibm.record**

VisualAge に付属の Java レコード・フレームワークから  
ByteBuffer を使用するレガシー・プログラム用の Java API。以前  
は `dfjcics.jar` ファイル内にありました。

- e. 「ファイル」 > 「保管」を選択してマニフェスト・ファイルを保管します。

## タスクの結果

JCICS API の依存関係を含む新しいプラグイン・プロジェクトが作成されました。

## 次のタスク

これで、CICS Java アプリケーションを作成できます。CICS 用の Java アプリケーションの開発が初めての場合は、IBM CICS SDK for Java とともに提供される JCICS のサンプルを使用して開始することができます。

注: アプリケーションを開発したら、CICS-MainClass 宣言をマニフェスト・ファイルに追加し、アプリケーションで使われるクラスを宣言する必要があります。詳しくは、関連リンクを参照してください。

プラグインの開発について詳しくは、Eclipse のヘルプ文書の *Plug-in Development Environment (PDE)* の『*User Guide*』セクションを参照してください。

Java アプリケーションが終了したら、CICS バンドル内のそのアプリケーションを zFS にデプロイする必要があります。CICS バンドルは、1 つ以上のプラグインを含むことができ、CICS のアプリケーションの配置の単位です。

## プラグイン・プロジェクトのマニフェスト・ファイルの更新

JCICS アプリケーションを開発する場合、または既存のアプリケーションをプラグイン・プロジェクトにパッケージ化する場合には、プロジェクトのマニフェスト・ファイルを更新して CICS-MainClass ヘッダーを含める必要があります。

### このタスクについて

CICS-MainClass ヘッダーは、LINK、START、または RUN コマンド、あるいはトランザクションの初期プログラムで呼び出せるクラスを宣言するために使用します。

CICS メインクラスを宣言する OSGi バンドルに、「遅延」アクティベーション・ポリシーを使用しないでください。CICS は、OSGi フレームワークで OSGi バンドルが開始されると、すぐにそれらをアクティブにします。この宣言は、手動でマニフェスト・ファイルに追加する必要があります。

### 手順

1. マニフェスト・ファイルをエディターでまだ開いていない場合は、パッケージ・エクスプローラー・ビューでプロジェクト名を右クリックして、「プラグイン・ツール」>「マニフェストを開く」をクリックします。 マニフェスト・ファイルがマニフェスト・エディターで開きます。
2. 「**MANIFEST.MF**」タブを選択します。ファイルの内容が表示されます。
3. 次の宣言をマニフェスト・ファイルに追加します。 CICS-MainClass:packagename.classname ここで:

**packagename**

完全修飾 Java パッケージ名です。

**classname**

アプリケーションで使用されるクラスの名前です。複数のクラスを使用する場合、**packagename.classname** エレメントをコンマで区切って繰り返します。

CICS-MainClass ヘッダーでは、別名を使用できます。例えば、宣言 CICS-MainClass: examples.hello.HelloCICSWorld; alias=greeting は、別名 greeting を CICS-MainClass examples.hello.HelloCICSWorld に割り当てます。プログラムを CICS に定義する際は、クラス名の代わりに別名 greeting を使用します。同じプログラムの複数のバージョンがあり、それぞれが同じクラス名を持つ場合には、別名が便利です。別名を使用することによって、異なるバージョンを識別できます。

以下の例は、HelloCICSWorld および HelloWorld クラスに CICS-MainClass ヘッダーを設定したマニフェスト・ファイルを示しています。

```

Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Hello Plug-in
Bundle-SymbolicName: com.ibm.cics.server.examples.hello
Bundle-Version 1.0.0
Bundle-RequiredExecutionEnvironment: JavaSE-1.7
Import-Package: com.ibm.cics.core.bundle,
 com.ibm.cics.core.model.builders,
 com.ibm.cics.server;version="[1.300.0,2.0.0)"
CICS-MainClass: examples.hello.HelloCICSWorld,
 examples.hello.HelloWorld

```

4. クラス宣言をすべて追加した後、「ファイル」>「保管」を選択してマニフェスト・ファイルを保管します。

## タスクの結果

これで、プラグイン・プロジェクトを CICS バンドルに追加して、zFS に配置できるようになりました。CICS バンドルは、1 つ以上のプラグインを含むことができ、CICS のアプリケーションの配置の単位です。

## 次のタスク

CICS バンドル・プロジェクトを作成します。CICS Explorer 製品資料内の『CICS バンドル・プロジェクトの作成』を参照してください。

## Java EE アプリケーションの作成

CICS Explorer および IBM CICS SDK for Java のヘルプには、アプリケーションを開発して配置する以下のステップの実行方法が詳細に説明されています。

### 手順

1. Java 開発用のターゲット・プラットフォームをセットアップします。 の関連するステップを参照してください。

ターゲット・プラットフォームを使用すると、アプリケーション開発において CICS のターゲット・リリースに適した Java クラスのみを確実に使用できます。

2. Java アプリケーション開発用の OSGi バンドル・プロジェクトまたはプラグイン・プロジェクトを作成します。
  - a. プロジェクトのデフォルト・バージョンは 1.0.0.qualifier です。「バージョン」フィールドで .qualifier を使用する必要がない場合は、バージョン番号の最後から .qualifier を削除します。または、日付/タイムスタンプなどの意味のあるものを設定します。

ベスト・プラクティスを使用して Java アプリケーションを開発します。例えば、OSGi バンドルどうしの従属関係を構成するには、Require-Bundle ではなく Import-Package/Export-Package を設定します。

3. CICS 用の Java アプリケーションの開発に慣れていない場合は、IBM CICS SDK for Java で提供されるサンプルを使用して開発に取り掛かることができます。OSGi Java アプリケーションで JCICS を使用するには、com.ibm.cics.server パッケージをインポートする必要があります。
4. オプション: Liberty では、アプリケーション・プレゼンテーション層を開発するために、動的 Web アプリケーション (WAR) または Web 使用可能 OSGi

バンドル・プロジェクト (WAB) を作成します。動的 Web プロジェクト内でサーブレットや JSP ページを作成できます。また、WAR ファイルの場合、Liberty API バンドルにアクセスできるように、Liberty ライブラリーをビルド・パスに追加する必要もあります。詳細については、751 ページの『開発環境のセットアップ』を参照してください。

5. 以下のようにして、配置用にアプリケーションをパッケージ化します。
  - a. Web 使用可能 OSGi バンドル・プロジェクト (WAB) をデプロイする場合は、OSGi アプリケーション・プロジェクト (EBA) を作成します。
  - b. EBA、EAR ファイル、または Web アプリケーション (WAR ファイル) を参照する CICS バンドル・プロジェクトを 1 つ以上作成します。CICS バンドルは、CICS におけるアプリケーションのデプロイメント単位です。一緒に更新および管理する複数の Web アプリケーションを、1 つの CICS バンドル・プロジェクトに入れてください。アプリケーションの配置先の JVMSERVER リソースの名前を知っている必要があります。

CICS リソースも、PROGRAM、URIMAP、および TRANSACTION リソースなどの CICS バンドル・プロジェクトに追加できます。これらのリソースは、Java アプリケーションで動的にインストールされて管理されます。

  - c. オプション: アプリケーションを CICS プラットフォームに配置する場合は、CICS バンドルを参照するアプリケーション・プロジェクトを作成します。CICS では、アプリケーションにより、単一の管理点から CICSplex 全体でアプリケーションをデプロイおよび管理することが可能になります。詳細については、仕組み: アプリケーションを参照してください。
  - d. ツールがエラーを示していない場合でも、OSGi バンドルが使用するパッケージを常に明示的に宣言する必要があります。これを行うには、OSGi バンドル・マニフェストで **Import-Package** バンドル・ヘッダーを追加または更新します。Eclipse などのツールは、明示的な Import が必要なランタイム環境にとって適切ではない可能性のある **javax.\*** パッケージの使用を想定します。
6. アプリケーション・プロジェクトまたは CICS バンドル・プロジェクトをエクスポートして、Java アプリケーションを zFS にデプロイします。あるいは、配置用にソース・リポジトリにプロジェクトを保管することもできます。

## タスクの結果

これで、IBM CICS SDK for Java を使用したアプリケーションの開発とエクスポートが正常に完了しました。

## 次のタスク

アプリケーションを JVM サーバーにインストールします。CICS 内にリソースを作成する権限がない場合は、システム・プログラマーまたは管理者にアプリケーションの作成を依頼します。システム・プログラマーまたは管理者には、エクスポートしたバンドルがある場所とターゲット JVM サーバーの名前を伝える必要があります。

## CICS バンドル・プロジェクトへのプロジェクトの追加

CICS バンドル・プロジェクトを作成すると、マニフェスト・ファイルが META-INF ディレクトリに作成されます。マニフェスト・ファイルを編集して、各種タイプのプロジェクト (動的 Web プロジェクト、エンタープライズ・アプリケーション・プロジェクト、OSGi アプリケーション・プロジェクト、または OSGi バンドル・プロジェクト) を 1 つ以上組み込むことができます。組み込まれたプロジェクトは、ソースであっても、事前ビルドされていても構いません。CICS バンドル・プロジェクトをエクスポートすると、組み込まれているすべてのプロジェクトが、zFS 上の CICS バンドルに含まれます。

### 始める前に

このタスクでは、CICS バンドルにプロジェクトの詳細を追加する方法について説明します。CICS バンドル・プロジェクトを作成していない場合、CICS Explorer 製品資料内の『CICS バンドル・プロジェクトの作成』を参照してください。

### このタスクについて

プロジェクトの詳細を CICS バンドルに追加できます。これを行うには、各種ウィザード (「動的 Web プロジェクトの組み込み」、「エンタープライズ・アプリケーション・プロジェクトの組み込み」、「OSGi アプリケーション・プロジェクトの組み込み」、または「OSGi バンドル・プロジェクトの組み込み」) のいずれかを使用します。ウィザードは、追加されるプロジェクトの詳細を含めるようにバンドル・マニフェスト・ファイルを更新し、プロジェクトを指す .warbundle、.earbundle、.ebabundle、または .osgibundle というファイル拡張子の付いたリソース・ファイルを作成します。

注: OSGi アプリケーションに含まれていない OSGi バンドルを CICS バンドル・プロジェクトに追加するには、出力フォルダーのロケーションを含む build.properties ファイルが必要です。build.properties ファイルの内容の例を以下に示します。

```
source.. = src/
output.. = bin/
bin.includes = META-INF/
```

### 手順

1. 「パッケージ・エクスプローラー」ビューで、更新するバンドル・プロジェクトを右クリックし、「新規」 > 「その他」をクリックして新規ウィザードを開きます。
2. CICS リソース・フォルダーを展開し、「動的 Web プロジェクトの組み込み」、「エンタープライズ・アプリケーション・プロジェクトの組み込み」、「OSGi アプリケーション・プロジェクトの組み込み」、または「OSGi バンドル・プロジェクトの組み込み」をクリックします。「次へ」をクリックします。ウィザードが開き、ワークスペース内の当該タイプのプロジェクトが表示されます。ウィザードには、選択したバンドル・プロジェクト内にあるビルドされたプロジェクト (例えば、JAR、EAR、EBA、および WAR の各ファイル) もすべて表示されます。
3. バンドルに含めるプロジェクトをクリックします。プロジェクトをクリックすると、ウィザードにはシンボル名と、該当する場合にはバージョンが表示されま

す。プロジェクト上にカーソルを置くと、それがビルドされたプロジェクトであるか、またはソース・プロジェクトであるかを識別できます。

4. オプション: OSGi プロジェクトの場合、含めるバージョンまたはバージョン範囲を指定します。
  - 「このバージョンを使用する (**Use this version**)」を選択して、「バージョン」フィールドに示されているような方法で、選択した OSGi プロジェクトの特定のバージョンを指定します。
  - 「バージョン範囲を使用する (**Use version range**)」を選択して、選択した OSGi プロジェクトをエクスポートするときに含める、その OSGi プロジェクトの定義済みのバージョン範囲の最上位バージョンを指定します。デフォルトでは、バージョン範囲には、選択した OSGi プロジェクトのバージョンから次の最上位のメジャー・バージョンまでが含まれます。フィールドとボタンを使用して、別の範囲を指定することもできます。
5. 「JVM サーバー」フィールドに、アプリケーション・コンポーネントを実行する JVM サーバーの名前を入力します。
6. オプション: プロジェクト名に基づいて、作成するリソース・ファイルの名前が生成され、それがウィザードに表示されます。ファイル名を変更する場合は、「戻る」ボタンを使用できます。
7. 「完了 (**Finish**)」をクリックします。

## タスクの結果

プロジェクトのリソース・ファイルがバンドル・プロジェクトに追加され、マニフェスト・ファイルが更新されます。これらのステップを繰り返して、プロジェクトをさらに CICS バンドル・プロジェクトに追加することができます。

## 次のタスク

アプリケーションの CICS バンドル・プロジェクトにリソースを追加できます。例えば、Java アプリケーションを CICS 内の他のアプリケーションで使用できるようにするプログラムを作成できます。

CICS Explorer 製品資料内の『CICS バンドルのデプロイ』で説明されているように、CICS バンドルを z/OS UNIX ファイル・システムにデプロイできます。CICS バンドル・プロジェクトを zFS にエクスポートすると、アプリケーションに必要なすべてのファイルと成果物がコンパイルされてエクスポートされます。

または、CICS バンドル・プロジェクトをクラウド・スタイルのアプリケーション・プロジェクトにパッケージし、CICS プラットフォームにデプロイできます。アプリケーション・プロジェクトを使用すると、アプリケーションを構成するすべての CICS バンドル・プロジェクトをまとめてグループ化し、それを単一ステップでデプロイおよびインストールすることができます。詳しくは、CICS Explorer 製品資料内の『CICS アプリケーション・バインディング・プロジェクトの作成』を参照してください。

## プロジェクト・ビルド・パスの更新

プロジェクト・ビルド・パスの更新方法。

## このタスクについて

動的 Web プロジェクトを使用する場合は、デプロイメント用の WAR ファイル・アーカイブが作成されます。この場合、Eclipse の OSGi フレームワークは使用されないため、サード・パーティーの JAR ファイルをプロジェクト・ビルド・パスに追加する必要があります。この例では、IBM MQ JAR ファイルを使用します。

### 手順

1. Eclipse で Web プロジェクトを選択し、「ビルド・パス」 > 「ビルド・パスの構成」を右クリックします。これにより、「Java のビルド・パス」ウィンドウが表示されます。
2. CICS および Liberty ライブラリーを追加し、「ライブラリーの追加」 > 「Liberty JVM サーバー」 > 「次へ」 > 「終了」をクリックします。
3. 「外部 JAR の追加」をクリックし、前にダウンロードした IBM MQ JAR ファイルが配置されているディレクトリーにナビゲートします。アプリケーションで使用するインポートに応じて、以下の JAR ファイルを選択します。
  - com.ibm.mq.jar
  - com.ibm.mq.jmq1.jar
  - com.ibm.mq.headers.jar

注: ステップ 3 は、IBM MQ のみを使用する場合のオプションです。

### タスクの結果

プロジェクトのビルド・パスに、CICS と IBM MQ の両方の API を使用する Web アプリケーションを開発するための適切なインターフェースが含まれるようになりました。

---

## その他のツールを使用したアプリケーションの開発

IBM CICS SDK for Java は、CICS Java アプリケーションの 1 次開発環境ですが、この SDK の使用は必須ではありません。CICS Java プログラムを作成、パッケージ化、およびデプロイするための独自のスクリプトとツールを作成できます。

CICS には、複数の .jar ファイルが付属して配布されており、Java アプリケーションはこれらのファイルに対してコンパイルする必要があります。CICS ライセンスを使用すると、アプリケーション開発の目的で、これらの .jar ファイルを CICS インストール・ディレクトリーからコピーできます。これらのファイルをまとめたものが JCICS ライブラリーであり、CICS インストール zFS ライブラリーの /lib ディレクトリー内に配置されます。

アプリケーション開発の .jar ファイルは、以下のとおりです

- com.ibm.cics.server.jar
- com.ibm.cics.server.pipeline.jar
- com.ibm.cics.server.invocation.jar
- com.ibm.cics.server.invocation.binding.jar
- com.ibm.cics.server.invocation.annotations.jar
- com.ibm.record.jar

場合によっては、新しい Java API およびメソッドが、PTF によって CICS に追加されることがあります。更新によって導入された API を使用する場合は、コピーされた .jar ファイルをリフレッシュするメカニズムを組み込む必要があります。新しいリリースの CICS をインストールする場合は、フルリフレッシュが必要です。

要確認: Java コンパイル・セットアップだけでなく独自のスクリプトを使用する場合は、OSGi のパッケージ化、Java EE コンポーネント、および CICS バンドル・リソースを考慮する必要があります。IBM CICS SDK for Java は、これらの問題を単純化するように設計されています。

---

## 共用 JVM に関する考慮事項

CICS 内で実行する Java アプリケーションを開発する際は、JVM 内の共用リソースを変更すると、その変更がすべての実行中アプリケーションおよびスレッドによって検出される場合があることに注意してください。開発するアプリケーションにおいて、他のアプリケーションが依存する JVM を予期しない状態のままにしないようにしてください。

考慮すべき重要な点は次のとおりです。

- アプリケーションがデフォルトのタイム・ゾーンをリセットする場合、同じ JVM サーバーを使用する他のアプリケーションは新しいデフォルトのタイム・ゾーンを使用することになりますが、この変更が他のアプリケーションにとって予期されないものである可能性があります。
- 開発するアプリケーションでは **System.exit()** を使用しないでください。**System.exit()** を使用すると、JVM サーバーと CICS の両方がシャットダウンします。
- アプリケーションがスレッド・セーフになるようにしてください。アプリケーションの間で共用する静的変数を慎重に調べて、アプリケーションが相互に悪影響を与えないことを確認する必要があります。固有性を確実にする標準的なパターンは、スレッド・ローカル変数を使用することです。
- オブジェクトが静的変数で参照される場合、それらのオブジェクトはガーベッジ・コレクションの候補にはなりません。JVM サーバーでは、システム・プログラマーが JVM サーバーを使用不可にするまで、すべてのアプリケーションの静的状態が持続します。
- Db2 には異なる複数のアプリケーションが接続する可能性があります。したがって、Db2 での処理が完了したら、接続を閉じることがベスト・プラクティスです。これは、後でタスクが完了した時点で接続が削除される場合にも適用されます。
- `java.net` パッケージに含まれるクラスを使用して作成されたソケットは CICS ドメインのソケットではないため、CICS で管理またはモニターすることはできません。

---

## JCICS を使用した Java の開発

CICS Java クラス・ライブラリー (JCICS) を使用して CICS サービスにアクセスする Java アプリケーションを作成できます。JCICS は、Java において、CICS でサポートされている他の言語 (COBOL など) に提供される **EXEC 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相当するものです。

JCICS を使用すると、CICS リソースにアクセスする Java アプリケーションを作成し、他の言語で作成されたプログラムと統合することができます。**EXEC CICS** API の大部分の機能がサポートされます。このライブラリーは、`com.ibm.cics.server.jar` ファイルで CICS および IBM CICS SDK for Java に提供されています。

保守作業または開発作業で API の追加、API の削除、バグ修正が発生すると、`com.ibm.cics.server` パッケージ (JCICS) のバージョン番号が増えます。リリースの境界では、バージョンの増分は保証されていません。バージョン、およびこれらのバージョンが適用される CICS リリースについては、パッケージ `com.ibm.cics.server` を参照してください。

インポートには、アプリケーションの最小サポート・レベルから (そのレベルを含む) 次の大きな API 変更まで (含まない) の互換可能な範囲を宣言しておくことをお勧めします。例: `Import-Package: com.ibm.cics.server;version="[1.600.0,2.0.0)"`

## CICS 用 Java クラス・ライブラリー (JCICS)

JCICS では、**EXEC CICS** API コマンドのほとんどの機能がサポートされます。

JCICS クラスは、クラス定義から生成される Javadoc で詳しく説明されています。Javadoc は、JCICS Javadoc 情報から入手できます。

### JavaBeans

JCICS の一部のクラスは JavaBeans として使用できます。つまり、Eclipse などのアプリケーション開発ツールでカスタマイズし、直列化し、JavaBeans API を使用して操作することができます。

以下の JavaBeans が JCICS で使用可能です。

- Program
- ESDS
- KSDS
- RRDS
- TDQ
- TSQ
- AttachInitiator
- EnterRequest

これらの Bean はイベントを定義しません。プロパティーとメソッドで構成されます。次の 3 つの方法のいずれかで実行時にインスタンス化することができます。

- クラス自体の `new` メソッドを呼び出す。この方法が優先されます。

- プロパティ値を手動で設定して、クラスの名前の `Beans.instantiate()` を呼び出す。
- 設計時に設定されたプロパティ値を使用して、`.ser` ファイルから `Beans.instantiate()` を呼び出す。

最初の 2 つのオプションのどちらかを選択する場合、プロパティ値 (CICS リソースの名前を含めて) は、実行時に適切な `set` メソッドを呼び出すことによって設定されなければなりません。

## ライブラリー構造

各 JCICS ライブラリー・コンポーネントは、4 つのカテゴリ (インターフェース、クラス、例外、エラー) のいずれかに分類されます。

### インターフェース

一部のインターフェースは、1 組の定数を定義するために提供されます。例えば、`TerminalSendBits` インターフェースは、`java.util.BitSet` の構成に使用できる 1 組の定数を提供します。

### クラス

指定されたクラスは、大部分の JCICS 機能を提供します。API クラスは抽象クラスであり、`ABEND` と例外を除いて、CICS API の一部に対応する、すべてのクラスに共通する初期化を提供します。例えば、`Task` クラスは、CICS タスクに対応する 1 組のメソッドと変数を提供します。

### エラーと例外

Java 言語は、クラス `Throwable` のサブクラスとして例外とエラーの両方を定義します。JCICS は、`Error` のサブクラスとして `CicsError` を定義します。`CicsError` は、重大エラーに使用される他のすべての CICS エラー・クラスのスーパークラスです。

JCICS は、`Exception` のサブクラスとして `CicsException` を定義します。`CicsException` は、(CICS `QIDERR` 条件を表す、`InvalidQueueIdException` などの `CicsConditionException` クラスを含む) すべての CICS 例外クラスのスーパークラスです。

詳しくは、726 ページの『エラー処理と異常終了』を参照してください。

## CICS リソース

プログラムや一時記憶域キューなどの CICS リソースは、該当する Java クラスのインスタンスによって表され、リソースの名前などの各種プロパティの値によって識別されます。

CICS リソースは、CICS Explorer、CEDA トランザクション、または CICSplex SM WUI を使用して定義します。暗黙的なリモート・アクセスを使用するには、リモート・リソースを指すリソースをローカルで定義します。

CICS リソースの定義について詳しくは、CICS リソースを参照してください。

### データを渡すための引数

チャンネルとコンテナを使用するか、通信域 (`COMMAREA`) を使用して、プログラム間でデータを受け渡すことができます。

COMMAREA を使用する場合、一度に渡すデータは 32 KB に制限されます。チャンネルとコンテナを使用する場合、プログラム間で 32 KB より多く渡すことができます。COMMAREA またはチャンネル、およびその他のすべてのパラメーターは、引数として該当するメソッドに渡されます。

メソッドの多くは多重定義されています。すなわち、バージョンが異なれば、取る引数の数か、引数のタイプのどちらかが異なります。引数がないか、最小限の必須引数があるメソッドや、すべての引数があるメソッドがあります。例えば、Program クラスには以下の各種の link() メソッドが含まれています。

#### **link()**

このメソッドは、COMMAREA を使用してデータを受け渡したり、他のオプションを指定したりせず、単純なリンクを行います。

#### **link(com.ibm.cics.server.CommAreaHolder)**

このメソッドは、COMMAREA を使用してデータを受け渡すのみで、他のオプションを指定せず、単純なリンクを行います。

#### **link(com.ibm.cics.server.CommAreaHolder, int)**

このメソッドは、COMMAREA を使用してデータを受け渡し、DATALENGTH 値を使用して COMMAREA 内のデータの長さを指定することにより、分散リンクを行います。

#### **link(com.ibm.cics.server.Channel)**

このメソッドは、チャンネルを使用して 1 つ以上のコンテナ内のデータを受け渡すことによって、リンクを行います。

### 直列化可能クラス

JCICS 直列化可能クラスのリスト。

- AddressResource
- AttachInitiator
- CommAreaHolder
- EnterRequest
- ESDS
- File
- KeyedFile
- KSDS
- NameResource
- Program
- RemotableResource
- Resource
- RRDS
- StartRequest
- SynchronizationResource
- SyncLevel
- TDQ
- TSQ

- TSQType

## Task.out および Task.err

Java 関連の CICS タスクごとに、CICS は、標準出力および標準エラー・ストリームとして使用できる、2 つの Java `PrintWriters` クラスを自動的に作成します。標準出力と標準エラー・ストリームは、`out` と `err` と呼ばれる、Task クラス内のパブリック・フィールドです。

CICS タスクが端末 (この場合、端末は基本機構 と呼ばれます) から駆動される場合、CICS は標準出力および標準エラー・ストリームをタスクの端末にマップします。

タスクに基本機構としての端末がない場合、標準出力および標準エラー・ストリームは `System.out` および `System.err` に送信されます。

## スレッド

JVM サーバー環境では、OSGi フレームワークで実行されているアプリケーションは `ExecutorService` を使用して、CICS タスクで非同期的に実行するスレッドを作成できます。

CICS には、Java `ExecutorService` インターフェースの実装が用意されています。この実装では、JCICS API を使用して CICS サービスにアクセスできるスレッドが作成されます。JVM サーバーは、始動時に CICS `ExecutorService` を OSGi サービスとして登録します。Java `Thread` クラスではなくこのサービスを使用して、JCICS を使用できるタスクを作成してください。

CICS で提供される `ExecutorService` は、スレッドを作成するためにアプリケーションで使用できるように、OSGi フレームワークにおいて優先順位が高いものとして登録されます。アプリケーションは通常、サービスをフィルタリングして特定の実装を使用するのでない限り、最も高い優先順位の `ExecutorService` を使用します。

アプリケーションにスレッドを作成する場合、OSGi レジストリーから汎用 `ExecutorService` を使用する方式が推奨されています。アプリケーションが JVM サーバーで実行されているときに、OSGi レジストリーは自動的に CICS `ExecutorService` を使用して CICS スレッドを作成します。この方式により、アプリケーションは実装環境とは分離されるので、JCICS API メソッドを使用してスレッドを作成する必要がなくなります。

ただし、CICS に固有のアプリケーションを作成している場合は、JCICS API 内の `CICSExecutorService` クラスを使用して、新しいスレッドを要求できます。

## CICSExecutorService

このクラスは `java.util.concurrent.ExecutorService` インターフェースを実装します。CICSExecutorService クラスには `runAsCICS()` という名前の静的メソッドがあり、このメソッドを使用して、Java オブジェクトの `Runnable` または `Callable` を、新しい JCICS 対応スレッドでの実行対象として送信できます。`runAsCICS()`

メソッドは、OSGi レジストリー検索を実行し、アプリケーションの `CICSExecutorService` のインスタンスを取得するユーティリティー・メソッドです。

親 CICS スレッドから `spawn` される作業の場合、新しい CICS タスクが作成され、親から継承される `user ID` と `transaction ID` のタスクの下で実行されます。非 CICS スレッドから `spawn` される作業の場合、デフォルトの `CJSA transaction ID` とデフォルトの `CICS user ID` が使用されます。自分で選択した `transaction ID` の下で新しいタスクが実行されるよう保証するには、`Runnable` または `Callable` オブジェクトが `CICSTransactionRunnable` または `CICSTransactionRunnable` インターフェースを実装している必要があります。

```
CICSExecutorService.runAsCICS(Runnable runnable)
```

```
CICSExecutorService.runAsCICS(Callable callable)
```

### 制約事項

OSGi フレームワーク (Axis2 Java プログラムなど) で実行されていないアプリケーションの場合は、`ExecutorService` を使用できないため、初期アプリケーション・スレッドのみで `JCICS` にアクセスできます。さらに、以下のいずれかのアクションを行う前に、初期スレッド以外のすべてのスレッドが終了している必要があります。

- `com.ibm.cics.server.Program` クラスでの `link` メソッド
- `com.ibm.cics.server.TerminalPrincipalFacility` クラスでの `setNextTransaction(String)` メソッド
- `com.ibm.cics.server.TerminalPrincipalFacility` クラスでの `setNextCOMMAREA(byte[])` メソッド
- `com.ibm.cics.server.Task` クラスでの `commit()` メソッド
- `com.ibm.cics.server.Task` クラスでの `rollback()` メソッド
- `com.ibm.cics.server` クラスから `AbendException` 例外を返す。

## データ・エンコード

JVM は文字エンコードに `CICS` とは異なるコード・ページを使用できます。`CICS` では常に `EBCDIC` コード・ページを使用しなければなりません。JVM では `ASCII` などの別のエンコード方式を使用できます。`JCICS API` を使用するアプリケーションを開発する際には、使用するエンコード方式が正しいものであることを確認する必要があります。

`JCICS API` は、基礎となっている JVM ではなく `CICS` 領域で指定されているコード・ページを使用します。そのために、JVM で別のファイル・エンコード方式を使用する場合は、アプリケーションが別のコード・ページを処理することが必要になります。`CICS` が使用しているコード・ページを判別できるように、`CICS` には以下の `Java` プロパティが用意されています。

- `com.ibm.cics.jvmserver.supplied.ccsid` プロパティは、`CICS` 領域に指定されているコード・ページを返します。デフォルトでは、`JCICS API` は文字エンコードにこのコード・ページを使用します。ただし、この値は JVM サーバー構成でオーバーライドできます。

- **com.ibm.cics.jvmserver.override.ccsid** プロパティは、JVM プロファイル内のオーバーライド値を返します。JCICS API は文字エンコードに、CICS 領域で使用されているコード・ページではなく、この値で示されたコード・ページを使用します。
- **com.ibm.cics.jvmserver.local.ccsid** プロパティは、JVM サーバーで JCICS API が文字エンコードに使用しているコード・ページを返します。

Java アプリケーションにこれらのプロパティを設定して、JCICS のエンコード方式を変更することはできません。コード・ページを変更するには、システム管理者に、JVM プロファイルを更新して JVM システム・プロパティ

**-Dcom.ibm.cics.jvmserver.override.ccsid** を追加するよう依頼する必要があります。

## エンコードの例

`java.lang.String` パラメーターを入力として受け入れる JCICS メソッドはいずれも、データが CICS に渡される前に、正しいコード・ページで自動的にエンコードされます。同様に、JCICS API から返されるすべての `java.lang.String` 値も、正しいコード・ページでエンコードされます。JCICS API は、ほとんどのクラスにヘルパー・メソッドを提供しています。これらのヘルパー・メソッドはアプリケーションのために、ストリングやデータを処理してコード・ページを判別し、設定します。

アプリケーションが `String.getBytes()` メソッドまたは `new String(byte[] bytes)` メソッドを使用する場合、そのアプリケーションは必ず正しいエンコード方式を使用する必要があります。これらのメソッドをアプリケーションで使用する場合、以下のように Java プロパティを使用すれば、データを正しくエンコードすることができます。

```
String.getBytes(System.getProperty("com.ibm.cics.jvmserver.local.ccsid"))
String(bytes, System.getProperty("com.ibm.cics.jvmserver.local.ccsid"))
```

以下の例に、アプリケーションが COMMAREA からフィールドを読み取る場合の JCICS エンコードの使用法を示します。

```
public static void main(CommAreaHolder ca)
{
 //Convert first 8 bytes of ca into a String using JCICS encoding
 String str=new String(ca.getValue(), 0, 8, System.getProperty("com.ibm.cics.jvmserver.local.ccsid"));
}
```

## JCICS API サービスと例

CICS は、Java アプリケーション用のさまざまな API とサービスをサポートしています。EXEC CICS API を介して非 Java プログラムで使用可能なサービスの多くは、Java SDK で提供される標準の Java SE API とともに、JCICS API を介して Java プログラムで使用可能です。

以下のトピックでは、JCICS サービス、および Java 例外処理との統合について詳しく説明しています。Liberty JVM サーバーでは、他の JEE API を使用できません。詳しくは、751 ページの『Liberty JVM サーバーで実行する Java アプリケーションの開発』を参照してください。

## Java プログラムでの CICS 例外処理

CICS で発生した問題を処理するために、CICS の ABEND および例外は Java 例外処理体系に統合されています。

通常のすべての CICS ABEND は単一の Java 例外 `AbendException` にマップされる一方、各 CICS 条件は個別の Java 例外にマップされます。これにより、Java の ABEND 処理モデルは他のプログラミング言語と同じようになります。つまり、すべての ABEND について単一のハンドラーに制御が与えられ、そのハンドラーは特定の ABEND を照会してから、処理内容を決定する必要があります。

条件を表す例外は、CICS 自体によってキャッチされると ABEND になります。

Java 例外処理は、他の言語の ABEND および条件処理に完全に統合されるため、ABEND は、言語に依存しない標準的な方法で、Java プログラムと非 Java プログラム間に波及することができます。条件は ABEND にマップされてから、その条件の原因となるか、またはその条件を検出したプログラムを終了します。

ただし、他のプログラミング言語の ABEND 処理モデルには、次のような複数の相違点があります。これらの相違点は、Java 例外処理体系の性質や、Java API の基礎となる一部のテクノロジーの実装に起因します。

- 他のプログラミング言語では処理できない ABEND を、Java プログラムでキャッチできます。これらの ABEND は通常、同期点処理時に発生します。これらの ABEND が Java アプリケーションを中断しないようにするために、チェックなし例外の拡張にマップされます。したがって、これらの ABEND の宣言もキャッチも必要ありません。
- プログラム終了などの複数の内部 CICS イベントも、Java 例外にマップされるので、Java アプリケーションでキャッチできます。また、正常な状態を中断しないように、これらのイベントはチェックなし例外の拡張にマップされ、キャッチも宣言も必要ありません。

例外の以下の 3 つのクラス階層が CICS に関連しています。

1. `CicsError` は `java.lang.Error` の拡張で、`AbendError` および `UnknownCicsError` の基本クラスです。
2. `CicsRuntimeException` は `java.lang.RuntimeException` の拡張で、次のクラスにより拡張されます。

### **`AbendCancelException`**

CICS ABEND CANCEL を表します。

### **`AbendException`**

通常の CICS ABEND を表します。

### **`EndOfProgramException`**

リンク先プログラムが通常どおりに終了したことを示します。

3. `CicsException` は `java.lang.Exception` の拡張で、次のサブクラスがあります。

### **`CicsConditionException`**

すべての CICS 条件の基本クラス。

## CICS エラー処理コマンド:

Java ではどのように **EXEC CICS** エラー処理コマンドがサポートされるかについて説明します。

721 ページの『Java プログラムでの CICS 例外処理』で説明されているように、CICS 条件処理は Java 例外処理体系に統合されています。Java では同等の **EXEC CICS** コマンドが以下のようにサポートされます。

### HANDLE ABEND

CICS でサポートされる任意の言語のプログラムによって生成された ABEND を処理するには、catch 節に `AbendException` を指定して、Java の try-catch ステートメントを使用します。

### HANDLE CONDITION

PGMIDERR などの特定の条件を処理するには、適切な例外の名前を指定した catch 節を使用します。この場合は、`InvalidProgramException` です。あるいは、すべての CICS 条件をキャッチする場合は、`CicsConditionException` を指定した catch 節を使用します。

### IGNORE CONDITION

このコマンドは、Java アプリケーションでは適切ではありません。

### POP HANDLE および PUSH HANDLE

これらのコマンドは、Java アプリケーションでは適切ではありません。CICS ABEND および条件を表すために使用される Java 例外は、スコープ内の catch ブロックによってキャッチされます。

## CICS 条件:

Java での条件処理モデルは、他の CICS プログラミング言語でのモデルとは異なります。

COBOL では、条件ごとに条件処理ラベルを定義できます。その条件が CICS コマンドの処理中に発生する場合、制御はラベルに転送されます。

C および C++ では、条件に例外処理ラベルを定義することはできません。条件を検出するために、各 CICS コマンドの後に、EIB 内の RESP フィールドが検査されなければなりません。

Java では、CICS コマンドから返された条件はいずれも Java 例外にマップされます。すべての CICS コマンドを try-catch ブロックに組み込んで、条件ごとに特定の処理を行うか、特定の例外が適切でない場合は、単一の null catch 節を持つことができます。または、条件を波及させて、より大きいスコープで catch 節によって処理できるようにすることができます。

## Java Web アプリケーションでの CICS 例外処理

Liberty Web コンテナが CICS アプリケーションで発生する問題に対処するために、CICS ABEND および例外が Java 例外処理体系に組み込まれています。Web アプリケーションで処理されない Java 例外は Web コンテナによってキャッチされ、サーブレット例外処理プロセスを起動します。この処理の一部として、コミットされていない CICS 作業単位は CICS によってロールバックされます。

HttpServlet インターフェースを拡張するすべての Java Web アプリケーションは、HttpServlet インターフェースに定義されているように、IOException や ServletException 以外のすべてのチェック例外を処理する必要があります。チェック例外は、未処理の CICS 条件を表す

com.ibm.cics.server.CicsConditionException のサブクラスのすべてに含まれます。したがって、CICS 条件をキャッチするすべての例外処理コードは、作業単位をロールバックする必要があります。例に示されているように、Task オブジェクトで rollback() メソッドを使用して明示的に同期点ロールバックを呼び出すか、AbendException をスローするエラー条件をすべて識別する必要があります。

```
try
{
 TSQ tsqQ = new TSQ();
 tsqQ.setName("tsq1");
 tsqQ.writeString("input data");

} catch (IOException e) {
 // Log error
 try
 {
 Task.getTask().rollback();

 } catch (InvalidRequestException e1) {
 throw new RuntimeException(e1);
 }
 }
}
```

Java のチェックなし例外は、java.lang.RuntimeException のサブクラスであり、Web アプリケーションを含む任意の Java アプリケーションでスローすることができます。これらのチェックなし例外には、com.ibm.cics.server.AabendException および com.ibm.cics.server.AabendCancelException が含まれます。したがって、AbendException をスローするか、トランザクションの異常終了を処理しない Web アプリケーションは、サーブレット例外処理プロセスおよび関連する作業単位のロールバック処理を起動します。

Java Transaction API (JTA) を使用して作業単位をコミットする Web アプリケーションは、Liberty トランザクション・マネージャーの制御に従ってコミットされます。詳細については、766 ページの『Java Transaction API (JTA)』を参照してください。

## JCICS 例外マッピング

Java では、CICS コマンドによって戻される条件は、Java 例外にマップされます。

表 54. Java 例外マッピング

| CICS 条件      | Java 例外                                            |
|--------------|----------------------------------------------------|
| ALLOCERR     | AllocationErrorException                           |
| CBIDERR      | InvalidControlBlockIdException                     |
| CCSIDERR     | CCSIDErrorException                                |
| CHANNELERR   | ChannelErrorException                              |
| CONTAINERERR | ContainerErrorException                            |
| DISABLED     | FileDisabledException<br>ResourceDisabledException |

表 54. Java 例外マッピング (続き)

| CICS 条件      | Java 例外                          |
|--------------|----------------------------------|
| DSIDERR      | FileNotFoundException            |
| DSSTAT       | DestinationStatusChangeException |
| DUPKEY       | DuplicateKeyException            |
| DUPREC       | DuplicateRecordException         |
| END          | EndException                     |
| ENDDATA      | EndOfDataException               |
| ENDFILE      | EndOfFileException               |
| ENDINPT      | EndOfInputIndicatorException     |
| ENQBUSY      | ResourceUnavailableException     |
| ENVDEFERR    | InvalidRetrieveOptionException   |
| EOC          | EndOfChainIndicatorException     |
| EODS         | EndOfDataSetIndicatorException   |
| EOF          | EndOfFileIndicatorException      |
| ERROR        | ErrorException                   |
| EXPIRED      | TimeExpiredException             |
| FILENOTFOUND | FileNotFoundException            |
| FUNCERR      | FunctionErrorException           |
| IGREQID      | InvalidREQIDPrefixException      |
| IGREQCD      | InvalidDirectionException        |
| ILLOGIC      | LogicException                   |
| INBFMH       | InboundFMHException              |
| INVERRTERM   | InvalidErrorTerminalException    |
| INVEXITREQ   | InvalidExitRequestException      |
| INVLDC       | InvalidLDCEXception              |
| INVMPSZ      | InvalidMapSizeException          |
| INVPARTNSET  | InvalidPartitionSetException     |
| INVPARTN     | InvalidPartitionException        |
| INVREQ       | InvalidRequestException          |
| INVTREQ      | InvalidTSRequestException        |
| IOERR        | IOException                      |
| ISCINVREQ    | ISCInvalidRequestException       |
| ITEMERR      | ItemErrorException               |
| JIDERR       | InvalidJournalIdException        |
| LENGERR      | LengthErrorException             |
| MAPERROR     | MapErrorException                |
| MAPFAIL      | MapFailureException              |
| NAMEERROR    | NameErrorException               |
| NODEIDERR    | InvalidNodeIdException           |
| NOJBUFSP     | NoJournalBufferSpaceException    |
| NONVAL       | NotValidException                |

表 54. Java 例外マッピング (続き)

| CICS 条件     | Java 例外                                      |
|-------------|----------------------------------------------|
| NOPASSBKRD  | NoPassbookReadException                      |
| NOPASSBKWR  | NoPassbookWriteException                     |
| NOSPACE     | NoSpaceException                             |
| NOSPOOL     | NoSpoolException                             |
| NOSTART     | StartFailedException                         |
| NOSTG       | NoStorageException                           |
| NOTALLOC    | NotAllocatedException                        |
| NOTAUTH     | NotAuthorisedException                       |
| NOTFINISHED | NotFinishedException                         |
| NOTFND      | RecordNotFoundException<br>NotFoundException |
| NOTOPEN     | NotOpenException                             |
| OPENERR     | DumpOpenErrorException                       |
| OVERFLOW    | MapPageOverflowException                     |
| PARTNFAIL   | PartitionFailureException                    |
| PGMIDERR    | InvalidProgramIdException                    |
| QBUSY       | QueueBusyException                           |
| QIDERR      | InvalidQueueIdException                      |
| QZERO       | QueueZeroException                           |
| RDATT       | ReadAttentionException                       |
| RETPAGE     | ReturnedPageException                        |
| ROLLEDBACK  | RolledBackException                          |
| RTEFAIL     | RouteFailedException                         |
| RTESOME     | RoutePartiallyFailedException                |
| SELNERR     | DestinationSelectionErrorException           |
| SESSBUSY    | SessionBusyException                         |
| SESSIONERR  | SessionErrorException                        |
| SIGNAL      | InboundSignalException                       |
| SPOLBUSY    | SpoolBusyException                           |
| SPOLERR     | SpoolErrorException                          |
| STRELERR    | STRELERRException                            |
| SUPPRESSED  | SuppressedException                          |
| SYMBOLERR   | SymbolErrorException                         |
| SYSBUSY     | SystemBusyException                          |
| SYSIDERR    | InvalidSystemIdException                     |
| TASKIDERR   | InvalidTaskIdException                       |
| TCIDERR     | TCIDERRException                             |
| TEMPLATERR  | TemplateErrorException                       |
| TERMERR     | TerminalException                            |
| TERMIDERR   | InvalidTerminalIdException                   |

表 54. Java 例外マッピング (続き)

| CICS 条件    | Java 例外                        |
|------------|--------------------------------|
| TOKENERR   | TokenErrorException            |
| TRANSIDERR | InvalidTransactionIdException  |
| TSIOERR    | TSIOErrorException             |
| UNEXPIN    | UnexpectedInformationException |
| USERIDERR  | InvalidUserIdException         |
| WRBRK      | WriteBreakException            |
| WRONGSTAT  | WrongStatusException           |

注: CICS コマンド WEB RECEIVE が、受信されたデータが非 HTTP メッセージ (TYPE=HTTPNO の設定による) であることを示す場合、NonHttpDataException が getContent() によってスローされます。

## エラー処理と異常終了

Java プログラムから ABEND を開始するには、Task.abend() メソッドまたは Task.forceAbend() メソッドのいずれかを呼び出す必要があります。

| メソッド                  | JCICS クラス | EXEC CICS コマンド |
|-----------------------|-----------|----------------|
| abend(), forceAbend() | タスク       | ABEND          |

### ABEND

Java プログラムから ABEND を開始するには、Task.abend() メソッドのいずれかを呼び出します。これにより、アベンド条件が CICS で設定され、AbendException がスローされます。AbendException が上位レベルのアプリケーション・オブジェクト内でキャッチされないか、呼び出し側プログラムに登録されている ABEND ハンドラー (ある場合) によって処理される場合、CICS はトランザクションを終了し、ロールバックします。

各種 abend() メソッドは次のとおりです。

- abend (String *abcode*)。ABEND に ABEND コード *abcode* が設定されます。
- abend (String *abcode*, boolean *dump*)。ABEND に ABEND コード *abcode* が設定されます。**dump** パラメーターが false である場合、ダンプは取られません。
- abend()。ABEND コードもダンプもない ABEND が生じます。

### ABEND CANCEL

処理できない ABEND を開始するには、Task.forceAbend() メソッドのいずれかを呼び出します。上記のように、これにより、AbendCancelException がスローされ、Java プログラムでキャッチされます。これを行う場合は、**ABEND\_CANCEL** 処理を完了するために例外を再スローする必要があります。その結果、制御が CICS に戻ると、CICS はトランザクションを終了し、ロールバックします。通知目的の AbendCancelException のキャッチのみを行ってから、再スローしてください。

各種 forceAbend() メソッドは次のとおりです。

- `forceAbend (String abcode)`。**ABEND CANCEL** に ABEND コード *abcode* が設定されます。
- `forceAbend (String abcode, boolean dump)`。**ABEND CANCEL** に ABEND コード *abcode* が設定されます。**dump** パラメーターが `false` である場合、ダンプは取られません。
- `forceAbend()`。ABEND コードもダンプもない **ABEND CANCEL** が生じます。

## APPC マップ式会話

APPC 非マップ式会話は、JCICS API からサポートされません。

APPC マップ式会話:

| メソッド                             | JCICS クラス                    | EXEC CICS コマンド              |
|----------------------------------|------------------------------|-----------------------------|
| <code>initiate()</code>          | <code>AttachInitiator</code> | ALLOCATE、CONNECT<br>PROCESS |
| <code>converse()</code>          | <code>Conversation</code>    | CONVERSE                    |
| <code>get*()</code> メソッド         | <code>Conversation</code>    | EXTRACT ATTRIBUTES          |
| <code>get*()</code> メソッド         | <code>Conversation</code>    | EXTRACT PROCESS             |
| <code>free()</code>              | <code>Conversation</code>    | FREE                        |
| <code>issueAbend()</code>        | <code>Conversation</code>    | ISSUE ABEND                 |
| <code>issueConfirmation()</code> | <code>Conversation</code>    | ISSUE CONFIRMATION          |
| <code>issueError()</code>        | <code>Conversation</code>    | ISSUE ERROR                 |
| <code>issuePrepare()</code>      | <code>Conversation</code>    | ISSUE PREPARE               |
| <code>issueSignal()</code>       | <code>Conversation</code>    | ISSUE SIGNAL                |
| <code>receive()</code>           | <code>Conversation</code>    | RECEIVE                     |
| <code>send()</code>              | <code>Conversation</code>    | SEND                        |
| <code>flush()</code>             | <code>Conversation</code>    | WAIT CONVID                 |

## 基本マッピング・サポート (BMS)

基本マッピング・サポート (BMS) は、CICS プログラムと端末装置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。JCICS は、BMS アプリケーション・プログラミング・インターフェースの一部をサポートします。

| メソッド                       | JCICS クラス                              | EXEC CICS コマンド       |
|----------------------------|----------------------------------------|----------------------|
| <code>sendControl()</code> | <code>TerminalPrincipalFacility</code> | SEND CONTROL         |
| <code>sendText()</code>    | <code>TerminalPrincipalFacility</code> | SEND TEXT            |
|                            | サポートされていない                             | SEND MAP、RECEIVE MAP |

## チャネルとコンテナの例

コンテナ 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。コンテナは、チャネル と呼ばれる集合にグループ化されます。この資料では、Java アプリケーションでチャネルとコンテナを使用する方法について説明し、いくつかのサンプル・コードを示します。

チャンネルとコンテナの概要、および非 Java アプリケーションでのチャンネル使用の手引きについては、チャンネルによるプログラム間データ転送を参照してください。Java プログラムが既存の CICS アプリケーション・データにアクセスできるようにするツールについては、821 ページの『Java からの構造化データとの対話』を参照してください。

表 55 では、チャンネルとコンテナに対する JCICS サポートを実装するクラスとメソッドをリストしています。

表 55. チャンネルとコンテナに対する JCICS サポート

| メソッド                | JCICS クラス                 | EXEC CICS コマンド                |
|---------------------|---------------------------|-------------------------------|
| containerIterator() | Channel                   | STARTBROWSE CONTAINER         |
| createContainer()   | Channel                   |                               |
| delete()            | Channel                   | DELETE CHANNEL                |
| deleteContainer()   | Channel                   | DELETE CONTAINER CHANNEL      |
| getContainer()      | Channel                   |                               |
| getContainerCount() | Channel                   | QUERY CHANNEL                 |
| getName()           | Channel                   |                               |
| delete()            | Container (コンテナ)          | DELETE CONTAINER CHANNEL      |
| get()               | Container (コンテナ)          | GET CONTAINER CHANNEL         |
| getLength()         | Container (コンテナ)          | GET CONTAINER CHANNEL NODATA  |
| getDatatype()       | Container (コンテナ)          |                               |
| getName()           | Container (コンテナ)          |                               |
| put()               | Container (コンテナ)          | PUT CONTAINER CHANNEL         |
| getOwner()          | ContainerIterator         |                               |
| hasNext()           | ContainerIterator         |                               |
| next()              | ContainerIterator         | GETNEXT CONTAINER BROWSETOKEN |
| remove()            | ContainerIterator         |                               |
| link()              | Program                   | LINK                          |
| setNextChannel()    | TerminalPrincipalFacility | RETURN CHANNEL                |
| issue()             | StartRequest              | START CHANNEL                 |
| createChannel()     | タスク                       |                               |
| getCurrentChannel() | タスク                       | ASSIGN CHANNEL                |
| containerIterator() | タスク                       | STARTBROWSE CONTAINER         |

CICS 条件 CHANNELERR の場合、結果として ChannelErrorException がスローされます。CONTAINERERR CICS 条件の結果は ContainerErrorException になります。CCSIDERR CICS 条件の結果は CCSIDErrorException になります。

JCICS におけるチャンネルとコンテナの作成:

チャンネルを作成するには、Task クラスの `createChannel()` メソッドを使用します。

以下に例を示します。

```
Task t=Task.getTask();
Channel custData = t.createChannel("Customer_Data");
```

`createChannel` メソッドに提供されるストリングは、Channel オブジェクトが CICS に認識されている名前です (この名前は、CICS 命名規則に準拠するために、16 文字までスペースで埋め込まれます)。

チャンネルに新しいコンテナを作成するには、Channel `createContainer()` メソッドを使用します。以下に例を示します。

```
Container custRec = custData.createContainer("Customer_Record");
```

`createContainer()` メソッドに提供されるストリングは、Container オブジェクトが CICS に認識されている名前です この名前は、CICS 命名規則に準拠するために、必要に応じて 16 文字までスペースで埋め込まれます。同じ名前のコンテナがこのチャンネルに既に存在する場合、`ContainerErrorException` がスローされます。

コンテナへのデータの書き込み:

Container オブジェクトにデータを書き込むには、Container `put()` メソッドを使用します。

データは、ストリングとしてコンテナに追加できます。以下に例を示します。

```
String custNo = "00054321";
string[] custRecIn = custNo.putString();
custRec.put(custRecIn);
```

または

```
custRec.putString("00054321");
```

別のプログラムまたはタスクへのチャンネルの受け渡し:

プログラム・リンクでチャンネルを渡すには、Program クラスの `link()` メソッドを使用します。

```
programX.link(custData);
```

プログラム戻り呼び出しで次のチャンネルを設定するには、TerminalPrincipalFacility クラスの `setNextChannel()` メソッドを使用します。

```
terminalPF.setNextChannel(custData);
```

START 要求でチャンネルを渡すには、StartRequest クラスの `issue` メソッドを使用します。

```
startrequest.issue(custData);
```

現行チャンネルの受け取り:

プログラムが現行チャンネルを明示的に受け取る必要はありません。ただし、プログラムは現行チャンネルを現行タスクから取得することができます。

プログラムが現行タスクから現行チャンネルを取得する場合、タスクはコンテナを名前で取り出すことができます。

```
Task t = Task.getTask();
Channel custData = t.getCurrentChannel();

if (custData != null) {
 Container custRec = custData.getContainer("Customer_Record");
} else {
 System.out.println("There is no Current Channel");
}
```

コンテナからのデータの取得:

コンテナ内のデータをバイト配列に読み取るには、`Container.get()` メソッドを使用します。

```
byte[] custInfo = custRec.get();
```

現行チャンネルのブラウズ:

チャンネルが渡される JCICS プログラムは、そのチャンネルを明示的に受け取ることなく、すべての `Container` オブジェクトにアクセスできます。

これを行うには、`ContainerIterator` オブジェクトを使用します

`ContainerIterator` クラスは `java.util.Iterator` インターフェースを実装します。

`Task` オブジェクトが現行タスクからインスタンス化される場合、その `containerIterator()` メソッドは、現行チャンネルの `Iterator` を返し、現行チャンネルがない場合は `null` を返します。以下に例を示します。

```
Task t = Task.getTask();
ContainerIterator ci = t.containerIterator();

while (ci.hasNext()) {
 Container custData = ci.next();
 // Process the container...
}
```

チャンネルとコンテナの例:

この例は、`PAYR` という名前の COBOL サーバー・プログラムを呼び出す、`Payroll` と呼ばれる Java クラスの抜粋を示しています。`Payroll` クラスはチャンネルとそのコンテナを処理するために、JCICS の `com.ibm.cics.server.Channel` クラスと `com.ibm.cics.server.Container` クラスを使用します。

```

import com.ibm.cics.server.*;

public class Payroll
{
 ...
 Task t=Task.getTask();

 // create the payroll_2004 channel
 Channel payroll_2004 = t.createChannel("payroll-2004");

 // create the employee container
 Container employee = payroll_2004.createContainer("employee");

 // put the employee name into the container
 employee.putString("John Doe");

 // create the wage container
 Container wage = payroll_2004.createContainer("wage");

 // put the wage into the container
 wage.putString("2000");

 // Link to the PAYROLL program, passing the payroll_2004 channel
 Program p = new Program();
 p.setName("PAYR");
 p.link(payroll_2004);

 // Get the status container which has been returned
 Container status = payroll_2004.getContainer("status");

 if (status != null)
 {
 // Get the status information
 byte[] payrollStatus = status.get();
 }

 ...
}

```

図 131. JCICS の *com.ibm.cics.server.Channel* および *com.ibm.cics.server.Container* クラスを使用して、COBOL サーバー・プログラムにチャンネルを渡す Java クラス

## 診断サービス

JCICS アプリケーション・プログラミング・インターフェースは、以下の CICS トレースおよびダンプ・コマンドをサポートします。

| メソッド         | JCICS クラス    | EXEC CICS コマンド |
|--------------|--------------|----------------|
|              | サポートされていない   | DUMP           |
| enterTrace() | EnterRequest | ENTER          |

## 文書サービス

このセクションでは、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コマンドの JCICS サポートについて説明します。

Document クラスは **EXEC CICS DOCUMENT** API にマップします。

Document クラスのデフォルトの引数なしのコンストラクターが、CICS 内に新しい文書を作成します。コンストラクター Document(byte[] docToken) は、以前に作成された既存の文書の文書トークンを受け入れます。例えば、別のプログラムが文書を作成して、その文書トークンを COMMAREA またはコンテナ内の Java アプリケーションに渡すことができます。

DocumentLocation クラスのコンストラクターは、EXEC CICS DOCUMENT API の AT および TO キーワードにマップされます。

SymbolList クラスの setter と getter は、EXEC CICS DOCUMENT API の SYMBOLLIST、LENGTH、DELIMITER、および UNESCAPE キーワードにマップされます。

| メソッド            | JCICS クラス | EXEC CICS コマンド    |
|-----------------|-----------|-------------------|
| create*()       | Document  | DOCUMENT CREATE   |
| append*()       | Document  | DOCUMENT INSERT   |
| insert*()       | Document  | DOCUMENT INSERT   |
| addSymbol()     | Document  | DOCUMENT SET      |
| setSymbolList() | Document  | DOCUMENT SET      |
| retrieve*()     | Document  | DOCUMENT RETRIEVE |
| get*()          | Document  | DOCUMENT          |

## 環境サービス

CICS 環境サービスは、アプリケーション・プログラムに関連する CICS データ域、パラメーター、およびリソース属性にアクセスできるようにします。

JCICS サポートに相当する EXEC CICS コマンドとオプションは次のとおりです。

- ADDRESS
- ASSIGN
- INQUIRE SYSTEM
- INQUIRE TASK
- INQUIRE TERMINAL/NETNAME

### ADDRESS:

ADDRESS API コマンド・オプションには次のサポートが提供されます。

EXEC CICS ADDRESS コマンドの詳細については、ADDRESSを参照してください。

ACEE アクセス制御環境エレメント (ACEE) は、CICS ユーザーがサインオンするときに外部セキュリティ・マネージャーによって作成されます。このオプションは JCICS ではサポートされません。

### COMMAREA

COMMAREA には、コマンドで渡されるユーザー・データが入っています。COMMAREA ポインターは、CommAreaHolder 引数によって、リンクされたプログラムに自動的に渡されます。詳しくは、716 ページの『データを渡すための引数』を参照してください。

**CWA** 共通作業域 (CWA) には、タスク間で共用可能なグローバル・ユーザー・データが入っています。CWA のコピーは、Region クラスの `getCWA()` メソッドを使用して取得できます。

**EIB** EIB フィールド には、最後に実行された CICS コマンドに関する情報が入っています。EIB 値へのアクセスは、該当するオブジェクトのメソッドによって提供されます。以下に例を示します。

**eibtrnid**

Task クラスの `getTransactionName()` メソッドによって戻されます。

**eibaid**

TerminalPrincipalFacility クラスの `getAIDbyte()` メソッドによって戻されます。

**eibcposn**

Cursor クラスの `getRow()` および `getColumn()` メソッドによって戻されます。

**TCTUA**

端末管理テーブル・ユーザー域 (TCTUA) には、CICS トランザクションを起動する端末 (基本機構) に関連したユーザー・データが入っています。この領域は、アプリケーション・プログラム間で情報を渡すのに使用されますが、関係するアプリケーション・プログラムに同じ端末が関連付けられている場合のみです。TCTUA の内容は、TerminalPrincipalFacility クラスの `getTCTUA()` メソッドを使用して取得できます。

**TWA** トランザクション作業域 (TWA) には、CICS タスクに関連したユーザー・データが入っています。この領域は、アプリケーション・プログラム間で情報を渡すのに使用されますが、同じタスク内にある場合のみです。TWA のコピーは、Task クラスの `getTWA()` メソッドを使用して取得できます。

**ASSIGN:**

**ASSIGN** API コマンド・オプションには次のサポートが提供されます。

このコマンドについて詳しくは、ASSIGNを参照してください。

| メソッド                                 | JCICS クラス                                                      |
|--------------------------------------|----------------------------------------------------------------|
| <code>getABCODE()</code>             | AbendException                                                 |
| <code>getApplicationContext()</code> | タスク                                                            |
| <code>getAPPLID()</code>             | 領域                                                             |
| <code>getCurrentChannel()</code>     | タスク                                                            |
| <code>getCWA()</code>                | 領域                                                             |
| <code>getName()</code>               | TerminalPrincipalFacility または<br>ConversationPrincipalFacility |
| <code>getFCI()</code>                | タスク                                                            |
| <code>getNetName()</code>            | TerminalPrincipalFacility または<br>ConversationPrincipalFacility |
| <code>getPrinSysid()</code>          | TerminalPrincipalFacility または<br>ConversationPrincipalFacility |

| メソッド                         | JCICS クラス                                                        |
|------------------------------|------------------------------------------------------------------|
| getProgramName()             | タスク                                                              |
| getQNAME()                   | タスク                                                              |
| getSTARTCODE()               | タスク                                                              |
| getSysid()                   | 領域                                                               |
| getTCTUA()                   | TerminalPrincipalFacility                                        |
| getTERMCODE()                | TerminalPrincipalFacility                                        |
| getTWA()                     | タスク                                                              |
| getUserID()、Task.getUserID() | Task、TerminalPrincipalFacility または ConversationPrincipalFacility |

その他の ASSIGN オプションはサポートされません。

#### INQUIRE SYSTEM:

**INQUIRE SYSTEM** SPI オプションに対するサポートが提供されます。

| メソッド        | JCICS クラス |
|-------------|-----------|
| getAPPLID() | 領域        |
| getSYSID()  | 領域        |

その他の **INQUIRE SYSTEM** オプションはサポートされません。

#### INQUIRE TASK:

**INQUIRE TASK** API コマンド・オプションには次のサポートが提供されます。

| メソッド                 | JCICS クラス |
|----------------------|-----------|
| getSTARTCODE()       | タスク       |
| getTransactionName() | タスク       |
| getUserID()          | タスク       |

#### FACILITY

タスクの基本機構で getName() メソッドを呼び出すことによって、タスクの基本機構の名前を見つけることができます。基本機構は、現行の Task オブジェクトで getPrincipalFacility() メソッドを呼び出すことによって見つけることができます。

#### FACILITYTYPE

Java instanceof 演算子を使用して、戻されたオブジェクト参照のクラスを確認することによって、機構のタイプを判別できます。

その他の **INQUIRE TASK** オプションはサポートされません。

#### INQUIRE TERMINAL および INQUIRE NETNAME:

**INQUIRE TERMINAL** および **INQUIRE NETNAME** SPI オプションには次のサポートが提供されます。

|                    |                                          |
|--------------------|------------------------------------------|
| メソッド               | JCICS クラス                                |
| getUserID()        | Terminal、ConversationalPrincipalFacility |
| Terminal.getUser() | Terminal、ConversationalPrincipalFacility |

現行の Task オブジェクトで、またはタスクの基本機構を表すオブジェクトで getUserID() メソッドを呼び出すことでも、USERID 値を見つけることができます。

他の **INQUIRE TERMINAL** または **INQUIRE NETNAME** オプションはサポートされません。

## ファイル・サービス

JCICS は、CICS ファイルおよび索引のタイプごとに **EXEC CICS** API コマンドにマップされるクラスおよびメソッドを提供します。

Java プログラムが既存の CICS アプリケーション・データにアクセスできるようにするツールについては、Java からの構造化データとの対話を参照してください。

CICS は、次のタイプのファイルをサポートします。

- 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)

KSDS および ESDS ファイルには、代替 (または 2 次) 索引を備えることができます。CICS は、2 次索引から RRDS ファイルへのアクセスをサポートしません。2 次索引は、それ自体が別々の KSDS ファイルである、つまり別々の FD 項目を持つ場合と同じように CICS によって扱われます。

KSDS、ESDS (1 次索引)、および ESDS (2 次索引) ファイルのアクセスにはいくつかの相違点があります。すなわち、常に共通インターフェースを使用できるとは限りません。

どのタイプのファイルでもレコードの読み取り、更新、削除、およびブラウズを行うことができますが、例外的に ESDS ファイルからはレコードを削除することはできません。

データ・セットについて詳しくは、VSAM data sets: KSDS, ESDS, RRDS を参照してください。

データを読み取る Java コマンドは、**EXEC CICS** コマンドの SET オプションに相当するもののみをサポートします。戻されるデータは、CICS ストレージから Java オブジェクトに自動的にコピーされます。

ファイル制御に関連する Java インターフェースには、5 つのカテゴリがあります。

**File**  他のファイル・クラスのスーパークラス。すべてのファイル・クラスに共通のメソッドが含まれます。

## KeyedFile

1 次索引を使用してアクセスされる KSDS ファイル、2 次索引を使用してアクセスされる KSDS ファイル、および 2 次索引を使用してアクセスされる ESDS ファイルに共通のインターフェースが含まれます。

## KSDS

KSDS ファイルに固有のインターフェースが含まれます。

**ESDS** 相対バイト・アドレス (RBA、1 次索引) または拡張相対バイト・アドレス (XRBA) からアクセスされる ESDS ファイルに固有のインターフェースが含まれます。RBA ではなく、XRBA を使用するには、`setXRBA(true)` メソッドを発行します。

## RRDS

相対レコード番号 (RRN、1 次索引) からアクセスされる RRDS ファイルに固有のインターフェースが含まれます。

ファイルごとに、作動可能な 2 つのオブジェクト、つまり、File オブジェクトと FileBrowse オブジェクトがあります。File オブジェクトはファイル自体を表し、次の API オペレーションを実行するメソッドで使用できます。

- DELETE
- READ
- REWRITE
- UNLOCK
- WRITE
- STARTBR

File オブジェクトは、必要なファイル・クラスを明示的に開始するユーザー・アプリケーションによって作成されます。FileBrowse オブジェクトは、ファイル上のブラウズ・オペレーションを表します。特定のファイルに対して常に複数のアクティブ・ブラウズが可能であり、各ブラウズは REQID で区別されます。メソッドは、次の API オペレーションを実行するために FileBrowse オブジェクトに対してインスタンス化することができます。

- ENDBR
- READNEXT
- READPREV
- RESETBR

FileBrowse オブジェクトは、ユーザー・アプリケーションによって明示的にインスタンス化されません。STARTBR オペレーションを実行するメソッドによって作成され、ユーザー・クラスに戻されます。

以下の表では、JCICS クラスとメソッドが、CICS ファイルおよび索引のタイプごとに **EXEC CICS** API コマンドにどのようにマップされるかを示しています。これらの表では、JCICS クラスとメソッドは `class.method()` の形式で示されます。例えば、`KeyedFile.read()` は、`KeyedFile` クラスの `read()` メソッドを参照しています。

最初の表では、キー付きファイルのクラスとメソッドを示しています。

表 56. キー付きファイルのクラスとメソッド

| <b>KSDS 1 次または 2 次索引の<br/>クラスとメソッド</b> | <b>ESDS 2 次索引の<br/>クラスとメソッド</b> | <b>CICS File API コマン<br/>ド</b> |
|----------------------------------------|---------------------------------|--------------------------------|
| KeyedFile.read()                       | KeyedFile.read()                | <b>READ</b>                    |
| KeyedFile.readForUpdate()              | KeyedFile.readForUpdate()       | <b>READ UPDATE</b>             |
| KeyedFile.readGeneric()                | KeyedFile.readGeneric()         | <b>READ GENERIC</b>            |
| KeyedFile.rewrite()                    | KeyedFile.rewrite()             | <b>REWRITE</b>                 |
| KSDS.write()                           | KSDS.write()                    | <b>WRITE</b>                   |
| KSDS.delete()                          |                                 | <b>DELETE</b>                  |
| KSDS.deleteGeneric()                   |                                 | <b>DELETE GENERIC</b>          |
| KeyedFile.unlock()                     | KeyedFile.unlock()              | <b>UNLOCK</b>                  |
| KeyedFile.startBrowse()                | KeyedFile.startBrowse()         | <b>START BROWSE</b>            |
| KeyedFile.startGenericBrowse()         | KeyedFile.startGenericBrowse()  | <b>START BROWSE GENERIC</b>    |
| KeyedFileBrowse.next()                 | KeyedFileBrowse.next()          | <b>READNEXT</b>                |
| KeyedFileBrowse.previous()             | KeyedFileBrowse.previous()      | <b>READPREV</b>                |
| KeyedFileBrowse.reset()                | KeyedFileBrowse.reset()         | <b>RESET BROWSE</b>            |
| FileBrowse.end()                       | FileBrowse.end()                | <b>END BROWSE</b>              |

次の表は、キー付きでないファイルのクラスとメソッドを示しています。ESDS と RRDS は 1 次索引によってアクセスされます。

| <b>ESDS 1 次索引の<br/>クラスとメソッド</b> | <b>RRDS 1 次索引の<br/>クラスとメソッド</b> | <b>CICS File API コマン<br/>ド</b> |
|---------------------------------|---------------------------------|--------------------------------|
| ESDS.read()                     | RRDS.read()                     | <b>READ</b>                    |
| ESDS.readForUpdate()            | RRDS.readForUpdate()            | <b>READ UPDATE</b>             |
| ESDS.rewrite()                  | RRDS.rewrite()                  | <b>REWRITE</b>                 |
| ESDS.write()                    | RRDS.write()                    | <b>WRITE</b>                   |
|                                 | RRDS.delete()                   | <b>DELETE</b>                  |
| KeyedFile.unlock()              | RRDS.unlock()                   | <b>UNLOCK</b>                  |
| ESDS.startBrowse()              | RRDS.startBrowse()              | <b>START BROWSE</b>            |
| ESDS_Browse.next()              | RRDS_Browse.next()              | <b>READNEXT</b>                |
| ESDS_Browse.previous()          | RRDS_Browse.previous()          | <b>READPREV</b>                |
| ESDS_Browse.reset()             | RRDS_Browse.reset()             | <b>RESET BROWSE</b>            |
| FileBrowse.end()                | FileBrowse.end()                | <b>END BROWSE</b>              |
| ESDS.setXRBA()                  |                                 |                                |

ファイルに書き込まれるデータは、Java バイト配列でなければなりません。

データはファイルから RecordHolder オブジェクトに読み取られます。ストレージは CICS によって提供され、プログラムの終わりに自動的に解放されます。

File メソッドには **KEYLENGTH** 値を指定する必要はありません。使用される長さは、渡されるキーの実際の長さです。FileBrowse オブジェクトが作成されると、

startBrowse メソッドで指定されたキーの長さが入っています。この長さは、そのオブジェクトに対する以降のブラウズ要求で CICS に渡されます。

ブラウズ・オペレーションに **REQID** を指定する必要はありません。各ブラウズ・オブジェクトには、固有の REQID が含まれ、そのブラウズ・オブジェクトに対する以降のすべてのブラウズ要求に自動的に使用されます。

## HTTP サービスおよび TCP/IP サービス

HttpHeader、NameValueData、および FormField クラスの getter は、HTTP ヘッダー、名前と値のペア、および該当する API コマンドのフォーム・フィールド値を戻します。

| メソッド                   | JCICS クラス       | EXEC CICS コマンド                         |
|------------------------|-----------------|----------------------------------------|
| get*()                 | CertificateInfo | EXTRACT CERTIFICATE /<br>EXTRACT TCPIP |
| get*()                 | HttpRequest     | EXTRACT WEB                            |
| getHeader()            | HttpRequest     | WEB READ HTTPHEADER                    |
| getFormField()         | HttpRequest     | WEB READ FORMFIELD                     |
| getContent()           | HttpRequest     | WEB RECEIVE                            |
| getQueryParm()         | HttpRequest     | WEB READ QUERYPARM                     |
| startBrowseHeader()    | HttpRequest     | WEB STARTBROWSE HTTPHEADER             |
| getNextHeader()        | HttpRequest     | WEB READNEXT HTTPHEADER                |
| endBrowseHeader()      | HttpRequest     | WEB ENDBROWSE HTTPHEADER               |
| startBrowseFormField() | HttpRequest     | WEB STARTBROWSE FORMFIELD              |
| getNextFormField()     | HttpRequest     | WEB READNEXT FORMFIELD                 |
| endBrowseFormField()   | HttpRequest     | WEB ENDBROWSE FORMFIELD                |
| startBrowseQueryParm() | HttpRequest     | WEB STARTBROWSE QUERYPARM              |
| getNextQueryParm()     | HttpRequest     | WEB READNEXT QUERYPARM                 |
| endBrowseQueryParm()   | HttpRequest     | WEB ENDBROWSE QUERYPARM                |
| writeHeader()          | HttpResponse    | WEB WRITE                              |
| getDocument()          | HttpResponse    | WEB RETRIEVE                           |
| getCurrentDocument()   | HttpResponse    | WEB RETRIEVE                           |
| sendDocument()         | HttpResponse    | WEB SEND                               |

注: HttpRequest オブジェクトを取得するには、メソッド getHttpRequestInstance() を使用してください。

CICS Web サポートによって処理される各着信 HTTP 要求には、HTTP ヘッダーが含まれています。要求で POST HTTP verb を使用する場合、文書データも含まれます。CICS Web サポートによって生成される各応答 HTTP 要求には、HTTP ヘッダーと文書データが含まれています。

これを処理するために、JCICS は次の Web および TCP/IP サービスを提供します。

### HTTP ヘッダー

HttpRequest クラスを使用して HTTP ヘッダーを調べることができます。

GET モードの HTTP では、クライアントが HTTP フォームに入力し、送信ボタンを選択した場合、照会ストリングが送信されます。

**SSL** CICS Web サポートは、TcpipRequest クラスを提供します。これは、要求を送信したクライアントに関する詳細情報と、SSL サポートに関する基本情報を取得するために、HttpRequest によって拡張されます。SSL 証明書が提供される場合、CertificateInfo クラスを使用して詳細に調べることができます。

**文書** 文書がサーバーに公開される (HTTP POST) 場合、CICS 文書として提供されます。HttpRequest クラスの getDocument() メソッドを呼び出すことによって、この文書にアクセスできます。既存文書の処理の詳細については、731 ページの『文書サービス』を参照してください。

要求から生じる HTTP クライアント Web コンテンツを提供するために、サーバー・プログラマーは、Document Services API を使用して CICS 文書を作成し、sendDocument() メソッドを呼び出す必要があります。

CICS Web サポートについて詳しくは、CICS Web サポートを参照してください。JCICS Web クラスについて詳しくは、JCICS Javadoc 情報を参照してください。

## プログラム・サービス

JCICS は、CICS プログラム制御コマンド LINK、RETURN、および INVOKE APPLICATION をサポートします。

Java プログラムが既存の CICS アプリケーション・データにアクセスできるようにするツールについては、Java からの構造化データとの対話を参照してください。

表 57 には、CICS プログラム制御コマンドにマップされるメソッドと JCICS クラスがリストされています。

表 57. メソッド、JCICS クラス、および CICS コマンド間の関係：

| EXEC CICS コマンド     | JCICS クラス                 | JCICS メソッド                                                        |
|--------------------|---------------------------|-------------------------------------------------------------------|
| LINK               | Program                   | link()                                                            |
| RETURN             | TerminalPrincipalFacility | setNextTransaction()<br>、 setNextCOMMAREA()<br>、 setNextChannel() |
| INVOKE APPLICATION | Application               | invoke()                                                          |

### LINK

link() メソッドを使用して、CICS に対して定義される別のプログラムに制御を移動することができます。ターゲット・プログラムは、CICS でサポートされる任意の言語にすることができます。

### RETURN

このコマンドの疑似会話型の側面のみがサポートされます。return に対する CICS 呼び出しを行う必要はありません。アプリケーションは通常どおり終了できます。疑似会話型機能は、TerminalPrincipalFacility クラスのメソッドでサポートされます。setNextTransaction() は、RETURN の TRANSID オプションの使用に相当します。setNextCOMMAREA() は、COMMAREA オプションの使用に相当します。一方、setNextChannel() は、CHANNEL オプションの使用に

相当します。これらのメソッドは、プログラムの実行中にいつでも呼び出すことができ、プログラムが終了するときに有効になります。

## INVOKE

いずれか 1 つのプログラム・エン트리・ポイントに対応する操作を指定することで、アプリケーションを呼び出すことができます。アプリケーション・エン트리・ポイント・プログラムの名前を知っておく必要はなく、プログラムがパブリック/プライベートのどちらであるかも無関係です。

注: 指定される COMMAREA の長さは、CICS の LENGTH 値として使用されます。COMMAREA が任意の 2 つの CICS サーバー (製品/バージョン/リリースの任意の組み合わせ) 間で渡される場合、この値は 24 KB を超えてはなりません。この制限により、ヘッダーに COMMAREA およびスペースを使用できます。

## スケジューリング・サービス

JCICS は、CICS スケジューリング・サービスをサポートします。これにより、タスク用に保管されたデータを取り出し、インターバル制御要求を取り消し、指定された時間にタスクを開始することができます。

| メソッド       | JCICS クラス    | EXEC CICS コマンド |
|------------|--------------|----------------|
| cancel()   | StartRequest | CANCEL         |
| retrieve() | タスク          | RETRIEVE       |
| issue()    | StartRequest | START          |

Task.retrieve() メソッドによって取り出される内容を定義するには、java.util.BitSet オブジェクトを使用します。com.ibm.cics.server.RetrieveBits クラスは、BitSet オブジェクトで設定できる次のビットを定義します。

- RetrieveBits.DATA
- RetrieveBits.RTRANSID
- RetrieveBits.RTERMID
- RetrieveBits.QUEUE

これらは、EXEC CICS RETRIEVE コマンドのオプションに対応します。

Task.retrieve() メソッドは、RetrieveBits の設定に応じて、単一の呼び出しで最大 4 つの情報を取り出します。DATA、RTRANSID、RTERMID および QUEUE データは、RetrievedData オブジェクトに置かれ、このオブジェクトは RetrievedDataHolder オブジェクトに保持されます。次の例では、データと transid を取り出します。

```
BitSet bs = new BitSet();
bs.set(RetrieveBits.DATA, true);
bs.set(RetrieveBits.RTRANSID, true);
RetrievedDataHolder rdh = new RetrievedDataHolder();
t.retrieve(bs, rdh);
byte[] inData = rdh.value.data;
String transid = rdh.value.transId;
```

## 直列化サービス

JCICS は、タスクによるリソースの使用をスケジュールできる CICS 直列化サービスをサポートします。

| メソッド                   | JCICS クラス               | EXEC CICS コマンド |
|------------------------|-------------------------|----------------|
| dequeue()              | SynchronizationResource | DEQ            |
| enqueue()、tryEnqueue() | SynchronizationResource | ENQ            |

## ストレージ・サービス

CICS サービスを使用した明示的なストレージ管理 (**EXEC CICS GETMAIN** など) はサポートされません。標準の Java ストレージ管理機能で、タスク専用ストレージのニーズを十分に満たすことができます。

タスク間のデータの共用は、CICS リソースを使用して行われなければなりません。

名前は一般的に、Java ストリングまたはバイト配列として表されます。これらが必要な長さであることを確認する必要があります。

## スレッドとタスクのサンプル

CICS Java アプリケーションを OSGi または Liberty 環境で実行する場合は、CICSExecutorService を使用して、CICS タスク/トランザクションごとに別のスレッドで処理を実行することができます。

Java 実行可能オブジェクトまたは呼び出し可能オブジェクトを Executor サービスに実行依頼すると、実行依頼されたアプリケーション・コードは、新規の CICS タスクの別のスレッドで実行されます。Java から作成された通常のスレッドとは異なり、Executor で制御されるスレッドは、JCICS API と CICS サービスにアクセスできます。CICS の OSGi または Liberty 環境では、標準の OSGi API を使用して CICSExecutorService を見つけることも、JCICS API の便利メソッド CICSExecutorService.runAsCICS() を使用することもできます。この便利メソッドは、サービスを見つけると、ユーザーの代わりに実行可能オブジェクトまたは呼び出し可能オブジェクトを実行依頼します。

注: Liberty での非 HTTP 要求に CICS タスクが作成されるのは、タイプ 2 の接続で JCICS または JDBC データ・ソースが最初に呼び出された場合のみです。

以下の例は、アプリケーション・コードの実行可能な部分を CICSExecutorService に実行依頼する Java クラスを抜粋したものです。このアプリケーション・コードは、単純に CICS TSQ への書き込みを行います。

```
public class ExecutorTest
{
 public static void main(String[] args)
 {
 // Inline the new Runnable class
 class CICSJob implements CICSTransactionRunnable
 {
 public void run()
 {
 // Create a temporary storage queue
 TSQ test_tsq = new TSQ();
 test_tsq.setType(TSQType.MAIN);

 // Set the TSQ name
 test_tsq.setName("TSQWRITE");

 // Write to the temporary storage queue
 // Use the CICS region local CCSID so it is readable
 String test_string = "Hello from a non CICS Thread - " + threadId;

 try
```

```

 {
 test_tsq.writeItem(test_string.getBytes(System.getProperty("com.ibm.cics.jvmserver.local.ccsid")));
 }
 catch (Exception e)
 {
 e.printStackTrace();
 }
 }

 @Override
 public String getTranid()
 {
 // *** This transaction id should be installed and available ***
 return "IJSJ";
 }
}

// Create and run the new CICSJob Runnable
Runnable task = new CICSJob();
CICSExecutorService.runAsCICS(task);
}
}

```

## 一時記憶域キュー・サービス

JCICS は、CICS 一時記憶コマンド DELETEQ TS、READQ TS、および WRITEQ TS をサポートします。

### JCICS メソッドと EXEC CICS コマンド間の対話

Java プログラムが既存の CICS アプリケーション・データにアクセスできるようにするツールについては、Java からの構造化データとの対話を参照してください。

表 58 は、CICS 一時記憶コマンドにマップされるメソッドと JCICS クラスをリストしています。

表 58. メソッド、JCICS クラスおよび CICS コマンド間の関係

| メソッド                                                                                  | JCICS クラス | EXEC CICS コマンド |
|---------------------------------------------------------------------------------------|-----------|----------------|
| delete()                                                                              | TSQ       | DELETEQ TS     |
| readItem()、 readNextItem()                                                            | TSQ       | READQ TS       |
| writeItem()、<br>rewriteItem()、<br>writeItemConditional()、<br>rewriteItemConditional() | TSQ       | WRITEQ TS      |

#### DELETEQ TS

TSQ クラスの delete() メソッドを使用して一時記憶域キュー (TSQ) を削除することができます。

#### READQ TS

CICS INTO オプションは Java プログラムではサポートされません。TSQ クラスの readItem() および readNextItem() メソッドを使用して、TSQ から特定の項目を読み取ることができます。これらのメソッドは、引数の 1 つとして ItemHolder オブジェクトを取ります。これには、バイト配列で読み取られるデータが含まれます。このバイト配列のストレージは、CICS によって作成され、プログラムの終わりにガーベッジ・コレクションされます。

## WRITEQ TS

Java バイト配列で一時記憶域キューに書き込まれるデータを提供する必要があります。NOSPACE 条件が検出される場合、writeItem() および rewriteItem() メソッドは中断し、データをキューに書き込むためのスペースが使用可能になるまで待機します。writeItemConditional() および rewriteItemConditional() メソッドは、NOSPACE 条件の場合に中断しませんが、この条件を NoSpaceException としてアプリケーションに即時に戻します。

## 端末サービス

JCICS は、以下の CICS 端末サービス・コマンドをサポートします。

| メソッド       | JCICS クラス                 | EXEC CICS コマンド |
|------------|---------------------------|----------------|
| converse() | TerminalPrincipalFacility | CONVERSE       |
|            | サポートされていない                | HANDLE AID     |
| receive()  | TerminalPrincipalFacility | RECEIVE        |
| send()     | TerminalPrincipalFacility | SEND           |
|            | サポートされていない                | WAIT TERMINAL  |

タスクに基本機能として端末が割り振られている場合、CICS は、標準出力と標準エラー・ストリームとして使用できる 2 つの Java PrintWriter コンポーネントを自動的に作成します。これらのコンポーネントはタスク端末にマップされます。out および err という名前の 2 つのストリームは Task オブジェクトのパブリック・ファイルであり、System.out や System.err と同様に使用できます。

端末に送られるデータは、Java バイト配列で提供されなければなりません。データは端末から DataHolder オブジェクトに読み込まれます。CICS では返されるデータ用のストレージが提供されます。これは、プログラムの終了時に割り振り解除されます。

## データと XML の間の変換

JCICS は、データから XML への変換とその逆の変換を実行するための API コマンドをサポートしています。それらのコマンドは、EXEC CICS TRANSFORM DATATOXML コマンドおよび TRANSFORM XMLTODATA コマンドと同等の機能を提供します。

| メソッド       | JCICS クラス      | EXEC CICS コマンド                                    |
|------------|----------------|---------------------------------------------------|
| SetName    | XmlTransform   | TRANSFORM<br>DATATOXML,<br>TRANSFORM<br>XMLTODATA |
| dataToXML  | Transform      | TRANSFORM<br>DATATOXML                            |
| xmltoData  | Transform      | TRANSFORM<br>XMLTODATA                            |
| setChannel | TransformInput | TRANSFORM<br>DATATOXML,<br>TRANSFORM<br>XMLTODATA |

| メソッド                | JCICS クラス       | EXEC CICS コマンド                                       |
|---------------------|-----------------|------------------------------------------------------|
| setDataContainer    | TransformInput  | TRANSFORM<br>DATATOXML<br><br>TRANSFORM<br>XMLTODATA |
| setElementName      | TransformInput  | TRANSFORM<br>DATATOXML,<br>TRANSFORM<br>XMLTODATA    |
| setElementNamespace | TransformInput  | TRANSFORM<br>DATATOXML,<br>TRANSFORM<br>XMLTODATA    |
| setNsContainer      | TransformInput  | TRANSFORM<br>XMLTODATA                               |
| setTypeNames        | TransformInput  | TRANSFORM<br>DATATOXML,<br>TRANSFORM<br>XMLTODATA    |
| setTypeNamespace    | TransformInput  | TRANSFORM<br>DATATOXML,<br>TRANSFORM<br>XMLTODATA    |
| setXmlContainer     | TransformInput  | TRANSFORM<br>DATATOXML,<br>TRANSFORM<br>XMLTODATA    |
| setXmltransform     | TransformInput  | TRANSFORM<br>DATATOXML,<br>TRANSFORM<br>XMLTODATA    |
| getElementName      | TransformOutput | TRANSFORM<br>DATATOXML,<br>TRANSFORM<br>XMLTODATA    |
| getElementNamespace | TransformOutput | TRANSFORM<br>DATATOXML,<br>TRANSFORM<br>XMLTODATA    |
| getTypeNames        | TransformOutput | TRANSFORM<br>DATATOXML,<br>TRANSFORM<br>XMLTODATA    |
| getTypeNamespace    | TransformOutput | TRANSFORM<br>DATATOXML,<br>TRANSFORM<br>XMLTODATA    |

## 一時データ・キュー・サービス

JCICS は、CICS 一時データ・コマンド DELETEQ TD、READQ TD、および WRITEQ TD をサポートします。INTO オプションを除くすべてのオプションがサポートされます。

### JCICS メソッドと EXEC CICS コマンド間の対話

Java プログラムが既存の CICS アプリケーション・データにアクセスできるようにするツールについては、Java からの構造化データとの対話を参照してください。

表 59 は、CICS 一時データ・コマンドにマップされるメソッドと JCICS クラスをリストしています。

表 59. メソッド、JCICS クラスおよび CICS コマンド間の関係

| メソッド                              | JCICS クラス | EXEC CICS コマンド |
|-----------------------------------|-----------|----------------|
| delete()                          | TDQ       | DELETEQ TD     |
| readData(), readDataConditional() | TDQ       | READQ TD       |
| writeData()                       | TDQ       | WRITEQ TD      |

### DELETEQ TD

TDQ クラスの delete() メソッドを使用して一時データ・キュー (TDQ) を削除することができます。

### READQ TD

CICS INTO オプションは Java プログラムではサポートされません。TDQ クラスの readData() または readDataConditional() メソッドを使用して TDQ から読み取ることができます。これらのメソッドは、バイト配列で読み取られるデータが入っている DataHolder オブジェクトのインスタンスをパラメーターとして取ります。このバイト配列のストレージは、CICS によって作成され、プログラムの終わりにガーベッジ・コレクションされます。

readDataConditional() メソッドは、CICS NOSUSPEND ロジックを駆動します。QBUSY 条件が検出されると、QueueBusyException として即時にアプリケーションに戻されます。

readData() メソッドは、別のタスクによって使用中のレコードにアクセスしようとするときに、コミットされたレコードがそれ以上ない場合は中断します。

### WRITEQ TD

Java バイト配列で TDQ に書き込まれるデータを提供する必要があります。

## 作業単位 (UOW) サービス

JCICS は、CICS SYNCPOINT サービスをサポートします。

表 60. UOW サービスの JCICS と EXEC CICS コマンド間の関係

| メソッド                 | JCICS クラス | EXEC CICS コマンド |
|----------------------|-----------|----------------|
| commit(), rollback() | タスク       | SYNCPOINT      |

Liberty JVM サーバーでは、UOW 同期点処理を Java Transaction API (JTA) を使用して制御できます。詳しくは、766 ページの『Java Transaction API (JTA)』を参照してください。

## Web サービスの例

JCICS は、アプリケーション内で Web サービスを操作するために使用できるすべての API コマンドをサポートします。

| メソッド             | JCICS クラス  | EXEC CICS コマンド               |
|------------------|------------|------------------------------|
| invoke()         | WebService | INVOKE WEBSERVICE            |
| create()         | SoapFault  | SOAPFAULT CREATE             |
| addFaultString() | SoapFault  | SOAPFAULT ADD<br>FAULTSTRING |
| addSubCode()     | SoapFault  | SOAPFAULT ADD<br>SUBCODESTR  |
| delete()         | SoapFault  | SOAPFAULT DELETE             |
| create()         | WSAEpr     | WSAEPR CREATE                |
| delete()         | WSAContext | WSACONTEXT DELETE            |
| set*()           | WSAContext | WSACONTEXT BUILD             |
| get*()           | WSAContext | WSACONTEXT GET               |

次の例に、JCICS を使用して Web サービス要求を作成する方法を示します。

```
Channel requesterChannel = Task.getTask().createChannel("TestRequester");
Container appData = requesterChannel.createContainer("DFHWS-DATA");
byte[] exampleData = "ExampleData".getBytes();
appData.put(exampleData);

WebService requester = new WebService();
requester.setName("MyWebservice");
requester.invoke(requesterChannel, "myOperationName");

byte[] response = appData.get();
```

Web サービス要求で送受信されるアプリケーション・データを処理する際に、構造化データを処理している場合、IBM Record Generator for Java などのツールを使用してクラスを生成できます。821 ページの『Java からの構造化データとの対話』を参照してください。また、Java を使用して XML の生成とコンシュームを直接行うこともできます。

## JCICS の使用

JCICS ライブラリーのクラスは、Java クラスと同じように使用します。アプリケーションで、必要なタイプの参照を宣言し、new 演算子を使用してクラスの新しいインスタンスを作成します。

基礎の CICS リソースの名前を渡すために、setName メソッドを使用して CICS リソースの名前を指定します。リソースを作成した後、標準的な Java 構成を使用してオブジェクトを操作します。宣言したオブジェクトのメソッドを通常の方法で呼び出します。クラスごとにサポートされるメソッドの詳細は、提供される Javadoc で入手可能です。

CICS は、PROGRAM リソースの JVMCLASS 属性で指定されるクラスで、`main(CommAreaHolder)` の署名を使用するメソッドに制御を渡そうとします。このメソッドが見つからない場合、CICS は、`main(String[])` メソッドを呼び出そうとします。

詳しくは、Java の制約事項およびJCICS Javadoc 情報を参照してください。

この例では、TSQ オブジェクトを作成し、作成したばかりの一時記憶域キュー・オブジェクトで `delete` メソッドを呼び出し、キューが空の場合はスローされた例外をキャッチする方法を示しています。

```
// Define a package name for the program
package unit_test;

// Import the JCICS package
import com.ibm.cics.server.*;

// Declare a class for a CICS application
public class JCICSTSQ
{
 // The main method is called when the application runs
 public static void main(CommAreaHolder cah)
 {
 try
 {
 // Create and name a Temporary Storage queue object
 TSQ tsq = new TSQ();
 tsq.setName("JCICSTSQ");

 // Delete the queue if it exists
 try
 {
 tsq.delete();
 }
 catch(InvalidQueueIdException e)
 {
 // Absorb QIDERR
 System.out.println("QIDERR ignored!");
 }

 // Write an item to the queue
 String transaction = Task.getTask().getTransactionName();
 String message = "Transaction name is - " + transaction;
 tsq.writeItem(message.getBytes());
 }
 catch(Throwable t)
 {
 System.out.println("Unexpected Throwable: " + t.toString());
 }

 // Return from the application
 return;
 }
}
```

## Java の制約事項

CICS 用の Java アプリケーションを開発する際に、プログラマーが知っておくべきいくつかの制約事項があります。

CICS で使用される Java アプリケーションには、以下の制約事項が適用されます。

- `System.exit()` メソッドを使用しないでください。アプリケーションが JVM サーバーで実行されているときにこのメソッドを使用すると、JVM サーバーが強制終了し、CICS 静止し、データの不一致が生じる可能性があります。`System.exit()` の使用を禁止するには、Java セキュリティー・ポリシーを使用します。関連情報については、Java セキュリティー・マネージャーの有効化を参照してください。
- JCICS API 呼び出し: これらの呼び出しを OSGi バンドルのアクティベーター・クラスで使用することはできません。

注: OSGi バンドル・アクティベーターを実行する Java スレッドは、JCICS 対応ではありません。開発者が `CICSExecutorService.runAsCICS()` API を使用して、アクティベーターから新しい JCICS 対応スレッドを開始することは可能です。JCICS コマンドはすべて、インストール・コマンドを実行したユーザー ID の権限で実行されます。そのため、管理者は、OSGi バンドル・アクティベーターをインストールする前に、その中で使用されるリソースを把握しておく必要があります。`runAsCICS()` API については、741 ページの『スレッドとタスクのサンプル』でさらに詳しく説明しています。

- OSGi バンドル・アクティベーターで使われる `start` メソッドおよび `stop` メソッド: これらのメソッドは、適切な時間内に戻す必要があります。
- スレッド間で JCICS オブジェクトを共用しないでください。JCICS オブジェクトに対するインスタンス・メソッドの呼び出しが許可されるのは、そのオブジェクトを作成したスレッドのみです。
- CICS Java プログラムでファイナライザーを使用しないでください。ファイナライザーを推奨しない理由の解説については、Troubleshooting and supportを参照してください。

---

## OSGi の使用に関するガイダンス

OSGi アプリケーションを開発する際は、いくつかの考慮事項があります。

### 依存関係の定義

OSGi バンドルで別の OSGi バンドルからの Java パッケージを使用する場合、2 つのバンドル間のインターフェースを明示的に表現する必要があります。別のバンドルからのパッケージを使用するバンドルでは、`manifest.mf` 内の **Import-Package** ステートメントにそのパッケージを追加する必要があります。パッケージを提供するバンドルでは、`manifest.mf` 内の **Export-Package** ステートメントに、提供するパッケージを追加する必要があります。両方の OSGi バンドルを環境にデプロイすると、この依存関係を解決できます。

JRE 拡張 (**javax.\*** など) を含め、OSGi バンドルで使用されるすべてのパッケージは、明示的にインポートされる必要があります。これは、ランタイムがこれらのパッケージをブート委任などの他の手段で検出する場合にも適用されます。デフォルトでは、コア **java.\*** パッケージのみが使用可能になると想定してください。

依存関係を表現するには、別の手段もあります。その 1 つは、バンドル・ヘッダー **Require-Bundle** です。ただし、**Require-Bundle** はより粗視化されており、ユーザーを特定のバンドルに関連付けます。**Require-Bundle** を使用すると、アーキテク

ャーに柔軟性がなくなり、パッケージを独立してバージョン管理する機能も制限されます。

## JRE クラスの可視性、ブート委任、および `system.packages.extra`

OSGi では、コアとなる JRE パッケージ/クラス (`java.*`) のロードは常にブートストラップ・クラス・ローダーに委任されます。システム内に JRE は 1 つのみ存在することが前提となっているため、明示的な依存関係ステートメントは必要ありません。そのため、`java.*` 依存関係をバンドル・マニフェストに追加する必要は一切ありません。ただし、JRE の他の部分では、これらのパッケージを必要とするアプリケーション・バンドルで `Import-Package` ステートメントをコーディングする必要があります。例えば、ベンダー固有の拡張である `javax.*`、`com.sun.*`、および `com.ibm.*` にはインポートが必要です。その理由は、これらはブートストラップ・クラス・ローダーに委任されるのではなく、OSGi システムの一部として扱われるためです。

OSGi フレームワークは、既知の拡張パッケージをシステムに自動的に公開するシステム・バンドルを提供します。OSGi バンドルで提供される他のすべてのパッケージと同様に、アプリケーション・バンドルは `Import` ステートメントを組み込むことによって、自身の依存関係を登録します。この手法の利点は、新しいコードが含まれる OSGi バンドルをインストールすることによって、拡張パッケージを新しい実装で置き換えられることです。

このプロセスの例外となるのは、特殊な OSGi プロパティを使用して特定のパッケージがブート委任リストに追加されている場合です。この方法は (パッケージにアクセスするために `Import` ステートメントが必要にならないため) 便利ではあるものの、OSGi の柔軟性が制限されてしまうため、ベスト・プラクティスであるとはみなされません。ベンダー固有の拡張パッケージの中には、OSGi 実装で自動的にシステム・バンドルに追加されないものもあります。そのような場合は、パッケージは本来 JRE から利用可能になるという前提に基づい

て、`-Dorg.osgi.framework.system.packages.extra` プロパティを使用してパッケージをシステム・バンドルに追加し、アプリケーションの `Import` で解決できるようにします。

## バンドル・アクティベーター

バンドル・アクティベーターは、OSGi バンドルに含まれる **BundleActivator** インターフェースを実装するクラスです。アクティベーターを使用するには、OSGi バンドルのバンドル・マニフェスト内で **Bundle-Activator** ヘッダーを使用して、アクティベーターを宣言する必要があります。**BundleActivator** インターフェースには、作業を初期化または終了するために使用できる `start` メソッドおよび `stop` メソッドがあります。一般的なパターンは、アプリケーション内で使用するサービス依存関係を検索することです。ただし、コンポーネントとそのサービス依存関係をアクティブにするには、宣言型サービスなどのコンポーネント・モデルを使用することのほうが推奨されます。

## シングルトン・バンドル

シングルトン・バンドルは、他のバージョンのバンドルがメモリーにロードされないようにするために使用されます。これにより、どの時点においてもランタイムに

存在できる解決済みバージョンは 1 つのみになります。一連のアプリケーションから単一のシステム・リソースにアクセスしなければならない場合は、シングルトン・バンドルを使用することが望ましいです。

## OSGi バンドルのフラグメント

フラグメントとは、OSGi フレームワークによって動的にホスト・バンドルに接続される OSGi バンドルのことです。フラグメントはホスト・バンドルのクラス・ローダーを共用し、バンドルのライフサイクルには参加しません。そのため、フラグメントではバンドル・アクティベーターをサポートしていません。フラグメントの一般的なユース・ケースは、バンドルのパッチとして使用することです。

**Bundle-ClassPath** で `.` の前にフラグメントを指定すると、クラスをホストからではなくフラグメントから優先的にロードできます。

## OSGi サービス・レジストリー

OSGi サービス・レジストリーにより、バンドルがオブジェクトを共用レジストリーに公開することが可能になります。サービスは、Java インターフェースで公示され、OSGi 環境にインストールされている他のバンドルで使用可能になります。

## マイクロサービス (μServices)

マイクロサービスは、小さな独立したコンポーネントを組み合わせて複合アプリケーションを構成し、これらのコンポーネントが特定の言語に依存しない API を使用して互いに通信するソフトウェア・アーキテクチャー・スタイルです。サイズが小さく、非常に細かく分離され、小さいタスクを処理することに重点を置くマイクロサービスを使用することで、モジュラー手法のシステム構築が容易になります。OSGi コンポーネントの間で `μService` を使用すると、バンドルのワイヤリングのみでは達成できない柔軟性および動的更新機能が実現します。このことから、バンドルのワイヤリングよりも `μService` を優先して使用することが推奨されます。

## バンドルおよびパッケージのバージョン管理

OSGi でのパッケージ・バージョン管理に望ましい手法は、セマンティック・バージョン管理モデルです。バージョン番号が MAJOR.MINOR.PATCH だとすると、以下のように増分します。

1. 互換性のない API 変更を行った場合は、バージョンのメジャー部分 (MAJOR) を増分します。
2. 前のバージョンと互換性のある機能を追加した場合は、バージョンのマイナー部分 (MINOR) を増分します。
3. 前のバージョンと互換性のあるバグ修正を行った場合は、バージョンのパッチ部分 (PATCH) を増分します。

## 実行環境

実行環境 (EE) は、JRE のシンボル表現です。次に例を示します。

**Bundle-RequiredExecutionEnvironment:** JavaSE-1.7

必要なすべての機能を提供する最小バージョンの EE を使用する必要があります。新しい OSGi バンドルを作成する際は、通常はアクティブに保守されている最新の

Java 実行環境で作成するのが適切です。特殊なアプリケーションでそれよりも低いバージョンが必要となる場合に限り、低いレベルに設定します。特定の EE を選択したら、バージョン・アップに明らかな利点がない限り、バージョンを変更してはなりません。EE のバージョンを高くしても、コードが新しい警告にさらされたり、非推奨が生じたりするなど、必要な作業が増えるのみで実質的な価値はありません。

---

## Liberty JVM サーバーで実行する Java アプリケーションの開発

WebSphere® Application Server Liberty を使用する Java EE アプリケーションをデプロイする場合は、Web コンテナを実行するように Liberty JVM サーバーを構成します。

### 開発環境のセットアップ

JCICS API を使用する Java EE アプリケーションを開発するには、CICS Explorer および IBM CICS SDK for Java EE and Liberty をインストールする必要があります。

#### このタスクについて

CICS Liberty JVM サーバーでホストされている Java EE アプリケーションは、以下の一連のツールを使用して開発、パッケージ、およびデプロイできます。

- Eclipse と Eclipse Web Tools Platform には、Java EE アプリケーションを開発するためのツールが用意されています。
- IBM CICS SDK for Java EE and Liberty は、JCICS、Java EE、および Liberty の各 API を、Java ビルド・パス・ライブラリーまたは OSGi ターゲット・プラットフォームの形式で提供します。
- CICS Explorer。CICS バンドル内で Java EE アプリケーションをパッケージ、デプロイ、および管理するためのツールを提供します。
- Explorer for z/OS。JVM サーバー・ログ・ファイルの表示など、z/OS 上でファイル、データ・セット、およびジョブを処理するためのツールを提供します。

これらのツールは、CICS Explorer のインストールの一部として一緒にインストールできます。

#### 手順

1. CICS Explorer のインストールについて詳しくは、CICS Explorer 製品資料内の『CICS Explorer のダウンロードおよび開始』および CICS Explorer 製品資料内の『CICS SDK for Java EE and Liberty のインストール』を参照してください。
2. 開発環境を再始動します。

#### タスクの結果

これで、Liberty JVM サーバーで Java EE アプリケーションを開発、デプロイ、およびテストするための開発環境が整いました。

## 次のタスク

Java EE アプリケーションの開発を開始したり、JCICS、Java EE、および Liberty API を使用したりすることができます。IBM CICS SDK for Java に用意されているサンプルを使用して、作業を始めることができます。詳しくは、Java サンプル: サブレットの例を参照してください。

## Java EE アプリケーションと Liberty アプリケーション

CICS アプリケーションに対する最新のインターフェースを提供するために、Web アプリケーション・テクノロジーを使用するプレゼンテーション層を開発できます。Eclipse ベースの Web 開発ツールは、これらのアプリケーションを作成するための開発プラットフォームを提供します。オプションで CICS Explorer にインストールされる IBM CICS SDK for Java EE and Liberty は、CICS での実行用にこれらをビルドし、パッケージ化し、デプロイするためのサポートを提供します。

### このタスクについて

Liberty サーバーにデプロイできる Web アプリケーション・プロジェクトには、次の 3 つのタイプがあります。

- 動的 Web プロジェクト (WAR)
- OSGi アプリケーション・プロジェクト (EBA)
- エンタープライズ・アプリケーション・プロジェクト (EAR)

WAR には、イメージや HTML ファイルのような静的リソースの他に、CICS での Liberty、フィルター、関連メタデータなどの動的 Java EE リソースを含めることができます。

EBA は、WAB と OSGi バンドルを含めることができる Java アーカイブ・ファイルです。WAB は Web 使用可能 OSGi バンドルで、イメージや HTML ファイルのような静的リソースの他に、JSP サブレットとファイル、フィルター、関連メタデータなどが含まれます。

EAR は、EBA が WAB と OSGi バンドルを編成するのと同じ方法で、WAR と EJB モジュールを 1 つのコンテナに編成する方法です。

### 動的 Web プロジェクトの作成

Java アプリケーション用の Web プレゼンテーション層を開発する場合、動的 Web プロジェクトを作成できます。

#### 始める前に

Eclipse IDE に Web 開発ツールがインストールされていることを確認します。詳しくは、751 ページの『開発環境のセットアップ』を参照してください。

Liberty によって追加された制限により、WAR ファイルにデプロイされたサブレットから OSGi バンドルにアクセスすることはできません。この制限には、CICS バンドルによって直接インストールされた OSGi バンドルへのアクセスも含まれます。この制限を回避するには、アプリケーションを EBA (OSGi アプリケーション・プロジェクト) に含まれる WAB としてデプロイする必要があります。EBA は、Web と OSGi コンポーネントが対話できるコンテナです。

## このタスクについて

CICS Explorer および IBM CICS SDK for Java のヘルプには、Web アプリケーションを開発してパッケージ化する以下のステップの実行方法が詳細に説明されています。

### 手順

1. アプリケーション用に 動的 Web プロジェクトを作成します。ビルド・パスを更新して Liberty のライブラリーを追加する必要があります。
  - a. 動的 Web プロジェクトを右クリックし、「ビルド・パス」 > 「ビルド・パスの構成」をクリックします。プロジェクトの「プロパティ」ダイアログが開きます。
  - b. 「Java のビルド・パス」で、「ライブラリー」タブをクリックします。
  - c. 「ライブラリーの追加」をクリックし、「**Liberty JVM** サーバー・ライブラリー」を選択します。
  - d. 「次へ」をクリックして CICS のバージョンを選択し、「終了」をクリックしてライブラリーの追加を完了します。
  - e. 「**OK**」をクリックして、変更内容を保管します。
2. Web アプリケーションを開発します。JCICS API を使用して CICS サービスに、JDBC を使用して DB2® に、また、JMS を使用して IBM MQ にアクセスできます。IBM CICS SDK for Java EE and Liberty には、JCICS と JDBC を使用した Web コンポーネントの例が含まれています。
3. オプション: CICS セキュリティーでアプリケーションを保護する場合は、動的 Web プロジェクトに web.xml ファイルを作成して、CICS セキュリティー制約を組み込みます。IBM CICS SDK for Java EE and Liberty には、CICS 用の正しい情報が入った、このファイル用のテンプレートが含まれます。詳しくは、Liberty JVM サーバーでのユーザー認証 を参照してください。
4. アプリケーションをパッケージするために、1 つ以上の CICS バンドル・プロジェクトを作成します。CICS リソースの定義とインポートを追加します。各 CICS バンドルには、ID とバージョンが含まれており、変更を細かく管理できます。
5. オプション: URI からのインバウンド Web 要求をマップして特定のトランザクションの下で実行する場合は、URIMAP および TRANSACTION リソースを CICS バンドルに追加します。これらのリソースを定義しない場合、CJSA という名前の提供されたトランザクションの下ですべての処理が実行されます。これらのリソースは、動的にインストールされ、CICS の中のバンドルの一部として管理されます。

### タスクの結果

開発環境のセットアップ、動的 Web プロジェクトからの Web アプリケーションの作成、そのアプリケーションをデプロイするためのパッケージ化が終わりました。

### 次のタスク

アプリケーションをデプロイする用意ができれば、CICS バンドル・プロジェクトを zFS にエクスポートします。参照されたプロジェクトは、ビルドされ、zFS への転

送に組み込まれます。または、Liberty デプロイメント・モデルに従って、アプリケーションを WAR としてエクスポートし、稼働中の Liberty JVM サーバーの dropins ディレクトリーにそれをデプロイすることもできます。

## OSGi アプリケーション・プロジェクトの作成

OSGi アプリケーション・プロジェクト (EBA) とは、一連のバンドルをまとめたものです。アプリケーションは、さまざまな種類の OSGi バンドルで構成できます。

### 始める前に

Eclipse IDE に WebSphere Application Server Developer Toolsがインストールされていることを確認します。詳しくは、751 ページの『開発環境のセットアップ』を参照してください。

### このタスクについて

CICS Explorer および IBM CICS SDK for Java のヘルプには、Web アプリケーションを開発してパッケージ化する以下のステップの実行方法が詳細に説明されています。

### 手順

1. 「CICS TS 5.5 with Java EE and Liberty」テンプレートを使用して、Java 開発用のターゲット・プラットフォームをセットアップします。ターゲットが、Eclipse の現行インストールよりも新しいバージョンであるという警告が表示されることがありますが、この警告メッセージは無視してかまいません。
2. アプリケーション用に OSGi バンドル・プロジェクトを作成します。ターゲット・プラットフォームでパッケージが実質的に使用可能になるため、適切な Import ステートメントをバンドル・マニフェストに含める必要があります。Web 使用可能 OSGi バンドル・プロジェクトは、動的 Web プロジェクトと同等のバンドルです。Web 使用可能 OSGi バンドル・プロジェクトを使用して、アプリケーションを OSGi アプリケーション・プロジェクト (エンタープライズ・バンドル・アーカイブまたは EBA ファイル) 内にデプロイできます。Web 使用可能 OSGi バンドル・プロジェクト (WAB ファイル) および Web 使用可能でない OSGi バンドル・プロジェクトを OSGi アプリケーション・プロジェクト内で混合できます。通常、Web 使用可能 OSGi バンドル・プロジェクトはアプリケーションのフロントエンドを実装し、(ビジネス・ロジックを含んでいる) 非 Web OSGi バンドルと対話します。
3. Web アプリケーションを開発します。JCICS API を使用して CICS サービスおよび JDBC にアクセスし、DB2 に接続することができます。IBM CICS SDK for Java には、JCICS と DB2 を使用する、Web コンポーネントと OSGi バンドルの例が含まれます。ビジネスとプレゼンテーションのロジックを分離するために、JCICS を使用する OSGi バンドルを作成してください。また、OSGi バンドルの中でセマンティック・バージョン管理を使用して、アプリケーションのビジネス・ロジックの更新を管理できます。JDBC DriverManager インターフェースを介して Db2 を使用する WAB または OSGi バンドルごとに、com.ibm.db2.jcc の Import-Package ヘッダーをバンドル・マニフェストに含めます。このインポートを省略すると、「java.sql.SQLException: jdbc:default:connection に適切なドライバーが見つかりません (java.sql.SQLException: No suitable driver found for

jdbc:default:connection)」というエラー・メッセージが出されます。JDBC DataSource インターフェースを使用する場合、このインポートは不要です。

4. オプション: Web アプリケーションのユーザーを認証したい場合は、Web プロジェクトに web.xml ファイルを作成して、セキュリティ制約を含めます。  
IBM CICS SDK for Java には、CICS 用の正しい情報が入った、このファイル用のテンプレートが含まれます。詳しくは、Liberty JVM サーバーでのユーザー認証を参照してください。
5. OSGi バンドルを参照する OSGi アプリケーション・プロジェクトを作成します。
6. OSGi アプリケーション・プロジェクトを参照する CICS バンドル・プロジェクトを作成します。CICS リソースの定義とインポートを追加することもできます。各 CICS バンドルには、ID とバージョンが含まれており、変更を細かく管理できます。
7. オプション: URI からのインバウンド Web 要求をマップして特定のトランザクションの下で実行する場合は、URIMAP および TRANSACTION リソースを CICS バンドルに追加します。これらのリソースを定義しない場合、CJSA という名前の提供されたトランザクションの下ですべての処理が実行されます。これらのリソースは、動的にインストールされ、CICS の中のバンドルの一部として管理されます。

## タスクの結果

これで、開発環境をセットアップし、OSGi Web アプリケーションを作成し、それをデプロイメント用にパッケージしました。

## 次のタスク

アプリケーションをデプロイする用意ができたなら、CICS バンドル・プロジェクトを zFS にエクスポートします。参照されたプロジェクトは、ビルドされ、zFS への転送に組み込まれます。または、開発デプロイメント・モデルに従って、アプリケーションを EBA ファイルとしてエクスポートし、稼働中の Liberty JVM サーバーの dropins ディレクトリーにそれをデプロイすることもできます。dropins を使用する場合は、セキュリティおよび他のサービス品質は構成できないことに注意してください。

## エンタープライズ・アプリケーション・プロジェクトの作成

Enterprise Java Bean モジュール (EJB モジュール) などのコンポーネントを開発する場合や、Web プロジェクトまたは EJB、あるいはその両方をグループ化するには、エンタープライズ・アプリケーション・プロジェクトを使用できます。

## 始める前に

Eclipse IDE に Web 開発ツールがインストールされていることを確認します。詳しくは、751 ページの『開発環境のセットアップ』を参照してください。

## このタスクについて

CICS Explorer および IBM CICS SDK for Java のヘルプには、エンタープライズ・アプリケーションを開発してパッケージ化するための次の各ステップの実行方法が詳細に説明されています。

### 手順

1. エンタープライズ・アプリケーション・プロジェクトを作成します。
2. アプリケーションのコンポーネントを開発します。これらのコンポーネントは、通常、EJB モジュールと動的 Web プロジェクトです。開発したコンポーネントをエンタープライズ・アプリケーション・プロジェクトに追加します。詳しくは、Enterprise JavaBeans (EJB) プロジェクトの作成を参照してください。
3. エンタープライズ・アプリケーションをパッケージ化するために、1 つ以上の CICS バンドル・プロジェクトを作成します。CICS リソースの定義とインポートを追加します。すべての CICS バンドルには、ID とバージョンが含まれており、変更を細かく管理できます。
4. オプション: URI からのインバウンド Web 要求をマップして特定のトランザクションの下で実行する場合は、URIMAP および TRANSACTION リソースを CICS バンドルに追加します。これらのリソースを定義しない場合、CJSA という名前の提供されたトランザクションの下ですべての処理が実行されます。これらのリソースは、動的にインストールされ、CICS の中のバンドルの一部として管理されます。

### タスクの結果

これで、開発環境をセットアップし、エンタープライズ・アプリケーション・プロジェクトを作成し、それをデプロイメント用にパッケージ化しました。

### 次のタスク

アプリケーションをデプロイする用意ができれば、CICS バンドル・プロジェクトを zFS にエクスポートします。参照されたプロジェクトは、ビルドされ、zFS への転送に組み込まれます。あるいは、Liberty デプロイメント・モデルに従って、アプリケーションを EAR としてエクスポートして、その EAR を <application> 要素と一緒にデプロイするか、または稼働中の Liberty JVM サーバーの drop-ins ディレクトリーに格納することもできます。

### URI マップとトランザクションの作成

CSD や BAS などの従来の方法でアプリケーション・リソースをインストールすることも、アプリケーション・リソースを CICS バンドルに追加することもできます。CICS バンドルは、アプリケーション・コードと CICS リソースをグループ化して同じ場所に配置する便利な手法です。この手法は、例えば Java EE アプリケーションを CICS バンドルにデプロイする場合に役立ちます。インバウンド Web 要求をマップする URI マップを指定し、特定のアプリケーション・トランザクションでそれらの要求を実行できます。

## 始める前に

アプリケーション・リソースを作成するには、プロジェクト・エクスプローラーに CICS バンドル・プロジェクトが存在していなければなりません。詳細については、CICS Explorer 製品資料内の『CICS バンドル・プロジェクトの作成』を参照してください。この CICS バンドル・プロジェクトを使用して、デプロイするアプリケーションをパッケージします。

## このタスクについて

デフォルトでは、すべての Java EE アプリケーション要求で、CICS によって提供される CJSA というトランザクションが使用されます。ただし、インバウンド要求のアプリケーション URI を別のトランザクションにマップすることもできます。このフィーチャーは、アプリケーションへのアクセスを安全な方法で制御する上で役立つ場合があります。セキュリティ管理者は、ユーザーがアクセスできるトランザクションを制御するように CICS を構成できるからです。

## 手順

1. 以下のようにして、アプリケーション・トランザクションの定義を作成します。
  - a. Eclipse リソース・パースペクティブに切り替えます。CICS バンドル・プロジェクトを右クリックして、「新規」 > 「トランザクション定義」をクリックします。「新規トランザクション定義 (New Transaction Definition)」ウィザードが開きます。
  - b. トランザクションの 4 文字の名前を入力します。トランザクション名の先頭を C にしないでください。この文字は CICS で予約されているためです。
  - c. プログラム名 DFHSJTHP を入力します。この CICS プログラムを使用する必要があるのは、Liberty サーバーへのインバウンド Java EE 要求のセキュリティ検査がこのプログラムで扱われるためです。
  - d. 「終了」をクリックして、定義を CICS バンドル・プロジェクトに作成します。

アプリケーション・トランザクションは Java EE アプリケーションが実行されている CICS 領域で常に実行される必要があるため、リモート・トランザクションを作成するための属性を設定しないでください。

2. 以下のようにして、URI マップの定義を作成します。
  - a. CICS バンドル・プロジェクトを右クリックして、「新規」 > 「URI マップ定義」をクリックします。
  - b. URI マップの 8 文字の名前を入力します。URI マップ名の先頭を DFH にしないでください。この接頭部は CICS で予約されているためです。
  - c. ホスト名を入力します。\* を使用すると任意のホスト名と一致させることができます。あるいは、アプリケーションが実行されるマシンのホスト名を指定することもできます。
  - d. アプリケーション URI のパスを入力します。CICS は、インバウンド要求内の URI を、URI マップ内の値と突き合わせ、アプリケーション・トランザクションを実行します。

- e. 「使用法」セクションで、「JVM サーバー」を選択し、オプションでポート番号を入力します。
  - f. 「終了」をクリックして、URI マップを作成します。
3. 以下のようにして、URI マップ定義を編集します。
- a. 「スキーム」フィールドを編集して、URI マップのスキームを入力します。HTTP がデフォルトですが、要求を暗号化するために SSL セキュリティーを使用する場合は、HTTPS を設定することができます。HTTP 要求と HTTPS 要求の両方で、HTTP ヘッダーにユーザー ID およびパスワードが指定される基本認証を使用することができます。
  - b. 「トランザクション」フィールドを編集して、アプリケーション・トランザクションの名前を入力します。
  - c. オプション: 「ユーザー ID」フィールドを編集して、アプリケーション要求を実行するユーザー ID を入力します。この値は、基本認証が使用可能である場合には無視されます。値を指定せず、HTTP 要求にユーザー ID とパスワードが含まれない場合、CICS は CICS 領域のデフォルト・ユーザー ID の下で要求を実行します。

## タスクの結果

URI マップとトランザクションが CICS バンドル・プロジェクトに作成されました。バンドルがデプロイおよびインストールされると、これらのリソースが CICS 領域に動的に作成されます。

## 次のタスク

複数の異なるトランザクションを使ってさまざまなアプリケーション操作を実行する場合、または HTTP スキームと HTTPS スキームの両方をサポートする場合は、さらにリソースを作成することができます。アプリケーションを配置する準備ができれば、CICS Explorer 製品資料内の『CICS バンドルのデプロイ』を参照してください。

## Liberty JVM サーバー内で実行するよう Java EE アプリケーションをマイグレーションする

ネットワークを介して CICS にアクセスする Liberty インスタンスの中で実行する Java EE アプリケーションがある場合、そのアプリケーションを Liberty JVM サーバーの中で実行すると、パフォーマンスを最適化できます。

### このタスクについて

CICS は、Liberty で使用可能な機能のサブセットをサポートしています。CICS 統合モードの Liberty でサポートされる機能のリストについては、806 ページの『Liberty フィーチャー』を参照してください。

アプリケーションでセキュリティを使用する場合は、Liberty セキュリティー機能を引き続き使用できますが、追加アクションなしで、CICS タスクがトランザクション CJSA 下で実行され、CICS での URIMAP マッチングが使用不可になり、すべてのリソース・アクセスが CICS のデフォルト・ユーザー ID で実行される可能性があります。Liberty で決定された同じユーザー ID で CICS タスクを実行できる

ようにして、セキュリティー・ソリューションをより効率的に CICS と統合する場合は、Liberty JVM サーバーでのユーザー認証を参照してください。

## 手順

1. アプリケーションが CICS とデータを受け渡しするときに正しい JCICS エンコードが使用されることを確認して、JCICS API を使用して CICS サービスに直接アクセスするようにアプリケーションを更新します。エンコードの詳細については、719 ページの『データ・エンコード』を参照してください。このステップが適用されるのは、CICS 統合モードの Liberty または CICS 標準モードの Liberty を、runAsCICS() API とともに使用する場合があります。
2. 基本認証に CICS セキュリティーを使用する場合は、動的 Web プロジェクトの web.xml ファイルのセキュリティー制約を更新して、認証に CICS ロールを使用するようにします。このステップが適用されるのは、CICS 統合モードの Liberty または CICS 標準モードの Liberty を使用して、runAsCICS() メソッドによって作業を CICSExecutorService に実行依頼する場合のみです。

```
<auth-constraint>
 <description>All authenticated users of my application</description>
 <role-name>cicsAllAuthenticated</role-name>
</auth-constraint>
```

3. アプリケーションを WAR (動的 Web プロジェクト)、EBA (OSGi アプリケーション・プロジェクト) ファイル、または EAR (エンタープライズ・アーカイブ) ファイルとして CICS バンドルにパッケージ化します。CICS バンドルは、アプリケーションのデプロイメントの単位です。バンドルの中のすべての CICS リソースは、動的にインストールされ、一緒に管理されます。一緒に管理するアプリケーション・コンポーネントの CICS バンドル・プロジェクトを作成します。
4. CICS バンドル・プロジェクトを zFS にデプロイし、CICS バンドルを Liberty JVM サーバーにインストールします。

## タスクの結果

アプリケーションは、JVM サーバーで実行中です。

## CICS プログラムからの Java EE アプリケーションへのリンク

Liberty JVM サーバーで実行される Java EE アプリケーションへ、CICS トランザクションの初期プログラムをリンクしたり、CICS プログラムから LINK、START、または START CHANNEL コマンドを実行して呼び出したりできます。

CICS プログラムからリンクする Java EE アプリケーションは、Web アーカイブ (WAR) またはエンタープライズ・アプリケーション・アーカイブ (EAR) としてパッケージ化された Plain Java Object (POJO) でなければなりません。EJB、CDI Bean、または OSGi のアプリケーションにリンクすることはできません。@EJB を使用した EJB の注入を含め、POJO での依存性注入はサポートされません。EJB などのリソースの参照を取得するには、代わりに JNDI 検索を使用できます。この情報は、CICS 統合モードの Liberty にのみ適用されます。

CICS プログラムから Java EE アプリケーションにリンクする主な理由は、以下の 3 つです。

- Java コードが既存の Web アプリケーションの一部であり、それを CICS アプリケーションにリンクする必要がある。そうすれば、ロジックを 1 つ保守するだけで JCICS API を使用してコードから CICS リソースにアクセスできるようになる。
- CICS アプリケーションの一部として新機能を Java で作成する必要がある。例えば、既に Java の形で存在するサード・パーティーのライブラリーや API を使用する必要がある場合など。
- Java で既存の COBOL アプリケーションを再実装する必要がある。例えば、保守コストを削減したり、Java のスキルを最大限に活用したり、汎用的なプロセッサではなく特殊なエンジンでアプリケーションを実行できるようにしたりすることが必要な場合があります。

CICS プログラムから Java EE アプリケーションへリンクすると、CICS は、Liberty 内で実行されている JCA リソース・アダプターにメッセージを送信します。JCA リソース・アダプターは、呼び出し側プログラムと同じ CICS タスクで、ターゲットの Java EE アプリケーションにリンクします。Java EE アプリケーションは呼び出し側プログラムと同じ作業単位 (UOW) で実行されるので、リカバリー可能な CICS リソースに対する更新は、トランザクションの終了時にコミット/バックアウトされます。ただし、Java EE アプリケーションが呼び出される場合、JTA トランザクション・コンテキストは存在しません。アプリケーションで JTA トランザクションが開始される場合、CICS UOW をコミットするために同期点が実行され、新しい同期点が作成されます。これは、アプリケーションが REQUIRED トランザクション属性を使用して EJB を呼び出す場合など、アプリケーションの代わりにコンテナによって JTA トランザクションが開始された場合にも発生します。

ベスト・プラクティスとして、CICS プログラムによってリンクされたコードは、アプリケーションのプレゼンテーション・ロジックではなくビジネス・ロジックに組み込むようにしてください。例えば、HTTP 要求が行われないのに CICS プログラムからサーブレットをリンクしても意味がありません。

## CICS プログラムへ Java EE アプリケーションをリンクするための Liberty JVM サーバーの構成

Java EE アプリケーションのリンクに対応できるように Liberty JVM サーバーを構成するには、server.xml に `cicsts:link-1.0` フィーチャーを追加します。Java EE アプリケーションをデプロイする前に、このフィーチャーを追加してください。

### セキュリティ

CICS プログラムへ Java EE アプリケーションをリンクすると、CICS タスクのユーザー ID が Java EE アプリケーションに渡されます。Liberty はユーザー認証を行わずに、CICS から渡された ID を信頼します。ただし、Liberty はそのユーザー ID が構成済みのユーザー・レジストリーに存在するかどうかを検査します。可能であれば、Liberty で SAF レジストリーを使用してください。このレジストリーが、CICS から渡されたユーザー ID を検査するからです。SAF 以外のタイプのユーザー・レジストリーを使用する場合は、レジストリーに同じユーザー ID が存在すれば、そのユーザー ID が Java EE アプリケーションに渡されます。そのユーザー

ID がユーザー・レジストリーに存在しなければ、認証されないそのユーザー ID で Java EE アプリケーションがリンクされます。

Java EE アプリケーションをリンクするときにセキュリティーを構成するには、`server.xml` に `cicsts:security-1.0` フィーチャーを組み込みます。このフィーチャーを組み込まなければ、認証なしで Java EE アプリケーションがリンクされます。その結果、Java EE アプリケーションの許可検査がまったく行われない可能性があります。それでも、JCICS API による CICS リソースへのアクセスは、CICS タスクのユーザー ID で実行されます。

CICS プログラムに Java EE アプリケーションをリンクする際に、以下の Web セキュリティー・メカニズムは適用されません。

- Java EE の Web セキュリティー・メカニズム。`web.xml` の `<auth-constraint>` やサーブレットの `@HttpConstraint` など。
- トラスト・アソシエーション・インターセプター (TAI)。
- JVM サーバー・プロファイルの `com.ibm.cics.jvmserver.unclassified.userid` プロパティー。
- URIMAP。

Java EE アプリケーションでは、EJB を呼び出したり EJB セキュリティーを適用したりして追加の許可検査を実行できます。例えば、`@RolesAllowed` 注釈を使用してセッション Bean メソッドに対する許可検査を実行できます。EJB セキュリティーの詳細については、Java EE Tutorialを参照してください。

Liberty におけるセキュリティーについては詳しくは、Liberty JVM サーバーのセキュリティーの構成 (Configuring security for a Liberty JVM server)を参照してください。

## CICS プログラムで呼び出すための Java EE アプリケーションの準備

注釈を使用すると、Java メソッドを CICS アプリケーションで呼び出すことができます。CICS がユーザーに代わって PROGRAM リソースを作成します。Java EE アプリケーションは Liberty JVM サーバーで実行され、WAR または EAR 内にデプロイできます。

### 始める前に

呼び出す Java クラスおよびメソッドを特定し、サイトの標準と CICS 命名規則に従って、適切な CICS プログラム名を決定します。

Liberty JVM サーバーが、Java EE アプリケーションへのリンクが有効になるよう構成されていることを確認します。詳しくは、CICS プログラムから Java EE アプリケーションへのリンクを参照してください。

### 手順

1. Web プロジェクトのクラスパスに、`@CICSProgram` 注釈クラスを追加します。  
CICS Explorer を使用している場合、`@CICSProgram` 注釈は Liberty JVM サー

バー・ライブラリーの一部として提供されます。CICS Explorer を使用していない場合は、com.ibm.cics.server.invocation.annotations.jar JAR ファイルを使用する必要があります。

2. CICS が呼び出すメソッドを含めるクラスを作成します。CICS 固有のコードをアプリケーションの他の部分から切り離しておくことができるため、クラスを作成することをお勧めします。
3. 作成予定の CICS PROGRAM リソースごとにメソッドを作成します。
4. 各メソッドに @CICSProgram で注釈を付け、PROGRAM 名でパラメーターを指定します (例えば、@CICSProgram("PROGNAME"))。

CICS PROGRAM 名は、以下のようになります。

- 1 文字から 8 文字であること。
- A-Z a-z 0-9 \$ @ # パターンと一致していること。

@CICSProgram で注釈を付けた、単一のメソッドが含まれる単純なクラスの例を示します。

```
public class CustomerLinkTarget
{
 @CICSProgram("CUSTGET")
 public void getCustomer()
 {
 // do work here
 }
}
```

5. CICS Explorer を使用している場合は、Web プロジェクトの注釈処理を有効にします。注釈処理を有効にするには、次のいずれかの操作を行います。
  - 警告の下線が表示されている @CICSProgram 注釈上にカーソルを置き、クリック・フィックスを使用して注釈処理を有効にします。
  - Web プロジェクトを右クリックして「プロパティ (Properties)」を選択します。「注釈処理 (Annotation Processing)」ページを検索します。「プロジェクト固有の設定を有効にする」と「注釈処理を使用可能にする」の両方をチェックします。
6. CICS Explorer を使用している場合、自動的に検証が行われ、注釈が正しく配置されていること、および注釈が付けられているメソッドとそのメソッドが含まれるクラスが以下の要件を満たしていることが確認されます。注釈の要件は以下のとおりです。
  - メソッドに配置されていること。
  - PROGRAM 名の属性値が指定されていること。メソッドの要件は以下のとおりです。
  - 具象メソッドであること (抽象メソッドではないこと)。
  - public メソッドであること。
  - 引数がないこと。クラスの要件は以下のとおりです。
  - すべての注釈付きメソッドが静的でない限り、引数 (暗黙的または明示的を問わず) を持たないコンストラクターを使用すること。

- 最上位レベルであること (ネストされていたり、匿名であったりしないこと)。
  - 同じ PROGRAM 名で注釈が付けられた複数のメソッドが含まれていないこと。
7. 注釈付きメソッドの内容を書き込みます。多くの場合、内容には以下のステージが伴います。
- a. チャンネルからコンテナを取得します。
  - b. チャンネル内のコンテナから入力データを取得します。
  - c. データ・マッピング・コードを使用して、入力データを Java オブジェクトに変換します。
  - d. アプリケーション・ビジネス・ロジックを呼び出します。
  - e. データ・マッピング・コードを使用して、生成された Java オブジェクトを出力データに変換します。
  - f. 出力データをチャンネル内のコンテナに格納します。

@CICSProgram で注釈を付けた単一のメソッドが含まれるクラスの例と、コンテナから入力データを取得し、出力データをコンテナに格納するコードの例を次に示します。

```
public class CustomerLinkTarget
{
 @CICSProgram("CUSTGET")
 public void getCustomer()
 {
 Channel currentChannel = Task.getTask().getCurrentChannel();
 Container dataContainer = currentChannel.getContainer("DATA");

 // do work here

 Container resultContainer = currentChannel.createContainer("RESULT");
 byte[] results = null; // change this to be the result of the work
 resultContainer.put(results);
 }
}
```

8. アプリケーションをビルドします。
- CICS Explorer を使用している場合は、Web プロジェクトを右クリックして「エクスポート」->「WAR ファイル (WAR file)」を選択するか、アプリケーションが含まれる CICS バンドル・プロジェクトを右クリックして「バンドルを z/OS UNIX ファイル・システムにエクスポート (Export Bundle to z/OS UNIX file system)」を選択します。
  - CICS Build Toolkit を使用している場合は、注釈プロセッサが自動的に起動されます。
  - その他のツールを使用して Java コードをビルドする場合は、  
com.ibm.cics.server.invocation.annotations.jar JAR ファイル  
(@CICSProgram 注釈を定義しているファイル) が Java コンパイラーのクラスパスにあることを確認してください。また、  
com.ibm.cics.server.invocation.jar JAR ファイル (注釈プロセッサが含まれているファイル) が Java コンパイラーのクラスパスにあるか、あるいは **-processorpath** オプションで指定されていることも確認する必要があります。

ます。両方の JAR ファイルは、z/OS UNIX の *usshome* /lib ディレクトリーにあります。ここで、*usshome* は **USSHOME** システム初期設定パラメーターの値です。

- WAR ファイルの WEB-INF/lib ディレクトリー内にあるライブラリー JAR にクラスがパッケージ化される場合、JAR をビルドする際に生成されたメタデータをエクスポートします。CICS Explorer では、ライブラリー・プロジェクトを動的 Web プロジェクトのデプロイメント・アセンブリーに追加することでメタデータをエクスポートできます。動的 Web プロジェクトのプロパティー・ダイアログで、「デプロイメント・アセンブリー (Deployment Assembly)」ページを選択し、「追加 (Add)」ボタンをクリックして、ライブラリー・プロジェクトを選択します。CICS は、EAR ファイル内のユーティリティー JAR にパッケージ化されたクラスでは、@CICSProgram 注釈をサポートしません。

#### 9. アプリケーションをデプロイします。

### タスクの結果

アプリケーションが CICS バンドルによってインストールされる場合、CICS バンドルが有効になると、PROGRAM リソースが作成されます。アプリケーションが *server.xml* から直接インストールされるか、<application> エレメントを使用してファイルからインストールされる場合、アプリケーションがインストールされる時点で PROGRAM リソースが作成されます。

これで、以下のコマンドを使用して、別の CICS プログラムから Java プログラムにリンクできるようになりました。

```
EXEC CICS LINK PROGRAM("CUSTGET") CHANNEL()
```

### プログラムのライフサイクル

Java EE アプリケーションが Liberty にインストールされると、*cicsts:link-1.0* フィーチャーは @CICSProgram で注釈が付けられたメソッドを検索します。該当する各メソッドに対して、動的に PROGRAM リソースをインストールします。CICS バンドルを使用して Java EE アプリケーションがインストールされる場合、バンドルが有効になった時点で PROGRAM リソースが作成されます。それ以外の場合は、Liberty がアプリケーションをインストールする際に PROGRAM リソースが作成されます。

アプリケーションが削除されると、CICS は、そのアプリケーションに関連付けられ、動的にインストールされたすべての PROGRAM リソースを削除します。アプリケーションが CICS バンドルを使用してインストールされた場合、バンドルが無効にされると、CICS がそのプログラムを削除します。アプリケーションを呼び出したタスクがまだ進行している間にアプリケーションが削除されると、エラーが発生する可能性があります。したがって、Java EE アプリケーションを削除する前に、そのアプリケーションに関連付けられているすべての PROGRAM リソースを無効にし、作業をドレインさせる必要があります。そうしなければ、Liberty がアプリケーションをアンインストールすると、プログラムが削除されます。

ほとんどの場合、独自の PROGRAM 定義を作成する必要はありません。ただし、CICS に自動的に作成させたくない場合、あるいは特定の属性を指定しなければならない場合は、独自の PROGRAM 定義を作成できます。プラットフォームにデプロ

イされる CICS アプリケーションの一部として専用プログラムを作成するには、そのアプリケーションの一部としてインストールされる CICS バンドルに、その専用プログラムを定義する必要があります。CSD、BAS、または CICS バンドル内にプログラム定義を作成し、自分でインストールできます。CICS が @CICSProgram で注釈が付けられたメソッドを検出し、そのメソッドに一致する PROGRAM リソースがすでにインストールされている場合、CICS は既存のリソースを置き換えません。

プログラム定義を作成する際は、@CICSProgram で注釈が付けられたメソッドを含むクラスと同じクラス名を指定する必要があります。オプションで、メソッド名も指定できます。CICS はこの情報を、プログラムの呼び出し時に検証します。

JVMCLASS 属性には、クラス名およびオプションのメソッド名を

wlp:classname#methodname の形式で含める必要があります。以下に例を示します。

wlp:com.example.CustomerLinkTarget#getCustomer

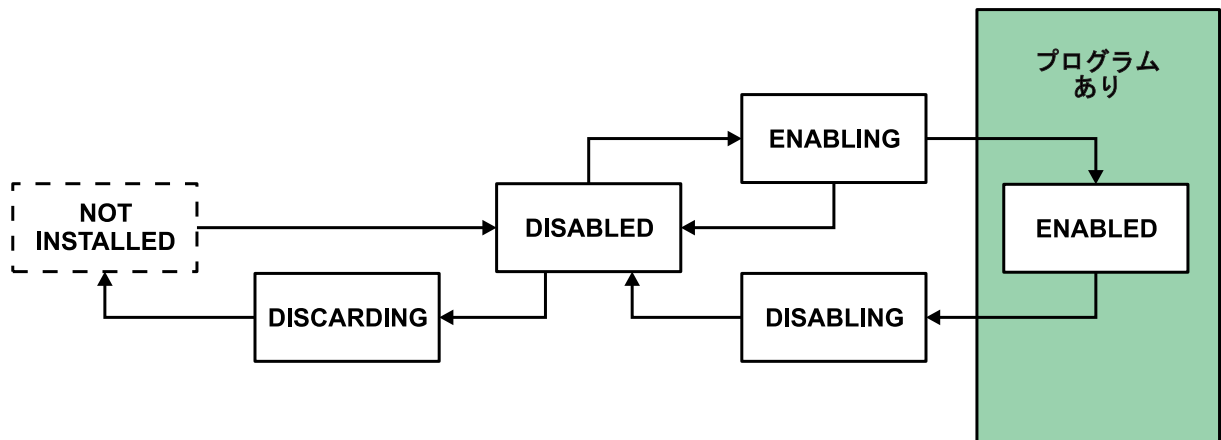


図 132. CICS バンドルのライフサイクル (@CICSProgram 注釈の PROGRAM リソースが存在する場合)

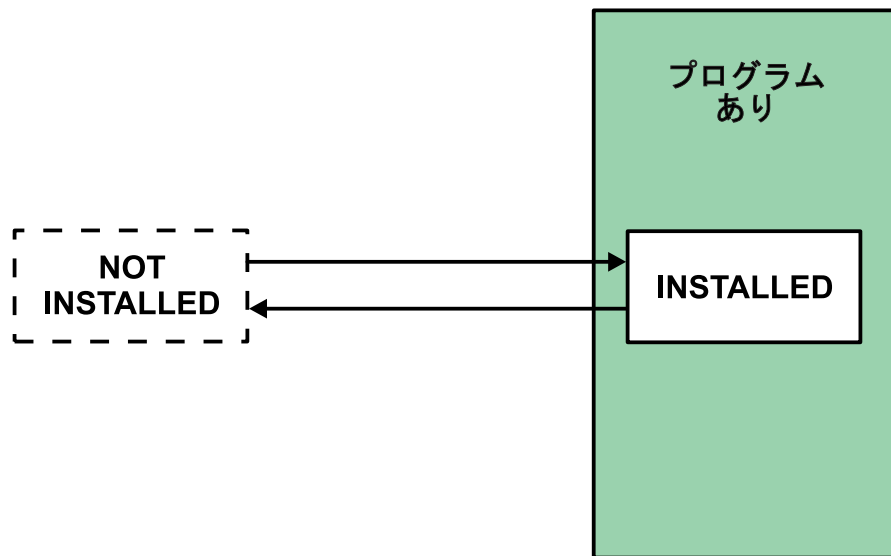


図 133. スタンドアロン Web アプリケーションのライフサイクル (@CICSProgram 注釈の PROGRAM リソースが存在する場合)

## Java Transaction API (JTA)

Java Transaction API (JTA) を使用すると、複数のリソース・マネージャーに対するトランザクション更新を調整することができます。

Java Transaction API (JTA) を使用すると、CICS リソースおよび他のサード・パーティー製リソース・マネージャー (Liberty JVM サーバー内のタイプ 4 データベース・ドライバ接続など) に対するトランザクション更新を調整できます。このシナリオでは Liberty トランザクション・マネージャーがトランザクション・コーディネーターです。CICS 作業単位は、CICS システム外部でトランザクションが開始したかのように従属しています。

注: JVM プロファイル・オプション

`com.ibm.cics.jvmserver.wlp.jta.integration=true` を設定して自動構成を使用する場合、または手動で `server.xml` を構成して `<cicsts_jta Integration="true"/>` エレメントを含める場合、CICS 作業単位は JTA トランザクションに参加せず、個別にコミットまたはロールバックされることになります。

CICS データ・ソースを使用したローカル Db2 データベースへのタイプ 2 ドライバ接続では、CICS Db2 接続を使用してアクセスします。JTA を使用して、他の CICS リソースに対する更新に合わせて調整する必要はありません。

JTA の中で、複数のリソース・マネージャーに対する更新をカプセル化して調整するための `UserTransaction` オブジェクトを作成します。以下のコード・フラグメントは、`UserTransaction` を作成して使用方法を示しています。

```
InitialContext ctx = new InitialContext();
UserTransaction tran = (UserTransaction)ctx.lookup("java:comp/UserTransaction");
```

```

DataSource ds = (DataSource)ctx.lookup("jdbc/SomeDB");
Connection con = ds.getConnection();

// Start the User Transaction
tran.begin();

// Perform updates to CICS resources via JCICS API and
// to database resources via JDBC/SQLJ APIs

if (allOk) {
 // Commit updates on both systems
 tran.commit();
} else {
 // Backout updates on both systems
 tran.rollback();
}

```

OSGi アプリケーションを使用している場合は、必ず次のエントリーを MANIFEST.MF に含めます。

```
Import-Package: javax.transaction;version="[1.1,2)"
```

開発環境によっては、この依存関係がデフォルトで強調表示されない場合もあります。明示的に調べて、最小バージョンの 1.1 が指定されていることを確認するようお勧めします。ランタイム環境自体に依存関係を解決させるようにしている場合、基礎となる JRE により、より低いバージョンのパッケージに解決されて、Liberty ランタイムとの競合が発生する可能性があります。

CICS 作業単位の場合とは異なり、begin() メソッドを使って明示的に UserTransaction を開始する必要があります。begin() を呼び出すと、CICS は UserTransaction の開始前に行われたすべての更新をコミットします。UserTransaction は commit() または rollback() のいずれかのメソッド呼び出しにより終了するか、Web アプリケーション終了時に Web コンテナによって終了されます。UserTransaction がアクティブ状態である間、プログラムは JCICS Task commit() メソッドや rollback() メソッドを呼び出すことができません。

JCICS の Task.commit() および Task.rollback() メソッドは JTA トランザクション・コンテキスト内では無効になります。どちらを試行した場合も、InvalidRequestException がスローされます。

Liberty のデフォルト動作では、未確定 JTA トランザクションのリカバリーを試みる前に最初の UserTransaction が作成されるのを待機します。しかし、CICS は、Liberty JVM サーバーの初期化が完了するとすぐにトランザクション・リカバリーを開始します。JVM サーバーが無効な状態でインストールされていると、JVM サーバーが有効になった時点でリカバリーが実行されます。

EJB を使用する場合は、EJB での JTA トランザクションの使用を参照してください。

## Java Persistence API (JPA)

開発者がアプリケーションで利用する、リレーショナル・データベース・エンティティのオブジェクト指向バージョンを作成するには、JPA を使用できます。

データ型、キー、およびテーブル間の関係を含め、データベース内のテーブルとそのコンテンツを記述するために使用できる注釈および XML 拡張は、JPA を使用して提供することができます。開発者は SQL を使用する代わりに、API を使用してデータベース操作を実行できます。

CICS は jpa-2.0 および jpa-2.1 をサポートしています。これらのバージョンの違いについては、Java Persistence API (JPA) フィーチャーの概要を参照してください。

- **Entity** オブジェクトは単純な Java クラスであり、具象クラスにも抽象クラスにもすることができます。それぞれがデータベース表内の行を表し、プロパティとフィールドを使用して状態を保守します。各フィールドはテーブル内の列にマップされ、その特定のフィールドに関する重要な情報が追加されます。例えば、1 次キー（つまり、ヌルにできないフィールド）を指定できます。

```
@Entity
@Table(name = "JPA")
public class Employee implements Serializable
{
 @Id
 @Column(name = "EMPNO")
 private Long EMPNO;

 @Column(name = "NAME", length = 8)
 private String NAME;

 private static final long serialVersionUID = 1L;

 public Employee()
 {
 super();
 }

 public Long getEMPNO()
 {
 return this.EMPNO;
 }

 public void setEMPNO(Long EMPNO)
 {
 this.EMPNO = EMPNO;
 }

 public String getNAME()
 {
 return this.NAME;
 }

 public void setName(String NAME)
 {
 this.NAME = NAME;
 }
}
```

- **EntityManagerFactory** は、パーシスタンス・ユニットの **EntityManager** を生成するために使用されます。**EntityManager** は、アプリケーションで使用されているアクティブな **entity** オブジェクトのコレクションを維持します。**EntityManager** クラスを使用して、クラスを初期化し、データ保全性を管理するためのトランザクションを作成できます。次に、**Entity** クラスの **get** メソッドと **set** メソッドを使用してデータと対話してから、**Entity** トランザクションを使用してデータをコミットします。

以下の例に、レコードを挿入する場合のサンプル・コードを記載します。

```
@WebServlet("/Create")
public class Create extends HttpServlet
{
 private static final long serialVersionUID = 1L;

 @PersistenceUnit(unitName = "com.ibm.cics.test.wlp.jpa.annotation.cics.datasource")
 EntityManagerFactory emf;

 InitialContext ctx;

 /**
 * @throws NamingException
 * @see HttpServlet#HttpServlet()
 */
 public Create() throws NamingException
 {
 super();
 ctx = new InitialContext();
 }

 /**
 * @see HttpServlet#doGet(HttpServletRequest request,
 * HttpServletResponse response)
 */
 protected void doGet(HttpServletRequest request, HttpServletResponse
 response) throws ServletException, IOException
 {
 // Get the servlet parms
 String id = request.getParameter("id");
 String name = request.getParameter("name");

 // Create a new employee object
 Employee newEmp = new Employee();
 newEmp.setEMPNO(Long.valueOf(id));
 newEmp.setNAME(name);

 // Get the entity manager factory
 EntityManager em = emf.createEntityManager();

 // Get a user transaction
 UserTransaction utx;

 try
 {
 // Start a user transaction and join the entity manager to it
 utx = (UserTransaction) ctx.lookup("java:comp/UserTransaction");
 utx.begin();
 em.joinTransaction();

 // Persist the new employee
 em.persist(newEmp);

 // End the transaction
 utx.commit();
 }
 catch (Exception e)
 {
 throw new ServletException(e);
 }

 response.getOutputStream().println("CREATE operation completed");
 }
}
```

- `@PersistenceUnit` は、`EntityManagerFactory` とこれに関連付けられたパーススタンス・ユニットへの依存関係を表します。パーススタンス・ユニットの名前は、`persistence.xml` ファイルで定義されます。次に、エンティティーとテーブルをデータベースに接続するために、バンドル内に `persistence.xml` ファイルを作成します。`persistence.xml` ファイルで、これらのエンティティーの接続先データベースを記述します。このファイルには、プロバイダーの名前、エンティティー自体、データベース接続 URL、およびドライバーなどの重要な情報を含めます。

以下の例には、サンプル `persistence.xml` が含まれています。

```

<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0"
 xmlns="http://java.sun.com/xml/ns/persistence"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
 http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">

 <persistence-unit name="com.ibm.cics.test.wlp.jpa.annotation.cics.datasource">
 <jta-data-source>jdbc/jpaDataSource</jta-data-source>

 <class>com.ibm.cics.test.wlp.jpa.annotation.cics.datasource.entities.Employee</class>

 <properties>
 <property name="openjpa.LockTimeout" value="30000" />
 <property name="openjpa.Log" value="none" />
 <property name="openjpa.jdbc.UpdateManager" value="operation-order" />
 </properties>
 </persistence-unit>
</persistence>

```

## Enterprise JavaBeans (EJB)

Enterprise JavaBeans (EJB) は、Java API であり、Java EE 仕様のサブセットです。 EJB にはアプリケーションのビジネス・ロジックが含まれており、CICS Liberty によって完全にサポートされています (Lite サブセットを含む)。

EJB のサポートを提供する Liberty のフィーチャーは次のとおりです。

表 61. サポートを提供する Liberty のフィーチャー

フィーチャー	サポート	Java EE バージョン
ejbLite-3.1	このフィーチャーは、EJB 仕様で定義されている EJB テクノロジーの Lite サブセットを有効にします。このサブセットには、EJB 3.x API に書き込まれるローカル・セッション Bean のサポートが含まれています。	Java EE 6
mdb-3.1	このフィーチャーは、EJB テクノロジーのメッセージ駆動型 Bean サブセットを有効にします。これは、ejbLite フィーチャーがセッション Bean のサポートを有効にするのに似ています。	Java EE 6
ejbLite-3.2	このフィーチャーは、EJB 仕様で定義されている EJB テクノロジーの Lite サブセットを有効にします。このサブセットには、EJB 3.x API、非永続 EJB タイマー、および非同期ローカル・インターフェース・メソッドに書き込まれるローカル・セッション Bean のサポートが含まれています。	Java EE 7

表 61. サポートを提供する Liberty のフィーチャー (続き)

フィーチャー	サポート	Java EE バージョン
mdb-3.2	このフィーチャーは、EJB テクノロジーのメッセージ駆動型 Bean サブセットを有効にします。これは、ejbLite フィーチャーがセッション Bean のサポートを有効にするのに似ています。	Java EE 7
ejbHome-3.2	EJB 2.x API のサポート (特に javax.ejb.EJBLocalHome インターフェースのサポート) を有効にします。 ejbRemote フィーチャーと組み合わせると、javax.ejb.EJBHome インターフェースもサポートされます。	Java EE 7
ejbRemote-3.2	リモート EJB インターフェースのサポートを有効にします。	Java EE 7
ejbPersistentTimers-3.2	永続 EJB タイマーのサポートを有効にします。	Java EE 7
ejb-3.2	EJB 3.2 の完全なサポートを有効にします。リモート EJB テクノロジーを含むすべての EJB 3.2 テクノロジーに対応します。	Java EE 7

## 手順

server.xml ファイル内でフィーチャーを有効にします。以下に例を示します。

```
<featureManager>
 <feature>ejb-3.2</feature>
</featureManager>
```

詳しくは、以下の情報を参照してください。

- **Developing EJB 3.x applications** 。WebSphere Developer Tools を使用して EJB アプリケーションを開発する方法について説明しています。
- **エンタープライズ Bean (EJB) パーシスタント・タイマー・アプリケーションの開発**。EJB 永続タイマー・アプリケーションを開発する方法について説明しています。
- **別のアプリケーションにあるローカル EJB コンポーネントを呼び出す Enterprise JavaBeans アプリケーションの使用**。別のアプリケーション内のローカル EJB コンポーネントを呼び出す Enterprise JavaBeans アプリケーションの使用方法について説明しています。

## Enterprise JavaBeans (EJB)プロジェクトの作成

Java アプリケーションの EJB を開発する場合は、EJB プロジェクトを作成できます。

### 始める前に

Eclipse IDE に Web 開発ツールがインストールされていることを確認します。詳しくは、751 ページの『開発環境のセットアップ』を参照してください。

### このタスクについて

CICS Explorer および IBM CICS SDK for Java のヘルプには、EJB アプリケーションを開発してパッケージ化する以下のステップの実行方法が詳細に説明されています。

### 手順

1. アプリケーション用に EJB プロジェクトを作成します。
2. EJB アプリケーションを開発します。JCICS API を使用して CICS サービスに、JDBC を使用して DB2 に、また、JMS を使用して IBM MQ にアクセスできます。
3. オプション: アプリケーションを保護するには、セキュリティー注釈を使用するか、ejb-jar.xml ファイル内にセキュリティー制約を指定できます。詳しくは、エンタープライズ・アプリケーション・セキュリティーを参照してください。
4. EJB プロジェクトをエンタープライズ・アプリケーション・プロジェクト (EAR) に追加します。

### タスクの結果

これで、開発環境のセットアップ、EJB プロジェクトの作成、およびデプロイメント用のパッケージ化が終了しました。

## EJB での JTA トランザクションの使用

Liberty の Enterprise JavaBeans (EJB) で JTA トランザクションを使用する方法。

### このタスクについて

EJB は、Liberty JVM サーバーによって管理される Java オブジェクトで、Java アプリケーションのモジュラー・アーキテクチャーを可能にします。Liberty JVM サーバーは、EJB Lite 3.1、EJB Lite 3.2、および EJB 3.2 をサポートしています。EJB は、エンタープライズ・アプリケーション・プロジェクトで作成されたエンタープライズ・アプリケーション・アーカイブ (EAR) ファイルを使用して Liberty サーバーにデプロイされます。エンタープライズ・アプリケーション・プロジェクトには、EJB プロジェクトと Web プロジェクトの両方を含めることができます。

EJB Lite を有効にするには、該当する ejbLite-3.1 または ejbLite-3.2 フィーチャーを server.xml 構成ファイルに追加します。EJB を有効にするには、ejb-3.2 を server.xml 構成ファイルに追加します。EJB はコンテナにデプロイされます。このコンテナはバックグラウンドで動作し、セッション管理、トランザクション、およびセキュリティーなどの面で一定の規準に準拠するようにします。

EJB は、コンテナ管理と Bean 管理の 2 種類のトランザクション管理をサポートします。コンテナ管理トランザクションは、Bean メソッドの呼び出しにトランザクション・コンテキストを提供し、Java アノテーションまたはデプロイメント記述子ファイル `ejb-jar.xml` を使用して定義されます。Bean 管理トランザクションは、Java Transaction API (JTA) を使用して直接制御されます。CICS JTA の統合を無効にしていなければ、いずれの場合も、CICS UOW は JTA トランザクションの結果に従属したままの状態になります。コンテナ管理トランザクションには、次の 6 つの異なるトランザクション属性を指定することができます。

- Mandatory
- Required
- RequiresNew
- Supports
- NotSupported
- Never

JTA トランザクションは、JEE 仕様に定義されている分散作業単位です。メソッドのトランザクション属性の設定によって、メソッドを実行する CICS タスクが独自の作業単位として実行されるのか、それともより広範囲の分散 JTA トランザクションの一部として実行されるのかが決まります。トランザクション属性および呼び出し側のアプリケーションが既に JTA トランザクション・コンテキストを持っているかどうかに応じて、呼び出された EJB メソッドのトランザクション・コンテキストがどのようになるかを、下の表に示します。

重要: Liberty では、アウトバウンドおよびインバウンドのトランザクションの伝搬はサポートされません。詳しくは、Liberty でのリモート・インターフェースによる Enterprise JavaBeans の使用を参照してください。

表 62. EJB トランザクション・サポート

トランザクション属性	JTA トランザクションが存在しない	JTA トランザクションが既に存在する	既存の JTA トランザクションでリモート EJB にアクセスする	例外の動作
Mandatory	例外 <code>EJBTransactionRequiredException</code> をスローする。	既存の JTA トランザクションを継承する。	例外 <code>com.ibm.websphere.csi.CSITransactionMandatoryException</code> をスローする。	Rollback
Required	EJB コンテナは新しい JTA トランザクションを作成する。	既存の JTA トランザクションを継承する。	例外 <code>com.ibm.websphere.csi.CSITransactionRequiredException</code> をスローする。	Rollback
RequiresNew	EJB コンテナは新しい JTA トランザクションを作成する。	例外 <code>javax.ejb.EJBException</code> をスローする。	EJB コンテナは、リモート・サーバーで管理される新しい JTA トランザクションを作成する。	Rollback
Supports	JTA トランザクションなしで続ける。	既存の JTA トランザクションを継承する。	例外 <code>com.ibm.websphere.csi.CSITransactionSupportedException</code> をスローする。	JTA から呼び出されるとロールバックする。
NotSupported	JTA トランザクションなしで続ける。	JTA トランザクションを停止するが、CICS UOW は停止しない。	リモート・サーバーは JTA トランザクションなしで続ける。	ロールバックなし
Never	JTA トランザクションなしで続ける。	例外 <code>javax.ejb.EJBException</code> をスローする。	リモート・サーバーは JTA トランザクションなしで続ける。	ロールバックなし

重要: 「NotSupported」のマークが付けられているメソッドを呼び出すと、JTA トランザクションは停止されますが、CICS UOW は停止されません。このメソッドの呼び出し中に行われた CICS リソースへの変更は、元に戻すことができます。

注: トランザクション属性 `RequiresNew` は、CICS Liberty JVM サーバーでサポートされていますが、CICS UOW をネストできないという制限があります。既に

JTA トランザクションが開始された状態で「RequiresNew」のマークが付けられているメソッドを呼び出そうとすると、例外がスローされます。

さらなる Enterprise JavaBeans (EJB) フィーチャーの制限については、『Liberty: ランタイム環境での既知の問題および制約事項』を参照してください。

## リモート・インターフェースを備えたエンタープライズ Java Bean (EJB) メソッド

リモート・インターフェースを備えた EJB メソッドには、RMI-IIOP テクノロジーを使用して CICS Liberty でリモートからアクセスしたり、ホストしたりできます。リモート EJB サポートは、ejbRemote-3.2 フィーチャーを使用して有効化できます。

リモート EJB インターフェースを使用する場合は、注意すべき考慮事項があります。詳しくは、Liberty でのリモート・インターフェースによる Enterprise JavaBeans の使用を参照してください。

### リモート・インターフェースを持つ EJB メソッドへのアクセス

1. リモート・インターフェースを使用して EJB メソッドにアクセスするアプリケーションを実行するように CICS Liberty を構成するには、以下のように、ejbRemote-3.2 フィーチャーを server.xml ファイルに追加して有効にする必要があります。

```
<featureManager>
 <feature>ejbRemote-3.2</feature>
</featureManager>
```

2. デプロイメント記述子 <ejb-ref> またはソース・コード・アノテーション (例えば、@EJB) で定義されたリモート EJB 参照に対して、アプリケーション・バイnding・ファイル (例えば、ibm-\*.bnd.xml) を構成します。アノテーションまたはデプロイメント記述子でルックアップ名を指定する EJB 参照にはバイndingは必要ありません。以下のように、バイnding・ファイル内で、EJB の java: 名のいずれかを使用するか、corbaname:names のいずれかを使用して、EJB 参照をバインドできます。

```
@EJB(name="TestBean")
TestRemoteInterface testBean;
```

バイndingは次のように定義されます:

```
<ejb-ref name="TestBean" binding-name=
"corbaname:rir:#ejb/global/TestApp/TestModule/TestBean!test.TestRemoteInterface"/>
```

3. スタブ・クラスを含むようにアプリケーション・クライアントを構成します。

### リモート・インターフェースを持つ EJB メソッドのホスト

1. 他の JVM によって EJB を呼び出すことができるように、CICS Liberty でこの EJB をホストするには、以下のように、ejbRemote-3.2 フィーチャーを server.xml ファイルに追加して有効にする必要があります。

```
<featureManager>
 <feature>ejbRemote-3.2</feature>
</featureManager>
```

2. ポートとセキュリティー設定をカスタマイズするように IIOP サーバーを構成します。詳しくは、リモート EJB の IIOP-RMI トランスポートの構成を参照してください。

3. EJB アプリケーションを作成します。詳しくは、Enterprise JavaBeans (EJB) プロジェクトの作成を参照してください。
4. スタブ・クラスを生成します。Eclipse で、EJB プロジェクトを右クリックし、「Java EE ツール」 > 「EJB クライアント Jar を作成します」を選択します。
5. EAR の一部として EJB アプリケーションを CICS Liberty にデプロイします。詳しくは、エンタープライズ・アプリケーション・プロジェクトの作成を参照してください。
6. Liberty messages.log ファイルを確認し、EJB が有効であること、および名前空間にバインドされていることを確認します。次のメッセージが表示されます。  

```
CNTR0167I: サーバーが、ejb.remote アプリケーションの ejb.remote.ejb.jar
モジュールにある MyBean エンタープライズ Bean の
ejb.remote.ejb.view.MyBeanRemote インターフェースを
バインディングしています。バインディング・ロケーションは、
java:global/ejb.remote/ejb.remote.ejb/MyBean!remote.ejb.view.MyBeanRemote です。
```

リモート EJB 用の IIOP-RMI トランスポートの構成:

CICS Liberty では、Internet Inter-ORB Protocol Remote Method Invocation (IIOP-RMI) トランスポートを使用して、リモート・インターフェースを備えた EJB メソッドと通信できます。この通信は、Common Secure Interoperability Protocol バージョン 2 (CSIv2) を使用して保護できます。

CICS Liberty では、リモート・インターフェースを備えた EJB メソッドを呼び出すためのテクノロジーとして、IIOP-RMI を使用します。ejbRemote-3.2 フィーチャーを使用すると、インバウンドとアウトバウンドの両方の IIOP-RMI 呼び出しがサポートされます。

インバウンドの呼び出しでは、CICS Liberty は、オブジェクト・リクエスト・ブローカー (ORB) として TCP/IP ポートで IIOP-RMI 要求を listen し、ターゲットの EJB メソッドを呼び出すことができます。詳細については、『インバウンド IIOP 通信の構成』を参照してください。

アウトバウンドの呼び出しとは、CICS Liberty が ORB に対して EJB メソッドを開始するように要求する呼び出しです。アウトバウンドの呼び出しは、呼び出しが行われた同じ JVM サーバーに対して行うことも、ORB として動作可能な他の Java 仮想マシン (JVM) に対して行うこともできます。詳細については、776 ページの『アウトバウンド IIOP 通信の構成』を参照してください。

この通信は、CORBA (Common Object Request Broker Architecture) の認証、委任、および特権を満たすテクノロジーである CSIv2 を使用して保護できます。CSIv2 では、トランスポート層セキュリティ (TLS) の使用もサポートされます。詳しくは、IIOP 通信を保護するための CSIv2 の構成を参照してください。

詳しくは、Common Secure Interoperability バージョン 2 (CSIv2)を参照してください。

インバウンド IIOP 通信の構成

server.xml ファイルに ejbRemote-3.2 フィーチャーを追加して、このフィーチャーを有効にします。

```
<featureManager>
 <feature>ejbRemote-3.2</feature>
</featureManager>
```

オプションで、server.xml ファイルに IIOP エンドポイントを構成できます。

```
<iiopEndpoint id="defaultIiopEndpoint" host="host.example.com" iiopPort="2809" />
```

重要: デフォルトでは、IIOP エンドポイントは localhost:2809 を listen します。デフォルトの ORB は、IIOP エンドポイント defaultIiopEndpoint を参照します。ORB にインバウンド・セキュリティを構成する方法について詳しくは、IIOP 通信を保護するための CSIv2 の構成を参照してください。

#### アウトバウンド IIOP 通信の構成

server.xml ファイルに ejbRemote-3.2 フィーチャーを追加して、このフィーチャーを有効にします。

```
<featureManager>
 <feature>ejbRemote-3.2</feature>
</featureManager>
```

オプションで、リモート・サーバーのネーム・サービスを使用して ORB を構成できます。

```
<orb id="default0rb" nameService="corbaname::host.example.com:2809" />
```

重要: デフォルトでは、ORB はローカルの IIOP エンドポイント defaultIiopEndpoint を参照します。ORB にアウトバウンド・セキュリティを構成する方法について詳しくは、IIOP 通信を保護するための CSIv2 の構成を参照してください。

#### IIOP 通信を保護するための CSIv2 の構成:

ここでは、IIOP 通信のためにインバウンドとアウトバウンドの両方の CSIv2 セキュリティを構成する一般的なケースについて説明します。

インバウンドの呼び出しでは、CICS Liberty は、オブジェクト・リクエスト・ブローカー (ORB) として TCP/IP ポートで IIOP-RMI 要求を listen し、ターゲットの EJB メソッドを呼び出すことができます。

アウトバウンドの呼び出しとは、CICS Liberty が ORB に対して EJB メソッドを開始するように要求する呼び出しです。アウトバウンドの呼び出しは、呼び出しが行われた同じ JVM サーバーに対して行うことも、ORB として動作可能な他の Java 仮想マシン (JVM) に対して行うこともできます。

次の例では、client がアウトバウンド要求を出す JVM であり、server がインバウンド要求を受け取る JVM です。これらのうちの片方または両方を CICS Liberty JVM サーバーにすることができます。詳しくは、Liberty での Common Secure Interoperability Version 2 (CSIv2) の構成を参照してください。

#### TLS を使用するように CSIv2 を構成する

##### インバウンド

- サーバーの証明書を格納する鍵ストアを作成します。

```
<keyStore id="iiopKeyStore" ... />
```

- 鍵ストアを参照する SSL レポートリー(SSL エlement) を作成します。

```
<ssl id="iiopSSL" keyStoreRef="iiopKeyStore" />
```

- IIOPS ポートを指定して IIOP エンドポイントを作成します。

```
<iiopEndpoint id="defaultIiopEndpoint" host="host.example.com" iiopPort="2809">
 <iioptions iiopPort="9402" sslRef="iiopSSL" />
</iiopEndpoint>
```

重要: デフォルトでは、IIOPs オプションの `sslRef` は SSL レポートリー `defaultSSLConfig` を参照します。

#### アウトバウンド

- 鍵ストアを作成します。鍵ストアにおけるルート証明書に対する信頼を可能にする鍵を格納できます。これにより、その証明書で署名されたすべての証明書が信頼されます。

```
<keystore id="iiopTrustStore" ... />
```

- 鍵ストアを参照する SSL レポートリー(SSL エlement) を作成します。

```
<ssl id="iiopSSL" trustStoreRef="iiopTrustStore" ... />
```

- CSIV2 クライアント・ポリシーを指定して ORB を作成します。

```
<orb id="defaultOrb" nameService="corbaname::host.example.com">
 <clientPolicy.csiv2>
 <layers>
 <transportLayer sslRef="iiopSSL" />
 </layers>
 </clientPolicy.csiv2>
</orb>
```

クライアントからサーバーにユーザー ID を伝搬できるように **CSIV2** を構成する

#### インバウンド

- CSIV2 サーバー・ポリシーを指定して ORB を作成します。

```
<orb id="defaultOrb">
 <serverPolicy.csiv2>
 <layers>
 <attributeLayer identityAssertionEnabled="true" />
 </layers>
 </serverPolicy.csiv2>
</orb>
```

- オプションで、サーバーで信頼される ID を 1 つ以上指定できます。

```
<attributeLayer identityAssertionEnabled="true" trustedIdentities="MYUSER" />
```

#### アウトバウンド

- CSIV2 クライアント・ポリシーを指定して ORB を作成します。

```
<orb id="defaultOrb" nameService="corbaname::host.example.com:2809">
 <clientPolicy.csiv2>
 <layers>
 <attributeLayer identityAssertionEnabled="true" />
 </layers>
 </clientPolicy.csiv2>
</orb>
```

- オプションで、サーバーで許可する信頼できる ID を指定できます。

```
<attributeLayer identityAssertionEnabled="true" trustedIdentity="MYUSER"
 trustedPassword="MYPASSWD" />
```

重要: サーバーのユーザー・レジストリー内に、信頼できるユーザーが存在していなければなりません。trustedPassword は、Liberty securityUtility ツールを使用してエンコードできます。

**TLS** クライアント認証を使用するように **CSIv2** を構成する

インバウンド

- 鍵ストアを作成します。鍵ストアにおけるルート証明書に対する信頼を可能にする鍵を格納できます。これにより、その証明書で署名されたすべての証明書が信頼されます。

```
<keyStore id="iiopTrustStore" ... />
```

- 鍵ストアを参照する SSL レポートリー(SSL エlement) を作成します。

```
<ssl id="iiopSSL" trustStoreRef="iiopTrustStore" ... />
```

- IIOPS エンドポイントを指定して IIOP エンドポイントを作成します。

```
<iiopEndpoint id="defaultIiopEndpoint" host="host.example.com" port="2809">
 <iiopsOptions iiopsPort="9402" sslRef="iiopSSL" />
</iiopEndpoint>
```

- CSIv2 サーバー・ポリシーを指定して ORB を作成します。

```
<orb id="defaultOrb">
 <serverPolicy.csiv2>
 <layers>
 <attributeLayer identityAssertionEnabled="true" ... />
 <transportLayer sslRef="iiopSSL" />
 </layers>
 </serverPolicy.csiv2>
</orb>
```

アウトバウンド

- クライアント証明書を格納する鍵ストアを作成します。

```
<keyStore id="iiopKeyStore" ... />
```

- 鍵ストアを参照する SSL レポートリー(SSL エlement) を作成します。

```
<ssl id="iiopSSL" keyStoreRef="iiopKeyStore" />
```

- CSIv2 クライアント・ポリシーを指定して ORB を作成します。

```
<orb id="defaultOrb">
 <clientPolicy.csiv2>
 <layers>
 <attributeLayer identityAssertionEnabled="true" />
 <transportLayer sslRef="iiopSSL" />
 </layers>
 </clientPolicy.csiv2>
</orb>
```

## Java Message Service (JMS)

Java Message Service (JMS) は、Java EE をベースにしたアプリケーション・コンポーネントでメッセージの作成、送信、受信、および読み取りを可能にする API です。Liberty での JMS サポートは、JMS リソース・アダプターのデプロイメントをサポートする一群の関連フィーチャーとして提供されます。

JMS は、キュー、トピック、接続、および他のリソースがサーバー構成によって作成および管理される管理モードで実行できます。これには、JMS 接続ファクトリー、キュー、トピック、およびアクティベーション仕様の構成が含まれます。代わりに、すべてのリソースをアプリケーションの一部として手動で構成する非管理モードで実行することもできます。Liberty 組み込み JMS メッセージング・プロバイダーは管理対象であるため、すべてのリソースは `server.xml` 構成の一部としてセットアップされます。

## JMS 仕様

Liberty JVM サーバーでサポートされる JMS 仕様レベルは、JMS 2.0 サポートです。JMS 2.0 サポート (`jms-2.0`) では、2.0 仕様レベルでの Java Message Service API を使用してメッセージング・システムにアクセスするリソース・アダプターを構成できます。

## JMS クライアント

以下の Liberty 機能によって、Liberty JVM サーバーでは各種 JMS クライアント・プロバイダーがサポートされています。

- WebSphere MQ JMS 2.0 クライアント (`wmqJmsClient-2.0`)。この WebSphere MQ JMS クライアント機能によって、JMS 2.0 または 1.1 クライアント・アプリケーションは、リモート MQ サーバーとの間でメッセージを送受信できるようになります。
- WebSphere Application Server JMS 2.0 クライアント (`wasJmsClient-2.0`)。この WebSphere Application Server クライアント機能によって、JMS 2.0 または 1.1 クライアント・アプリケーションは、`wasJmsServer` フィーチャーによって有効にされたメッセージング・エンジンとの間でメッセージを送受信できるようになります。
- JCA 1.6 仕様に準拠するそれ以外の JMS リソース・アダプターも、汎用 JCA リソース・アダプター・リンクを使用することによって Liberty で使用できます (JCA 構成エレメントの概要を参照)。

## JMS プロバイダー

CICS TS 内の Liberty は、以下の使用をサポートします。

- Liberty 組み込み JMS メッセージング・プロバイダー。
  - WebSphere メッセージング・サーバー (`wasJmsServer-1.0`)。JMS サーバー機能によって、組み込み JMS メッセージング・プロバイダーを Liberty 内でホストできるようになります。これにより、JMS サーバーを別個にインストールして構成する必要がなくなります (単一 Liberty サーバーでの JMS メッセージングの有効化を参照)。このサーバーは、CICS 内部のまたは z/OS や他の分散プラットフォームでホストされる Liberty サーバーの、別個の Liberty インスタンスでホストすることもできます (2 つの Liberty サーバー間の JMS メッセージングの有効化を参照)。WebSphere JMS メッセージング・クライアント・コンポーネントを、WebSphere Application Server で実行される SIBUS 経由で JMS と対話するように構成することもできます (Liberty と WebSphere Application Server traditional との間のインターオペラビリティの使用可能化を参照)。

- WebSphere メッセージング・セキュリティ (wasJmsSecurity-1.0)。この JMS セキュリティ機能は、組み込み JMS メッセージング・プロバイダー・クライアントおよびサーバー・コンポーネントにセキュリティ・サポートを提供します。この JMS セキュリティ機能を `cicsts:security-1.0` フィーチャーと併用することで、組み込み JMS メッセージング・サーバーに対して要求を認証する際に、セキュリティ・レジストリーからどのユーザーを選択して接続ファクトリーで使用するかを指定できます。許可について詳しくは、メッセージング・エンジンに接続するユーザーの許可を参照してください。
- JMS アプリケーションがバインディング・モードまたはクライアント・モードのいずれかの転送を使用して接続する場合、CICS 標準モードの Liberty JVM サーバー内での IBM MQ への JMS アクセス。
- JMS アプリケーションがクライアント・モードの転送を使用して接続する場合、CICS 統合モードの Liberty JVM サーバー内での IBM MQ への JMS アクセス。
- JCA 1.6 仕様に準拠したサード・パーティー JMS リソース・アダプター。

## Java Management Extensions API (JMX)

Java Management Extensions API (JMX) は、リソースのモニターと管理に使用します。

JMX は、広く受け入れられている実装を使用してアプリケーションの情報を公開する手段を提供する、Java フレームワークおよび API です。公開された情報を読み取るように、JConsole などの各種のツールを構成できます。情報を公開するための手段としては、管理 Bean (MBean) が使用されます。MBean は、`public` コンストラクターを備えた非静的 Java クラスです。Bean の `get` および `set` メソッドは属性として公開されますが、それ以外のすべてのメソッドは操作として公開されます。

ローカルまたはリモート・マシンから Liberty JVM サーバーの JMX に接続して、これらの MBean の属性と操作を表示できます。ローカルに接続するには、`server.xml` に `localConnector-1.0` 機能を追加して、同じ JVM サーバー内から接続できるようにする必要があります。`restConnector-1.0` フィーチャーを `server.xml` に追加すると、RESTful インターフェースを接続手段として使用できるようになるため、JMX へのリモート・アクセスが可能になります。

### WebSphere MBean を使用したアプリケーションのモニター

1. まず、MBeanServer の参照を取得する必要があります。この例では、**JvmStats** MBean を検索し、**findMBeanServer** メソッドを使用して、この MBean の登録先サーバーを確認します。正しい MBeanServer オブジェクトの参照から、MBean の参照を取得し、その MBean が公開する属性からデータを取得できます。この例では、**JvmStats** MBean の **UpTime** 属性を検索します。

```
// Create an ObjectName object for the MBean that we're looking for.
ObjectName beanObjName = null;
beanObjName = new ObjectName("WebSphere:type=JvmStats");

// Obtain the full list of MBeanServers.
java.util.List servers = MBeanServerFactory.findMBeanServer(null);
MBeanServer mbs = null;
```

```
// Iterate through our list of MBeanServers and attempt to find the one we want.
for (int i = 0; i < servers.size(); i++)
{
 // Check if the MBean domain matches what we're looking for.
 mbs = (MBeanServer)servers.get(i);

 if (mbs.isRegistered(beanObjName))
 {
 Object attributeObj = mbs.getAttribute(beanObjName, "UpTime");
 System.out.println("UpTime of JVM is: " + attributeObj + ".");
 }
}
```

## Liberty での JMX へのリモート接続

Liberty JVM サーバーで JMX にリモート接続するには、SSL 接続と Java Platform, Enterprise Edition (JEE) ロール許可を使用する必要があります。それによって、クライアント・コードは JMXServiceURL を使用してリモート MBean の参照を取得します。

1. REST コネクタでアクセスされるすべての JMX MBeans は、単一の JEE ロール「管理者」によって保護されています。このロールへのアクセスを提供するには、server.xml を編集して、管理者ロールに認証済みユーザーを追加します。

```
<administrator-role>
<user>myuserid</user>
<group>group1</group>
</administrator-role>
```

JEE ロールの使用について詳しくは、SAF ロール・マッピングを使用した許可を参照してください。

2. リモートの RESTful JMX クライアントは SSL を使用して、Liberty JVM サーバーにアクセスする必要があります。Liberty JVM サーバー用の SSL サポートを構成する方法については、RACF を使用した Liberty JVM サーバー用の SSL (TLS) の構成のトピックを参照してください。さらに、JMX クライアントは、restConnector クライアント・サイド JAR ファイルと、サーバーの署名証明書が含まれる SSL クライアント鍵ストアにもアクセスする必要があります。CICS WLP インストールの一部として提供される restConnector.jar は、&USSHOME;/wlp/clients で使用可能になっています。
3. クライアント・サイドのコードで、JMXServiceURL オブジェクトを作成する必要があります。これによって、リモート MBeanServerConnection オブジェクトの参照を取得できます。以下の例を見ると、<host> と <httpsPort> が、サーバーのものと一致しています。

```
JMXServiceURL url = new JMXServiceURL("service:jmx:rest://<host>:<httpsPort>/IBMJMXConnectorREST");
JMXConnector jmxConnector = JMXConnectorFactory.connect(url, environment);
MBeanServerConnection mbsc = jmxConnector.getMBeanServerConnection();
```

4. 接続の取得に成功すると、MBeanServerConnection オブジェクトは MBeanServer オブジェクトからローカル接続する場合と同じ機能およびメソッド一式を提供します。

WebSphere で提供される MBeans について詳しくは、Liberty プロファイル: 提供されている MBean のリストを参照してください。

## Java Authorization Contract for Containers (JACC)

Liberty は、デフォルトの許可に加えて、Java Authorization Contract for Containers (JACC) 仕様を基にした許可をサポートしています。Liberty でセキュリティが有効になっていると、JACC プロバイダーが指定されない限り、デフォルトの許可が使用されます。

### このタスクについて

JACC により、アプリケーション・サーバーでの許可の管理に、サード・パーティーのセキュリティ・プロバイダーを使用できます。デフォルトの許可では特別な設定は必要なく、デフォルトの許可エンジンがすべての許可の決定を行います。ただし、JACC プロバイダーが Liberty で使用されるように構成され、設定された場合、すべてのエンタープライズ Bean および Web の許可の決定は、JACC プロバイダーが代行します。JACC は、アプリケーション・サーバーと許可ポリシー・モジュールとの間のセキュリティ契約を定義します。これらの契約で、許可プロバイダーのインストール方法、構成方法、およびアクセス判断での使用方法が指定されます。jacc-1.5 フィーチャーを Liberty サーバーに追加するには、Liberty には付属していない、サード・パーティーの JACC プロバイダーを追加します。

Liberty サーバーで提供されている

`com.ibm.wsspi.security.authorization.jacc.ProviderService` インターフェースを実装することによって、Java EE アプリケーションのカスタム許可決定を行う JACC プロバイダーを開発できます。JACC 仕様である JSR 115 は、許可プロバイダー用のインターフェースを定義しています。Liberty サーバーで、JACC プロバイダーをユーザー・フィーチャーとしてパッケージする必要があります。そのフィーチャーは `com.ibm.wsspi.security.authorization.jacc.ProviderService` インターフェースを実装する必要があります。

### 手順

1. OSGi バンドル・プロジェクトを作成して Java クラスを開発します。プロジェクトにコンパイル・エラーがある可能性があります。これらのエラーを修正するには、`javax.security.jacc` および `com.ibm.wsspi.security.authorization.jacc` の 2 つのパッケージをインポートする必要があります。

ファイル MANIFEST.MF を編集して、欠落しているパッケージをインポートします。

```
Manifest-Version: 1.0
Service-Component: OSGI-INF/myjaccExampleComponent.xml,
Bundle-ManifestVersion: 2
Bundle-Name: com.example.myjaac.osgiBundle
Bundle-SymbolicName: com.example.myjaac.osgiBundle
Bundle-Version: 1.0.0
Bundle-RequiredExecutionEnvironment: JavaSE-1.7
Import-Package: com.ibm.wsspi.security.authorization.jacc;version="1.0.0",
javax.security.jacc;version="1.5.0"
```

サービス・コンポーネント XML `myjaccExampleComponent.xml` の例は次のようになります。

```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0" immediate="true"
 name="TestPolicyServiceProvider">
```

```

<implementation class="com.example.myjaac.osgiBundle.TestPolicyServiceProvider"/>
<property name="javax.security.jacc.policy.provider" type="String" value=""/>
<property name="javax.security.jacc.PolicyConfigurationFactory.provider" type="String" value=""/>
<service>
 <provide interface="com.ibm.wsspi.security.authorization.jacc.ProviderService"/>
</service>
</scr:component>

```

- Liberty フィーチャー・プロジェクトを作成し、以前の OSGi バンドルをユーザーの Liberty フィーチャー (フィーチャー・マニフェスト・ファイルの Subsystem-Content の下) に追加します。
- フィーチャー・マニフェストを変更して、必要な OSGi サブシステム・コンテンツ `com.ibm.ws.javaee.jacc.1.5; version="[1,1.0.200)"; location:="dev/api/spec/"` を追加します。

```

Subsystem-ManifestVersion: 1.0
IBM-Feature-Version: 2
IBM-ShortName: jacc15CICSLiberty-1.0
Subsystem-SymbolicName: com.example.myjaac.libertyFeature;visibility:=public
Subsystem-Version: 1.0.0
Subsystem-Type: osgi.subsystem.feature
Subsystem-Content: com.example.myjaac.osgiBundle;version="1.0.0",
 com.ibm.ws.javaee.jacc.1.5;version="[1,1.0.200)";location:="dev/api/spec/"
Manifest-Version: 1.0

```

さらにもう 1 つのサブシステム・コンテンツを追加する必要がある場合は、コンテンツを入力する前に 1 つ以上のスペースを追加する必要があります。スペースを追加しなかった場合、CICS は `java.lang.IllegalArgumentException` を返します。

- Liberty フィーチャー・プロジェクトを Liberty フィーチャー (ESA) ファイルとしてエクスポートします。
- ESA ファイルを zFS に FTP でファイル転送します。
- `installUtility` コマンドを使用して、ESA ファイルをインストールします。  
`./wlpenv installUtility install myFeature.esa`
- `jacc-1.5` フィーチャー、および JACC プロバイダーが含まれた ESA ファイルをユーザー・フィーチャーとして `server.xml` に追加します。

```

<feature>jacc-1.5</feature>
<feature>usr:jacc15CICSLiberty-1.0</feature>

```

## Java Authentication Service Provider Interface for Containers (JASPIC)

Java Authentication Service Provider Interface for Containers (JASPIC) 仕様では、サービス・プロバイダー・インターフェース (SPI) を定義しています。メッセージ認証メカニズムを実装する認証プロバイダーを、クライアントまたはサーバーのメッセージ処理コンテナまたはランタイムに統合できます。

### このタスクについて

JASPIC インターフェースを介して統合された認証プロバイダーは、呼び出し側コンテナによって提供されたネットワーク・メッセージを操作します。プロバイダーがメッセージを変換することにより、受信側コンテナによるメッセージ送信元の

認証と、メッセージ送信側によるメッセージ受信側の認証が可能になります。着信メッセージが認証されて、呼び出し側コンテナに返されると、メッセージ認証の結果として ID が確立されます。

JSR 196 では標準 SPI を定義し、認証モジュールを Java EE コンテナに統合する方法を標準化しています。メッセージ処理モデル、およびクライアントとサーバー上の多数の対話ポイントの詳細が提供されます。互換性のある Web コンテナは、これらのポイントで SPI を使用して、対応するメッセージ・セキュリティ処理をサーバー認証モジュール (SAM) に委任します。

Liberty は、jaspic-1.1 に指定されたサーブレット・コンテナに対応するサード・パーティー認証プロバイダーの使用をサポートします。サーブレット・コンテナにはインターフェースが定義されており、セキュリティ・ランタイム環境は Web コンテナと連携して、それらのインターフェースを使用します。それらのインターフェースによって、アプリケーションが Web 要求を処理する前後に認証モジュールが始動されます。JASPIC モジュールを使用する認証は、セキュリティ構成で JASPIC が有効にされている場合のみ使用されます。

## 手順

1. OSGi バンドル・プロジェクトを作成して Java クラスを開発します。プロジェクトにコンパイル・エラーがある可能性があります。これらのエラーを修正するには、`javax.security.auth.message` および `com.ibm.wsspi.security.jaspi` の 2 つのパッケージをインポートする必要があります。ターゲット・プラットフォームを編集して、欠落している JAR を、`com.ibm.ws.security.jaspic` リスト (`<cics_install>/wlp/lib` ディレクトリー内) および `com.ibm.ws.javaee.jaspic.<version_number>` リスト (`<cics_install>/wlp/dev/api/spec` ディレクトリー内) に追加する必要があります。これらを FTP で開発システムに転送し、ビルド・パスに追加します。

ファイル MANIFEST.MF を編集して、欠落しているパッケージをインポートします。

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: com.example.myjaspic.osgiBundle
Bundle-SymbolicName: com.example.myjaspic.osgiBundle
Bundle-Version: 1.0.0
Bundle-RequiredExecutionEnvironment: JavaSE-1.7
Import-Package: com.ibm.wsspi.security.jaspi;version="1.0.13",
javax.security.auth.message;version="1.0.0",
javax.security.auth.message.callback;version="1.0.0",
javax.security.auth.message.config;version="1.0.0",
javax.security.auth.message.module;version="1.0.0",
javax.servlet;version="2.7.0",
javax.servlet.http;version="2.7.0"
Service-Component: myjaspicExampleComponent.xml
```

サービス・コンポーネント XML `myjaspicExampleComponent.xml` の例は次のようになります。

```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0" name="com.example.myjaspic.osgiBundle">
 <implementation class="com.example.myjaspic.osgiBundle.TestJASPICProviderService"/>
</scr:component>
```

```

<service>
 <provide interface="com.ibm.wsspi.security.jaspi.ProviderService"/>
</service>
</scr:component>

```

- Liberty フィーチャー・プロジェクトを作成し、以前の OSGi バンドルをユーザーの Liberty フィーチャー (フィーチャー・マニフェスト・ファイルの Subsystem-Content の下) に追加します。
- フィーチャー・マニフェストを編集して、必要な OSGi サブシステム・コンテンツ `com.ibm.websphere.appserver.jaspic-1.1`; `type="osgi.subsystem.feature"` を追加します。

```

Subsystem-ManifestVersion: 1.0
IBM-Feature-Version: 2
IBM-ShortName: jaspic11CICSLiberty-1.0
Subsystem-SymbolicName: com.example.myjaspic.libertyFeature;visibility:=public
Subsystem-Version: 1.0.0.201611081617
Subsystem-Type: osgi.subsystem.feature
Subsystem-Content: com.example.myjaspic.osgiBundle;version="1.0.0",
 com.ibm.websphere.appserver.jaspic-1.1;type="osgi.subsystem.feature",
 com.ibm.websphere.appserver.servlet-3.0;ibm.tolerates:="3.1";type="osgi.subsystem.feature"
Manifest-Version: 1.0

```

さらにもう 1 つのサブシステム・コンテンツを追加する必要がある場合は、コンテンツを入力する前に 1 つ以上のスペースを追加する必要があります。スペースを追加しなかった場合、CICS は `java.lang.IllegalArgumentException` を返します。

- Liberty フィーチャー・プロジェクトを Liberty フィーチャー (ESA) ファイルとしてエクスポートします。
- ESA ファイルを zFS に FTP でファイル転送します。
- `installUtility` を使用して、ESA ファイルをインストールします。  
`./wlpenv installUtility install myFeature.esa`
- `jaspic-1.1` フィーチャー、および JASPIC プロバイダーが含まれた ESA ファイルをユーザー・フィーチャーとして `server.xml` に追加します。  

```

<feature>jaspic-1.1</feature>
<feature>usr:jaspic11CICSLiberty-1.0</feature>

```

## Java EE コネクター・アーキテクチャー (JCA)

JCA は、CICS などのエンタープライズ情報システムを JEE プラットフォームに接続します。

JCA は、JEE アプリケーション・サーバーによって提供されるセキュリティー資格情報管理、接続プーリング、およびトランザクション管理のサービスの品質をサポートします。JCA を使用すると、これらのサービスの品質がアプリケーションではなく JEE アプリケーション・サーバーによって管理されるようになります。このことは、プログラマーがビジネス・コードの作成に専念でき、サービスの品質を意識する必要がないことを意味します。サービス品質の提供および構成のガイダンスについては、ご使用の JEE アプリケーション・サーバーの資料を参照してください。JCA は、共通クライアント・インターフェース (CCI) と呼ばれるプログラミング・インターフェースを定義します。このインターフェースにわずかな変更を加えるだけで、どのエンタープライズ情報システムとも通信できます。

## プログラミング・インターフェース・モデル

CCI を使用するアプリケーションは、あらゆるエンタープライズ情報システムのための共通の構造を持ちます。JCA は、CICS などのエンタープライズ情報システム (EIS) を JEE プラットフォームに接続します。これらの接続オブジェクトによって、JEE アプリケーション・サーバーは、リソース・アダプターのセキュリティ、トランザクション・コンテキスト、および接続プールを管理することができます。アプリケーションを開始するには、まず接続ファクトリーにアクセスする必要があります。そこから接続を取得することができます。この接続のプロパティは、ConnectionSpec オブジェクトによりオーバーライドすることができます。接続が取得された後で、特定の要求を出すために接続から対話を作成することができます。接続の場合と同様に、対話も InteractionSpec クラスによって設定されるカスタム・プロパティを持つことができます。対話を行うためには、execute() メソッドを呼び出し、record オブジェクトを使用してデータを保持します。以下に例を示します。

```
ConnectionFactory cf = <Lookup from JNDI namespace>
Connection c = cf.getConnection(ConnectionSpec);
Interaction i = c.createInteraction();
InteractionSpec is = newInteractionSpec();
i.execute(spec, input, output);
i.close();
c.close();
```

この例は、次のシーケンスを示しています。

1. ConnectionFactory オブジェクトを使用して、接続オブジェクトを作成する。
2. 接続オブジェクトを使用して、対話オブジェクトを作成する。
3. Interaction オブジェクトを使用して、エンタープライズ情報システムに対するコマンドを実行する。
4. 対話と接続を閉じる。

JEE アプリケーション・サーバーを使用する場合、接続ファクトリーは、サーバーの管理インターフェースを使用して構成することで作成します。Liberty サーバーでは、これは server.xml 構成で定義します。接続ファクトリーを作成したら、エンタープライズ・アプリケーションは、JNDI (Java Naming Directory Interface) でそれを検索して、アクセスできます。このタイプの環境は管理対象環境と呼ばれ、JEE アプリケーション・サーバーが接続のサービス品質を管理することを可能にします。管理対象環境について詳しくは、JEE アプリケーション・サーバーの資料を参照してください。

## レコード・オブジェクト

レコード・オブジェクトは、EIS との間で受け渡しされるデータを表わすために使用されます。これらのレコードを生成するには、アプリケーション開発ツールを使用することをお勧めします。Rational® Application Developer に備わっている J2C ツールを使用すると、COBOL コピーブックなどの特定のネイティブ言語構造から Record インターフェースの実装を作成でき、Java と Java 以外のデータ型の間のデータ・マーシャルも標準でサポートされます。

## リソース・アダプターのサンプル

リソース・アダプターの基本的なサンプルをインストールし、そのリソース・アダプターが提供するリソースのインスタンスを構成することができます。基本的な JCA リソース・アダプターの構成およびデプロイを参照してください。

### Common Client Interface

CCI が提供する標準インターフェースでは、開発者が汎用プログラミング・スタイルを使用して、各 EIS に対応するリソース・アダプターを介して任意の数の EIS と通信できます。CCI は、Java Database Connectivity (JDBC) で使用されるクライアント・インターフェースをモデルとして非常によく似た形に設計されており、Connection (接続) と Interaction (対話) についての考え方は JDBC と同様です。

### JCA ローカル ECI リソース・アダプターの使用

CICS TS で提供される JCA ローカル ECI リソース・アダプターは、ローカル CICS プログラムを呼び出します。これは、CICS Transaction Gateway ECI リソース・アダプターを使用するアプリケーションを CICS Liberty にマイグレーションするために最適化されたパスです。このセクションは、統合モードの Liberty にのみ適用されます。

JCA ローカル ECI リソース・アダプターを使用することで、CICS プログラムに接続し、COMMAREA またはチャンネルとコンテナのいずれかでデータを渡します。リソース・アダプターは、CICS Liberty フィーチャーで提供されます。

注: JCA ローカル ECI リソース・アダプターと CICS Transaction Gateway ECI リソース・アダプターを同じ Liberty JVM サーバー内で使用することはできません。

表 1 に、CICS 用語に対応する JCA オブジェクトを示します。

表 63. CICS 用語と対応する JCA オブジェクト

CICS 用語	JCA オブジェクト: プロパティ
異常終了コード	CICSTxnAbendException
COMMAREA	Record
Channel	ECIChannelRecord
データ・タイプ BIT のコンテナ	byte[]
データ・タイプ CHAR のコンテナ	String
プログラム名	ECIInteractionSpec:FunctionName
トランザクション	ECIInteractionSpec:TPNName

詳細については、839 ページの『JCA ローカル ECI サポート』を参照してください。

### JCA ローカル ECI リソース・アダプターの構成:

JCA 仕様の定義にしたがって、接続ファクトリーを使用して JCA ローカル ECI リソース・アダプターを構成できます。

JCA ローカル ECI の使用を開始するには、server.xml の featureManager エレメントにフィーチャー cicsts:jcaLocalEci-1.0 を追加します。

```
<featureManager>
<feature>cicsts:jcaLocalEci-1.0</feature>
</featureManager>
```

JCA ローカル ECI は、JNDI 名 **eis/defaultCICSConnectionFactory** にバインドされたデフォルトの接続ファクトリー **defaultCICSConnectionFactory** を提供します。オプションで、異なる JNDI 名が必要な場合は、次のようなプロパティー・サブエレメントを使用して追加の接続ファクトリーを構成することもできます。

```
<connectionFactory id="localEci" jndiName="eis/ECI">
<properties.com.ibm.cics.wlp.jca.local.eci/>
</connectionFactory>
```

ヒント: プロパティー・エレメントには属性は必要ありません。

**JCA ECI アプリケーションを Liberty JVM サーバーに移植する:**

JCA ローカル ECI リソース・アダプター・サポートを使用することで、JCA アプリケーションを簡単に Liberty JVM サーバーに移植できます。

#### 移植

CICS Transaction Gateway ECI リソース・アダプターを使用する既存の JCA アプリケーションを、スタンドアロン JEE アプリケーション・サーバーから CICS Liberty JVM サーバーに移植するには、以下の手順を使用します。

1. cicsts:jcaLocalEci-1.0 フィーチャーおよび webProfile-6.0 フィーチャーを server.xml ファイルに追加します。

以下に例を示します。

```
<featureManager>
...
<feature>cicsts:jcaLocalEci-1.0</feature>
<feature>webProfile-6.0</feature>
...
</featureManager>
```

2. リソースを更新して接続ファクトリーの JNDI 名を **eis/defaultCICSConnectionFactory** に変更するか、**connectionFactory** および **properties.com.ibm.cics.wlp.jca.local.eci** を server.xml に追加するかのいずれかの方法を使用できます。
3. アプリケーションを CICS にデプロイします (CICS バンドル内の Java EE アプリケーションの Liberty JVM サーバーへのデプロイを参照)。

ECI リソース・アダプターの制限されたフィーチャーを使用するアプリケーションの場合は、アプリケーションのコードを変更してそれらのサポートされていないフィーチャーを削除する必要があります。詳しくは、JCA ローカル ECI リソース・アダプターの制限を参照してください。

ローカル **ECI** リソース・アダプターを使用した **CICS** 内のプログラムへのリンク:

JCA ローカル ECI リソース・アダプターを使用して CICS 内のプログラムを実行するには、ECIInteraction クラスの execute() メソッドを使用します。

このタスクについて

このタスクでは、JCA ローカル ECI リソース・アダプターを使用して、CICS プログラムを COMMAREA に渡し、JCA レコードを使用して実行する方法をアプリケーション開発者に紹介します。Record インターフェースを拡張して CICS COMMAREA を表現する方法について詳しくは、を参照してください。チャンネルとコンテナを使用する CICS プログラムにリンクする方法について詳しくは、JCA ローカル ECI リソース・アダプターをチャンネルおよびコンテナと共に使用するを参照してください。

手順

1. JNDI を使用して、eis/defaultCICSConnectionFactory という名前の ConnectionFactory オブジェクトを検索します。
2. ConnectionFactory から Connection オブジェクトを取得します。
3. Connection から Interaction オブジェクトを取得します。
4. ECIInteractionSpec オブジェクトを新規作成します。
5. ECIInteractionSpec で set メソッドを使用して、プログラム名や COMMAREA の長さなど、実行に関わるプロパティを設定します。
6. 入力データを入れるレコード・オブジェクトを作成して (COMMAREA/チャンネルのトピックを参照)、データを取り込みます。
7. 出力データを入れるレコード・オブジェクトを作成します。
8. Interaction で execute メソッドを呼び出して、ECIInteractionSpec と 2 つの Record オブジェクトを渡します。
9. 出力したレコードからデータを読み取ります。

```
package com.ibm.cics.server.examples.wlp;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

import javax.annotation.Resource;
import javax.resource.cci.Connection;
import javax.resource.cci.ConnectionFactory;
import javax.resource.cci.Interaction;
import javax.resource.cci.Record;
import javax.resource.cci.Streamable;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.ibm.connector2.cics.ECIInteractionSpec;

/**
 * Servlet implementation class JCAServlet
 */
@WebServlet("/JCAServlet")
public class JCAServlet extends HttpServlet
{
 private static final long serialVersionUID = 4283052088313275418L;

 // 1. Use JNDI to look up the connection factory
```

```

@Resource(lookup = "eis/defaultCICSConnectionFactory")
private ConnectionFactory cf;

protected void doGet(HttpServletRequest request, HttpServletResponse
response)
throws ServletException, IOException
{
try
{
// 2. Get the connection object from the connection factory
Connection conn = cf.getConnection();

// 3. Get an interaction object from the connection
Interaction interaction = conn.createInteraction();

// 4. Create a new ECIIInteractionSpec
ECIIInteractionSpec is = new ECIIInteractionSpec();

// 5. Use the set methods on ECIIInteractionSpec
// to set the properties of execution.
// Change these properties to suit the target program
is.setCommareaLength(20);
is.setFunctionName("PROGNAME");
is.setInteractionVerb(ECIIInteractionSpec.SYNC_SEND_RECEIVE);

// 6. Create a record object to contain the input data and populate
データ
// Change the contents to suit the data required by the program
RecordImpl in = new RecordImpl();
byte[] commarea = "COMMAREA contents".getBytes();
ByteArrayInputStream inStream = new ByteArrayInputStream(commarea);
in.read(inStream);

// 7. Create a record object to contain the output data
RecordImpl out = new RecordImpl();

// 8. Call the execute method on the interaction
interaction.execute(is, in, out);

// 9. Read the data from the output record
ByteArrayOutputStream outStream = new ByteArrayOutputStream();
out.write(outStream);
commarea = outStream.toByteArray();
}
catch (Exception e)
{
// Handle any exceptions by wrapping them into an IOException
throw new IOException(e);
}
}

// A simple class which extends Record and Streamable representing a
commarea.
public class RecordImpl implements Streamable, Record
{
private static final long serialVersionUID = -947604396867020977L;

private String contents = new String("");

@Override
public void read(InputStream is)
{
try
{
int total = is.available();
byte[] bytes = null;
if (total > 0)

```

```

{
 bytes = new byte[total];
 is.read(bytes);
}
// Convert the bytes to a string.
contents = new String(bytes);
}
catch (Exception e)
{
 // Log the exception
 e.printStackTrace();
}
}

@Override
public void write(OutputStream os)
{
 try
 {
 // Output the string as bytes
 os.write(contents.getBytes());
 }
 catch (Exception e)
 {
 // Log the exception
 e.printStackTrace();
 }
}

@Override
public String getRecordName()
{
 // Required by Record, unused in this sample
 return "";
}
@Override
public void setRecordName(String newName)
{
 // Required by Record, unused in this sample
}
@Override
public void setRecordShortDescription(String newDesc)
{
 // Required by Record, unused in this sample
}
@Override
public String getRecordShortDescription()
{
 // Required by Record, unused in this sample
 return "";
}
@Override
public Object clone() throws CloneNotSupportedException
{
 // Required by Record, unused in this sample
 return super.clone();
}
}
}

```

タスクの結果

ECI リソース・アダプターを使用した CICS 内のプログラムへのリンクの作成が成功しました。

JCA ローカル ECI リソース・アダプターをチャンネルおよびコンテナーと共に使用する:

JCA ローカル ECI リソース・アダプターと共にチャンネルおよびコンテナーを使用するには、入出力レコードが ECICChannelRecord のインスタンスである必要があります。

ECICChannelRecord が ECIIInteraction の execute() メソッドに渡されると、そのメソッドは ECICChannelRecord そのものを使用してチャンネルを作成し、ECICChannelRecord 内の項目をコンテナーに変換してから、それを CICS に渡します。

この例は、ECI ChannelRecord で put() メソッドおよび get() メソッドを使用して JCA ローカル・リソース・アダプター用の入出力レコードを作成する方法を示しています。

```
ECICChannelRecord in = new
 ECICChannelRecord("CHANNELNAME");
byte[] bitData = "Container with BIT data".getBytes();
String charData = "Container with CHAR data";
in.put("BITCONTAINER", bitData);
in.put("CHARCONTAINER", charData);
ECICChannelRecord out = new ECICChannelRecord("CHANNELNAME");

interaction.execute(is, in, out);

bitData = (byte[]) out.get("BITCONTAINER");
charData = (String) out.get("CHARCONTAINER");
```

入力の種類に応じて、BIT コンテナーおよび CHAR コンテナーが次のように作成されます。

- 入力データが byte[] 型である場合、または Streamable インターフェースを実装するオブジェクトである場合には、BIT コンテナーが作成されます。コード・ページ変換は行われません。
- 入力データが String 型である場合には、CHAR コンテナーが作成されます。ストリング・データは Unicode でエンコードされ、コンテナーのエンコード方式に変換されます。EXEC CICS GET CONTAINER によってこのコンテナーから読み取られるデータは、コンテナーを使用したコード・ページ変換に従って変換されます。

ECICChannelRecord を作成するときに、名前は有効な CICS チャンネル名でなければなりません。作成された後、getRecordName() メソッドがチャンネルの名前を取得します。ECICChannelRecord にコンテナーを追加するとき、コンテナー名は有効な CICS コンテナー名でなければなりません。作成された後、KeySet() メソッドがすべてのコンテナーの名前を取得します。

JCA ローカル ECI リソース・アダプターを COMMAREA と共に使用する:

JCA ローカル ECI リソース・アダプターと共に COMMAREA を使用するには、入出力レコードが、javax.resource.cci.Record および javax.resource.cci.Streamable を実装するクラスのインスタンスである必要があります。

この例は、Streamable インターフェースで read() メソッドおよび write() メソッドを使用してローカル ECI リソース・アダプター用の入出力レコードを作成する方法を示しています。

```
RecordImpl in = new RecordImpl();
byte[] commarea = "COMMAREA contents".getBytes();
ByteArrayInputStream inStream = new ByteArrayInputStream(commarea);
in.read(inStream);
RecordImpl out = new RecordImpl();

interaction.execute(is, in, out);

ByteArrayOutputStream outStream = new ByteArrayOutputStream();
out.write(outStream);
commarea = outStream.toByteArray();
```

出力レコードからバイト配列を取得するには、**java.io.ByteArrayOutputStream** オブジェクトを使って Streamable インターフェースで write メソッドを使用します。**ByteArrayOutputStream** 上の toByteArray() メソッドは、COMMAREA からの出力データをバイト配列形式で提供します。

特定の JEE コンポーネントの機能を増強するために、コンストラクターを使用してレコード内容を設定できるようにする Record インターフェースの実装を作成することもできます。この方法により、例で使用した **java.io.ByteArrayInputStream** の使用を避けることができます。

Rational Application Developer に備わっている J2C ツールを使用すると、COBOL コピーブックなどの特定のネイティブ言語構造から Record インターフェースの実装を作成でき、Java と Java 以外のデータ型の間のデータ・マーシャルも標準でサポートされます。

#### JCA による作業単位の管理:

CICS ローカル ECI リソース・アダプターを使用する際には、CICS Liberty JVM サーバーによってトランザクション管理機能が提供されます。

CICS ローカル ECI リソース・アダプターを使用する他の CICS プログラムの呼び出しは、CICS 作業単位 (UOW) 管理に統合されます。これにより、syncpoint コマンドまたは JTA トランザクションを介して UOW を制御することができます。

リモート CICS 領域でのプログラムの呼び出しを実行すると、ミラー・トランザクションを使用した DPL 呼び出しになります。Java トランザクション・コンテキストが使用されている場合、このミラー・タスク UOW は呼び出し側 UOW によって調整されます。この場合、呼び出される側のプログラムは DPL コマンド・サブセットに制限されるため、syncpoint 呼び出しを発行できません。呼び出し側プログラムに JTA トランザクション・コンテキストが含まれない場合は、SYNCONRETURN オプションを使用してミラー・タスク UOW が呼び出されます。このシナリオでは、呼び出されるプログラムの UOW が呼び出し側プログラムによって調整されないため、呼び出されるプログラムは syncpoint コマンドを発行できません。

詳しくは、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、745 ページの『作業単位 (UOW) サービス』、および 766 ページの『Java Transaction API (JTA)』を参照してください。

**JCA ローカル ECI リソース・アダプターのトレースの有効化:**

JCA ローカル ECI リソース・アダプターのトレースの仕組みを詳しく説明します。リソース・アダプターを使用するアプリケーションの問題を解決する際には、トレースを有効にすると便利な場合があります。

- JCA ローカル ECI リソース・アダプターのトレースは、SJ ドメイン・トレース・レベル 4 (SJ = 4 または SJ = ALL) によって有効になります。
- リソース・アダプターからのトレースは、コンポーネント ID `com.ibm.cics.wlp.jca.local.eci.adapter` を使って zFS で JVM サーバー・トレース出力に含まれます。

**JCA ローカル ECI リソース・アダプターの制限:**

CICS Transaction Gateway ECI リソース・アダプターで使用可能な一部の API 呼び出しは、CICS TS JCA ローカル ECI リソース・アダプターによってサポートされていません。

制限されたメソッド

以下の API 呼び出しは、JCA ローカル ECI リソース・アダプターによってサポートされていません。

- `ECIInteractionSpec` クラスの `setExecuteTimeout()`、`getExecuteTimeout()`、`setReplyLength()`、`getReplyLength()`、`setTranName()`、`getTranName()` メソッド
- `CICSConnectionSpec` クラスの `setPassword()`、`getPassword()`、`setUserid()`、`getUserid()`、`addPropertyChangeListener()`、`removePropertyChangeListener()`、`firePropertyChange` メソッド
- `ECIConnection` クラスの `getLocalTransaction()` メソッド
- `ECIChannelRecord` クラスの `values()` メソッド
- `CICSUserInputException`
- コンストラクター `ECIConnectionSpec(String username, String password)`。`ECIConnectionSpec()` が代わりに追加されました。
- コンストラクター `ECIInteractionSpec(int verb, int timeout, String prog, int commLen, int repLen)`。`ECIInteractionSpec(int verb, String prog, int commLen)` が代わりに追加されました。

これらの呼び出しは、Web アプリケーションの開発に IBM CICS SDK for Java を使用している場合はサポートされません。既存の ECI JCA アプリケーションを Liberty JVM サーバーに移植可能にするために、これらのメソッドは引き続き機能しますが、トランザクション、タイムアウト、応答の長さ、およびトランザクション名の設定は影響を及ぼしません。`ECIInteractionSpec.setTPNName()` でトランザクション ID を設定すると、リモート・プログラム (DPL) にリンクするときに、指定されたトランザクションのみが使用されます。ローカル・プログラムへのリンクは、引き続き現在のトランザクションを使用します。

## 非管理対象環境

JCA ローカル ECI は、(server.xml の構成により作成された) 管理接続ファクトリーのみをサポートします。ManagedConnectionFactory のインスタンスを使用して作成された非管理対象接続はサポートされません。

## 例外処理

CICS Transaction Gateway ECI リソース・アダプターと CICS TS JCA ローカル ECI リソース・アダプターでは、例外処理が少し異なる可能性があります。CICS のエラーは、JCICS API の場合と同様に、ECI ローカル・リソース・アダプターに CICSException として伝搬されます。リソース・アダプターは、これらの例外を ResourceException でラップします。CICS の障害を特定できるように、CICSException が例外の原因として設定され、java.lang.Throwable の getCause() メソッドを使用してアクセスできます。

## 非同期呼び出し

非同期呼び出しは、JCA ローカル ECI リソース・アダプターでは完全な非同期ではありません。SYNC\_SEND 対話 verb を使用した呼び出しは、プログラムが完了するまでブロックし、その後 SYNC\_RECEIVE 対話 verb を使用して、同じ ECIIInteraction を使い、後続の呼び出しによって結果を収集できます。

## サポートされない CICS Transaction Gateway の機能

次の CICS Transaction Gateway の機能は、CICS TS ローカル ECI リソース・アダプターではサポートされません。

- CICS Transaction Gateway サーバーへのリモート接続
- ID 伝搬
- クロス・コンポーネント・トレース (XCT)
- ユーザー出口のモニタリング要求
- リソース・アダプターからのトレースは CICS TS によって制御され、CICSLogTraceLevels の使用はサポートされていません。

## Java 向け CICS リモート開発機能

Java 向け CICS リモート開発機能は、開発者のワークステーションで稼働する Liberty で使用できる ECI リソース・アダプターを提供します。この機能により、開発者は JCA API を使用して CICS TS 内のプログラムを呼び出す Java アプリケーションを迅速にテストおよびデバッグできます。準備ができたら、アプリケーションにさらに変更を加えることなく、アプリケーションを CICS 内で稼働する Liberty にデプロイできます。

この機能は CICS 領域に接続するために、TCPIPService リソースを使用して定義された IP 相互接続性 (IPIC) 接続を使用します。CICS TS 内のプログラムとの間で送受信されるデータの問題を識別するために、トレース機能を使用できるようになっています。

## IPIC 接続の構成:

CICS 領域で Java アプリケーションをテストする前に、IPIC 接続が使用可能になっている必要があります。CICS システム・プログラマーに連絡して、以下の詳細を使用して Liberty プロファイルからの IPIC 要求を受け入れる TCP/IP サービスを要請してください。

### このタスクについて

以下の手順では、CICS システム・プログラマーが CICS 内に TCP/IP サービスを定義して、IPIC 接続用のサンプル・ユーザー・プログラムをインストールするためのステップを説明します。

### 手順

1. **TCPIPService** リソースに次の属性を定義して、CICS に IPIC サポートをインストールします。

表 64. TCPIPService リソースの属性

TCPIPService リソース属性	必要な値
<b>URM</b>	DFHISAIP
ポート番号	n
状況	OPEN
<b>Protocol</b>	IPIC
トランザクション	CISS
<b>Backlog</b>	0
<b>Socketclose</b>	いいえ

2. **CEMT INQUIRE TCPIPService(JCA)** コマンドを実行して、**TCPIPService** がサービス中であることを確認します。
3. オプション: IPIC 接続をテストするためのサンプル・プログラムをインストールします。
  - a. CICSTransaction Gateway Software Development Kit (SDK) をダウンロードし (まだコピーを持っていない場合)、アーカイブ・ファイルを展開します。
  - b. cicsprograms/ec01.cpp メンバーを見つけて、z/OS 上の COBOL ソース・データにコピーします。
  - c. EC01 サンプル・プログラムをコンパイルし、生成されたモジュールを、CICS がアクセスできるロード・ライブラリーにコピーします。
  - d. autoinstall プログラムが有効になっていない場合は、EC01 のプログラム定義を作成してインストールします。
  - e. **CECI LINK PROG(EC01) COMMAREA(' ')** を実行して EC01 プログラムをテストします。 **RESPONSE** が **NORMAL** であることを確認します。

### タスクの結果

これで、IPIC サポートを CICS 領域で利用できるようになりました。

ローカル **Java** テスト環境のセットアップ:

CICS 領域で Java アプリケーションをテストする前に、必要なツールがインストールされていることを確認するとともに、ローカル作業環境を構成する必要があります。

このタスクについて

CICS 領域で Java アプリケーションをテストできるようにローカル作業環境を作成するには、次の手順を実行します。

手順

1. WebSphere Developer Tools (WDT) が含まれる Eclipse IDE for Java EE Developers をダウンロードしてインストールします。次に、ローカル Liberty プロファイル・サーバー・インスタンスをインストールし、Hello World JavaServer Pages (JSP) を作成します。インストールしたサーバー上に Hello World Web アプリケーションをデプロイすることによってテストします。詳しくは、「Getting started with WebSphere Developer Tools for Eclipse and Liberty」を参照してください。
2. Liberty リポジトリから JCA リモート ECI リソース・アダプターをインストールします。以下のように **installUtility** コマンドを使用することで、リポジトリからフィーチャーをインストールできます。

```
<liberty_install>/bin/installUtility install
--acceptLicense jcaRemoteEci-1.0
```

3. `usr:jcaRemoteEci-1.0`、`localConnector-1.0`、および `webProfile-6.0` フィーチャーを `server.xml` ファイルに追加します。例えば、Eclipse で「WebSphere Application Server Liberty Profile」プロジェクトを展開してから、「サーバー (servers)」を展開します。`server.xml` を編集するために、「defaultServer」をダブルクリックします。「ソース」タブをクリックし、以下のフィーチャーを追加します。

```
<featureManager>
...
<feature>usr:jcaRemoteEci-1.0</feature>
<feature>localConnector-1.0</feature>
<feature>webProfile-6.0</feature>
...
</featureManager>
```

4. **connectionFactory** および **properties.com.ibm.cics.wlp.jca.remote.eci** を `server.xml` に追加します。

**connectionFactory** `jndiName` は、アプリケーションが接続を作成するために使用します。**properties.com.ibm.cics.wlp.jca.remote.eci** は、JCA リモート ECI リソース・アダプターを構成する際に使用するため、少なくとも **serverName** で、TCPIPService リソースで定義されている IPIC 接続のホスト名とポート番号を指定する必要があります。

注: 追加のパラメーターを指定しなければならない場合があります。例えば、Secure Sockets Layer (SSL) を使用するには、ユーザー ID とパスワードを指定する必要があります。表 1 に、使用可能なパラメーターをリストします。表 2 に、JCA リモート ECI リソース・アダプターでサポートされていない ECI

リソース・アダプター・デプロイメント・パラメーターをリストします。詳しくは、ECIリソース・アダプター・デプロイメント・パラメーターを参照してください。

```
<server>
...
<connectionFactory id="com.ibm.cics.wlp.jca.local.eci" jndiName="eis/ECI">
 <properties.com.ibm.cics.wlp.jca.remote.eci serverName="tcp://"
hostname
:
port"/>
</connectionFactory>
...
</server>
```

表 1 に、サポートされている JCA リモート ECI リソース・アダプター・プロパティーを示します。

表 65. サポートされている JCA リモート ECI リソース・アダプター・プロパティー

JCA オブジェクト: プロパティー	注
<b>applid</b>	
<b>applidQualifier</b>	このプロパティーは必須です。
<b>cipherSuites</b>	
<b>ipicHeartbeatInterval</b>	
<b>ipicSendSessions</b>	このプロパティーはデフォルトで 5 に設定されます。
<b>keyRingClass</b>	
<b>keyRingPassword</b>	
<b>password</b>	
<b>socketConnectTimeout</b>	
<b>serverName</b>	このプロパティーは必須です。
<b>traceLevel</b>	
<b>traceRequest</b>	
<b>userName</b>	

表 2 に、JCA リモート ECI リソース・アダプターでサポートされていない CICS Transaction Gateway ECI リソース・アダプター・デプロイメント・パラメーターを示します。

表 66. サポートされていない JCA リモート ECI リソース・アダプター・プロパティー

JCA オブジェクト: プロパティー
<b>interceptPlugin</b>
<b>portNumber</b>
<b>tpNName</b>
<b>tranName</b>

サンプル **Java EE JCAServlet** アプリケーションのテスト:

サンプル Java EE JCAServlet アプリケーションを追加した後、この Java EE アプリケーションが CICS 内のサンプル・プログラムを呼び出せることを確認します

このタスクについて

次の手順を実行して、Java EE JCServlet アプリケーションを追加します。その後、Java EE JCServlet アプリケーションが CICS 内の EC01 サンプル・プログラムを呼び出せることを確認します。

手順

1. Hello World Web アプリケーション内に JCServlet クラスを作成します。  
「Hello World」プロジェクトを展開してから、「Java リソース」を展開します。「新規」を右クリックし、「サーブレット」を選択します。
  - 「Java パッケージ」には `com.ibm.ctg.samples.liberty` と入力します。
  - 「クラス名」には JCServlet と入力します。その後、「終了」をクリックします。
2. JCServlet.java を編集し、すべてのコードを、GitHub から取得した CICS のサンプル JCServlet.java に置き換えます。詳しくは、JCServlet.java を参照してください。
3. 「Hello World」プロジェクトを展開してから、「Java リソース」 > 「src」 > 「com.ibm.ctg.samples.liberty」を展開します。JCServlet.java アプリケーションを右クリックし、「実行」 > 「サーバーで実行」を選択します。
4. Liberty サーバーが始動され、Liberty サーバー・コンソールに URL を示すメッセージが表示されます。この URL をクリックすると Java EE アプリケーションを実行できます。以下に、表示されるメッセージの一例を示します。

```
[AUDIT] CWWKT0016I: Web application available
(default_host):
http://localhost:9080/GenappCustomerSearchWeb/
```

タスクの結果

これで、ローカル Liberty サーバーで Java EE アプリケーションをテストし、デバッグできるようになりました。

ローカル Liberty プロファイル内でのトレース機能の構成:

ローカル Liberty プロファイル内で Java Web アプリケーションをトレースする前に、ローカル作業環境を構成する必要があります。

このタスクについて

次の手順を実行して、ローカル Liberty プロファイル内でトレース機能を構成します。

手順

トレースを有効にするには、`server.xml` ファイル内の接続ファクトリーに `traceRequests="ON"` パラメーターを追加します。  
`traceRequests="ON"` を指定した状態でアプリケーション要求を送信すると、Eclipse コンソールにアプリケーションが送信する要求と CICS から受信する応答が表示されます。

以下の例に、CICS に送信された要求と、CICS から受信した応答を示します。

```

Starting DataFlowsMonitor log stream at
Thu Apr 07 14:21:54 BST 2016[00000000001]:
com.ibm.ctg.monitoring.DataFlowsMonitor:eventFired called with
event = RequestEntry
FlowType = EciSynconreturn Fully qualified APPLID = No APPLID
CtgCorrelator =
IProgram = EC01Server =
TCP://WINMVS2C.HURSLEY.IBM.COM:27723PayLoad = COMMAREA is 20
bytes00000000 00000000 00000000 00000000 00000000
'?????????????????'

[000000000001]:
com.ibm.ctg.monitoring.DataFlowsMonitor:eventFired called with
event = ResponseExit
FlowType = EciSynconreturn Fully qualified APPLID = No APPLID
CtgCorrelator =
IOriginData - Transaction Group ID = 1B114040
40404040 40402EF0 F0F0F0F0 F0F0F2D0 8F53F115 220100Program = EC01Server =
TCP://WINMVS2C.HURSLEY.IBM.COM:27723PayLoad = COMMAREA is 20
bytesF0F761F0 F461F1F6 40F1F47A F2F17AF5 F4000000
'??a??a??@??z??z?????'CtgReturnCode = 0CicsReturnCode = 0

```

## タスクの結果

これで、問題を識別できるよう、アプリケーションをトレースすることが可能になります。

## セキュア SSL 接続の構成:

SSL を使用して、JCA リモート ECI リソース・アダプターから CICS への IPIC 接続を保護できます。

## このタスクについて

セキュア SSL 接続を構成するには、次の手順を実行します。

このセットアップを完了すると、MVS とローカル・クライアントの両方からエクスポートされた信頼できる証明書が SSL に提供されます。認証には MVS ユーザー ID とパスワードも必要になります。

## 手順

1. CICS RACF 環境をセットアップします。詳しくは、Configuring SSL server authentication on the CICS server を参照してください。
2. クライアント・セキュリティーをセットアップします。詳しくは、Configuring SSL server authentication on the client を参照してください。
3. クライアント認証を構成します。詳しくは、Configuring SSL client authentication を参照してください。
4. CICS 上で IPIC 接続を構成します。詳しくは、Configuring the IPIC connection on CICS を参照してください。
5. sever.xml を変更して、ステップ 2 で作成したローカル **KeyRingClass** を使用し、ユーザー ID とパスワードを送信するようにします。

```

<connectionFactory id="com.ibm.cics.wlp.jca.local.eci"
jndiName="eis/ECI">
<properties.com.ibm.cics.wlp.jca.remote.eci
serverName="ssl://hostname:port"
keyRingClass="C:\Users\IBM_ADMIN\Documents\CICS\JCA\ctgclientkeyring.jks"
keyRingPassword="password"

```

```

 userName="user_ID"
 password="*****"
 applid="JCASSL"
 applidQualifier="ABCDEFGH"
 />
</connectionFactory>

```

タスクの結果

JCA リモート ECI リソース・アダプターが、SSL と、server.xml に指定された鍵リング、ユーザー ID、およびパスワードを使用して、CICS への要求を保護するようになります。

## MicroProfile によるマイクロサービスの開発

Eclipse MicroProfile では、Enterprise Java 環境でマイクロサービス・アプリケーションを開発するためのプログラミング・モデルを定義します。これは、Eclipse Foundation の下にあるオープン・ソース・プロジェクトであり、マイクロサービスを Enterprise Java コミュニティーに提供します。MicroProfile は Liberty でサポートされます。

MicroProfile では、弾力性がありセキュアで、モニターが容易なマイクロサービスを作成するための多数の仕様が定義されます。

表 67. Eclipse MicroProfile 1.2 に含まれる内容

仕様	説明
JSR 346: Contexts and Dependency Injection for Java EE 1.1	CDI では、Enterprise Java ランタイムでのオブジェクトの注入とライフサイクルを管理する一連のサービスが定義されます。
JSR 339: JAX-RS 2.0: The Java API for RESTful Web Services	JAX-RS は、Java API for RESTful Web Services です。
JSR 353: Java API for JSON Processing	JSON-P は、JSON を処理するための Java API です。
Eclipse MicroProfile Config 1.1	Config は、アプリケーション構成を管理するための Java API および SPI です。
Eclipse MicroProfile Fault Tolerance 1.0	Fault Tolerance では、外部サービスの呼び出し時に障害を処理するための戦略を提供します。
Eclipse MicroProfile Health Check 1.0	Health Check を使用すると、コンポーネントはその稼働状況を広範囲のシステムに報告できます。
Eclipse MicroProfile Health Metrics 1.0	Health Metrics では、アプリケーションでモニター・データを公開するための統一された方法を提供します。
Eclipse MicroProfile JWT Propagation 1.0	JWT Propagation では、Java EE の役割ベースのアクセス制御 (RBAC) による認証および許可に JSON Web Token (JWT) を使用できます。

## 既知の制約事項

- CDI は MicroProfile API で幅広く使用されていますが、Liberty では、エンタープライズ・バンドル・アーカイブ (EBA) にパッケージ化された OSGi Web アプリケーションでの CDI はサポートされません。代わりに、MicroProfile を使用するアプリケーションを Web アプリケーション・アーカイブ (WAR) またはエンタープライズ・アプリケーション・アーカイブ (EAR) にパッケージ化します。
- MicroProfile Fault Tolerance 1.0 は、他のサービスに対する呼び出しを管理するように設計されています。トランザクション・コンテキスト内のリソースに対する更新を管理するには設計されていません。CICS リソースは、@Bulkhead、@CircuitBreaker、@Fallback、@Timeout、または @Retry の注釈が付けられたメソッドでは更新できません。CICS では、JTA が使用されている場合でも、例外の発生時にこれらの更新がリカバリーされる保証はありません。
- mpJwt-1.0 フィーチャーが Liberty JVM サーバーの server.xml で有効になっている場合、すべての認証を JWT ベアラー・トークンを使用して行う必要があります。他の形式の認証を使用するには、別個の Liberty JVM サーバーを使用する必要があります。

## CICS Liberty JVM サーバーのサービス・アーキテクチャー

### モノリシック・アーキテクチャー

モノリシック・アーキテクチャーでは、単一の単位でアプリケーションを実装します。内部的にはロジックをモジュール形式にすることができますが、外部的には、アプリケーションは全体的に使用可能であるか、まったく使用不可であるかのいずれかになります。モノリスはマイクロサービスと比較して良好に機能し、セキュリティおよびトランザクション・コンテキストを管理する場合、それほど複雑ではありません。モノリスのスケーリングでは、アプリケーション全体のインスタンスを追加する必要があり、各パーツを個別にスケーリングすることはできません。

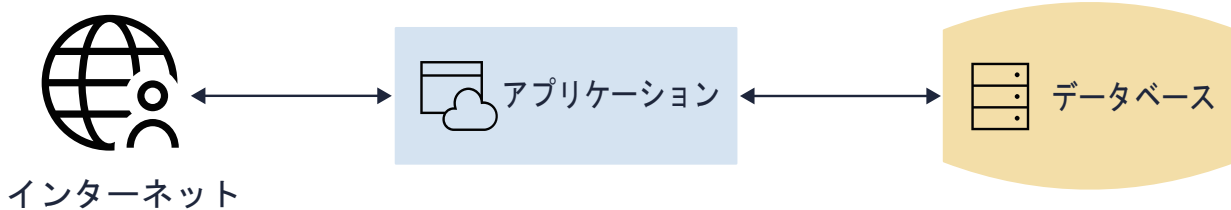


図 134. モノリシック・アーキテクチャー

### バックング・サービス

バックング・サービスを使用すると、バックエンド・データおよびプログラムをメイン・アプリケーションから分離できます。データとプログラムは、別々のアプリケーションにすることによって、プラットフォーム非依存の通信方式 (HTTP、ソケット、メッセージ・キューなど) を使用して呼び出されます。これらのソースと通信するためのすべてのロジックを保持するメイン・アプリケーションの代わりに、これらのサービスで一部の作業が行われます。

CICS では、z/OS Connect を使用し、CICS プログラムを REST API を介してバックキング・サービスとして公開します。この例では、アプリケーションは JDBC を使用してデータベースと通信します。E メールを送信には SMTP が使用され、z/OS Connect を介した CICS プログラムの呼び出しには HTTP が使用されます。

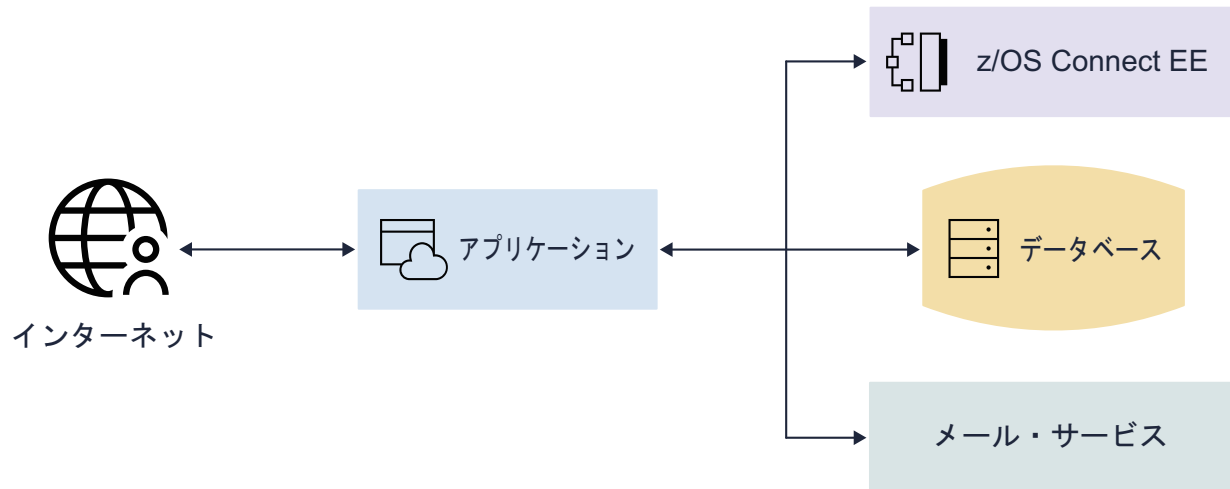


図 135. バックキング・サービス

### ホスト・サービス

サービスは、メイン・アプリケーションをさまざまなコンポーネントからさらに分離するために、CICS でホストされます。バックキング・サービスと同様に、追加機能は CICS Web サービスを介して CICS で公開されるか、またはサーブレット、JAX-RS、JAX-WS などのテクノロジーを使用して、アプリケーションにより CICS Liberty で公開されます。

JAX-RS は RESTful Web サービスを作成するための一般的なテクノロジーであり、JAX-WS は、リモート手続き呼び出し (RPC) 指向の Web サービスを作成するために使用されます。

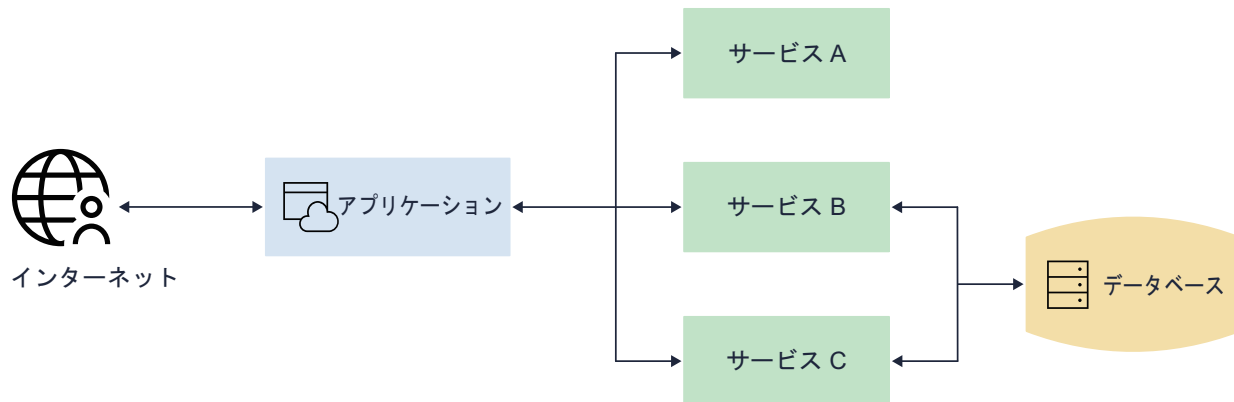


図 136. ホスト・サービス

注: REST と RPC はどちらも同様に、マイクロサービスでの通信に有効なオプションです。REST はリソース管理に重点を置いています。RPC はアクションに重点を置いています。マイクロサービス・アーキテクチャーでは、REST、RPC、またはその他のテクノロジーは必須ではありません。

## マイクロサービス

完全なマイクロサービス・アーキテクチャーは、分離されたサービスが相互接続された Web であり、単一の中心点はありませんが、専用のエントリー・ポイントは存在する場合があります。サービスは、必要に応じて相互に通信できます。マイクロサービスのスケーリングでは、スケーリングを必要とするパーツのインスタンスを追加する必要があります。マイクロサービスは、モノリスよりも障害に対して弾力性があります。

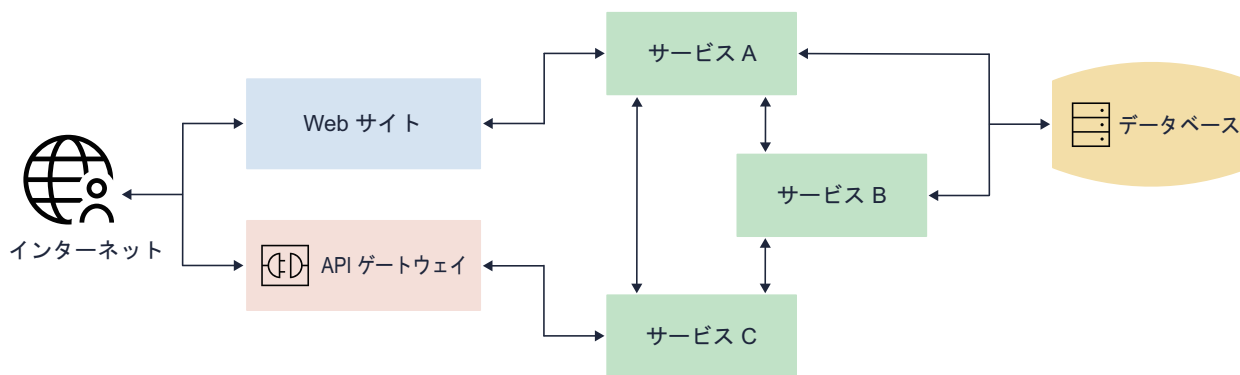


図 137. マイクロサービス

## CICS でのサービスのスケーリング

CICS では、領域のトポロジーおよびセットアップに応じて、サービスを複数の方法でスケーリングできます。通常、マイクロサービスは単一のコンテナに分離されます。CICS では、サービスまたはサービスのセットは、領域または JVM サーバー内で分離できます。スケーリングを行うには、同じサービスまたはサービスのセットをホストする複数の CICS 領域を実行します。また、スレッド数を増加して JVM

サーバーをスケーリングすることもできます。

## マイクロサービスの保護

可能な場合、マイクロサービスではパブリック・ネットワークをオフにしておく必要があります。API ゲートウェイを使用して、マイクロサービスへのアクセスを制御できます。MicroProfile では、マイクロサービス・エンドポイントの役割ベースのアクセス制御 (RBAC) に Open ID Connect (OIDC) ベースの JSON Web Token (JWT) を使用するための方式を提供します。セキュリティ・トークンにより、さまざまなサービスにわたるユーザー ID の単純で相互運用可能な伝搬が可能となります。

MicroProfile JWT Authentication 1.0 では、JWT ベアラー・トークンに基づいてユーザーを認証および許可する機能を提供します。トークンをサービス・コードに注入して、ID をマイクロサービス・ネットワーク全体に伝搬するために使用できます。JWT の伝搬は、アウトバウンド要求の許可 HTTP ヘッダーにベアラー・トークンとして JWT を組み込むことにより、手動で行うことができます。または、次の例のように `server.xml` の `webTarget` エレメントに `authnToken` を構成して、Liberty により JWT を自動的に伝搬することもできます。

```
<webTarget uri="http://microservice.example.ibm.com/protected/*" authnToken="mpjwt" />
```

**重要:** JWT ID は、自動的にユーザー・レジストリーにマップされず、CICS タスク・ユーザー ID に伝搬されません。ID のマッピングを有効にするには、`mapToUserRegistry="true"` 構成属性を `server.xml` の `<mpJwt>` エレメントに追加します。

Liberty での MicroProfile JWT 認証の構成について詳しくは、MicroProfile JSON Web トークンの構成を参照してください。

## マイクロサービスでのデータ整合性

マイクロサービスで分散トランザクションを使用することは容易ではありません。代わりに、saga パターンなどの代替のトランザクション戦略が使用されます。この場合、イベントはサービスで更新された後に公開されます。例えば、サービス A とサービス B にどちらも必須の更新が含まれる場合、以下の順序で行われます。

1. A の更新が保留状態になります。
2. A から B へメッセージが送信されます。
3. B の更新が完了状態になります。
4. B から A へメッセージが送信されます。
5. A の更新が完了状態になります。

## マイクロサービスを使用する場合

マイクロサービスを適用するのに最適なシチュエーションは、アプリケーションをより小さな分離されたサービスに分解可能な場合です。マイクロサービスを使用すると、制御されたスケーリング、独立したデプロイメント、およびより自律型の開発が可能となります。マイクロサービスのアーキテクチャーにより、デプロイメントおよびデータ整合性においては特に、複雑性が増す可能性があります。HTTP などのプロトコルを介した通信では、メモリー内の呼び出しと比較して、パフォーマンス・コストが増加します。コンポーネントを個別にスケーリングできるようにす

ることで、コンポーネントの障害に対する弾力性が向上します。マイクロサービス・アーキテクチャーを管理する際には、正常でないサービスの診断を支援するためにモニター・ソリューションがさらに重要となります。

## Liberty Web サーバー・プラグイン

Web サーバー・プラグインを使用することにより、サポートされる Web サーバーから 1 つ以上のアプリケーション・サーバーに HTTP 要求を転送することが可能になります。

Web サーバー・プラグインを使用する主な理由が 3 つあります。

- 静的コンテンツを提供する Web サーバーの統合を可能にします。
- HTTPS 使用時に Web サーバーの SSL エンドポイントの終了を可能にします。
- 一群の Liberty サーバー間で HTTP 要求のロード・バランシングとフェイルオーバーを可能にします。

Liberty サーバーで `plugin-cfg.xml` ファイルを生成し、そのファイルを Web サーバーのホスト・マシンにコピーすることによって、Web サーバー・プラグインは構成されます。プラグインはインバウンド要求を受け取り、その要求をこのファイルに含まれている構成データと照らしてチェックし、着信 HTTP 要求を構成済み Liberty サーバーの URI とホストに転送します。

Liberty プロファイル・サーバーで `plugin-cfg.xml` を生成する手順で使用する `generatePluginConfig` 操作は、Liberty が提供する `com.ibm.ws.jmx.mbeans.generatePluginConfig` MBean で公開されます。この JMX MBean をリモートから呼び出すには、IBM Java SDK で提供されている JConsole ユーティリティと Liberty サーバーの `restConnector-1.0` 機能を組み合わせて使用するか、または、MBean の必要な操作を呼び出すカスタム JMX アプリケーションを作成します。CICS Liberty サーバーでの JMX の使用について詳しくは、780 ページの『Java Management Extensions API (JMX)』を参照してください。

Web サーバー・プラグインのセットアップの詳細な情報は、WAS Knowledge Center にあります。Web サーバーへのプラグイン構成の追加を参照してください。

---

## Liberty フィーチャー

CICS では、WebSphere Application Server Liberty からのフィーチャーがサポートされます。これにより、Java EE アプリケーションを Liberty JVM サーバーにデプロイできます。

表 2 から 10 のすべてのフィーチャーは、CICS 統合モードの Liberty に関連しています。これらのフィーチャーは、特に断りのない限り、CICS 標準モードの Liberty でも制限なくサポートされます。表 11 では、Liberty フィーチャーを CICS のサービスの品質に統合するための一連の CICS フィーチャーを示します。

Java EE 6 および Java EE 7 からのフィーチャーは、同時に使用しないでください。`server.xml` の編集については、サーバー構成を参照してください。

注: CICS 標準モードの Liberty は Java EE 8 もサポートします。どのフィーチャーが含まれるかについて詳しくは、Liberty での Java EE 8を参照してください。

表 68. Liberty フィーチャーの英字順リスト

フィーチャー <b>A-D</b>	フィーチャー <b>E-Jd</b>	フィーチャー <b>Jm-MpJ</b>	フィーチャー <b>MpM-Z</b>
appClientSupport-1.0	ejb-3.2	jms-1.1	mpMetrics-1.0
appSecurity-1.0	ejbHome-3.2	jmsMdb-3.1	mongodb-2.0
appSecurity-2.0	ejbLite-3.1	jndi-1.0	monitor-1.0
batch-1.0	ejbLite-3.2	jpa-2.0	oauth-2.0
batchManagement-1.0	ejbPersistentTimer-3.2	jpa-2.1	openidConnectClient-1.0
beanValidation-1.0 for JEE7	ejbRemote-3.2	jsf-2.0	openidConnectServer-1.0
beanValidation-1.0 for JEE6	el-3.0	jsf-2.2	osgiConsole-1.0
beanValidation-1.1 for JEE7	j2eeManagement-1.1	json-1.0	osgi.jpa-1.0
beanValidation-1.1 for JEE6	jacc-1.5	jsonp-1.0	restConnector-1.0
blueprint-1.0	jaspic-1.1	jsp-2.2	servlet-3.0
cdi-1.0	javaMail-1.5	jsp-2.3	servlet-3.1
cdi-1.2	javaee-7.0	jta-1.1	sessionDatabase-1.0
cicsts:core-1.0	jaxb-2.2 (JEE7 の場合)	jta-1.2	ssl-1.0
cicsts:defaultApp-1.0	jaxb-2.2 (JEE6 の場合)	jwt-1.0	wab-1.0
cicsts:distributedIdentity-1.0	jaxrs-1.1	ldapRegistry-3.0	wasJmsClient-1.1
cicsts:jcaLocalEci-1.0	jaxrs-2.0	localConnector-1.0	wasJmsClient-2.0
cicsts:jdbc-1.0	jaxrsClient-2.0	managedBeans-1.0	wasJmsSecurity-1.0
cicsts:link-1.0	jaxws-2.2(JEE7 の場合)	mdb-3.1	wasJmsServer-1.0
cicsts:security-1.0	jaxws-2.2(JEE6 の場合)	mdb-3.2	webCache-1.0
cicsts:standard-1.0	jca-1.6	microProfile-1.0	webProfile-6.0
cicsts:zosConnect-1.0	jca-1.7	microProfile-1.2	webProfile-7.0
cicsts:zosConnect-2.0	jcaInboundSecurity-1.0 for JEE7	mpConfig-1.1	websocket-1.0
concurrent-1.0	jcaInboundSecurity-1.0 for JEE6	mpFaultTolerance-1.0	websocket-1.1
distributedMap-1.0	jdbc-4.0	mpHealth-1.0	wmqJmsClient-2.0
	jdbc-4.1	mpJwt-1.0	zosTransaction-1.0

## CICS Liberty フィーチャー

表 69. Java EE 7 Web Profile 用にサポートされる Liberty フィーチャー

Liberty フィーチャー	Liberty フィーチャーの説明	CICS でのフィーチャーの使用
beanValidation-1.0	JavaBeans を検証するためのアノテーション・ベースのモデルを提供します。	
beanValidation-1.1	JavaBeans を検証するためのアノテーション・ベースのモデルを提供します。	

表 69. Java EE 7 Web Profile 用にサポートされる Liberty フィーチャー (続き)

Liberty フィーチャー	Liberty フィーチャーの説明	CICS でのフィーチャーの使用
cdi-1.2	EJB や Managed Bean などのコンポーネントを、JSP や EJB といった他のコンポーネントに注入するためのメカニズムを提供します。	
ejbLite-3.2	EJB 仕様の EJB Lite サブセットに記載されている Enterprise JavaBeans のサポートを有効にします。	制約事項: トランザクション属性「NotSupported」は、Liberty JVM サーバーではサポートされません。
el-3.0	Enterprise Language (EL) 3.0 仕様のサポートを可能にします。	
jaxrs-2.0	Liberty での Java API for RESTful Web Services (JAX-RS) のサポートを提供します。	
jaxrsClient-2.0	Java Client API for JAX-RS 2.0 のサポートを有効にします。	jaxrsClient-2.0 フィーチャーは、jaxrs-2.0 によって有効になります。JAX-RS 2.0 クライアントの構成。
jdbc-4.1	アプリケーションからデータベースにアクセスするためのデータ・ソースの構成を使用可能にします。	注: DB2 JCC ドライバー内の jdbc-4.0 実装と jdbc-4.1 実装を同時に使用することはできません。
jndi-1.0	Liberty のサーバー構成における、単一の Java Naming and Directory Interface (JNDI) エントリ定義のサポートを提供します。	
jpa-2.1	アプリケーション管理およびコンテナ管理の JPA を使用したアプリケーションのサポートを有効にします。	
jsf-2.2	JavaServer Faces (JSF) フレームワークを使用する Web アプリケーションのサポートを提供します。	
jsonp-1.0	JavaScript Object Notation を処理する Java API の定義をサポートします。これには、JSON の構文解析、生成、変換、および照会機能のサポートが含まれます。	
jsp-2.3	サーブレットおよび JavaServer Pages (JSP) アプリケーションのサポートを有効にします。	752 ページの『Java EE アプリケーションと Liberty アプリケーション』
managedBeans-1.0	コンテナによって管理される、さまざまな種類の Java EE コンポーネントのための共通基盤を提供します。Managed Bean で利用できる共通サービスには、リソース・インジェクション、ライフサイクル管理、インターセプターの使用などがあります。	

表 69. Java EE 7 Web Profile 用にサポートされる Liberty フィーチャー (続き)

Liberty フィーチャー	Liberty フィーチャーの説明	CICS でのフィーチャーの使用
servlet-3.1	Java サブレット仕様に書き込まれる HTTP サブレットのサポートを提供します。	752 ページの『Java EE アプリケーションと Liberty アプリケーション』 <b>制約事項:</b> サブレット・ログインおよびログアウト API を使用すると、ユーザー ID が CICS タスクに伝搬されません。
webProfile-7.0	Java EE 7 Web プロファイルをサポートするのに必要な Liberty フィーチャーの便利な組み合わせを提供します。	
websocket-1.0	Web ブラウザーまたはクライアント・アプリケーションと Web サーバー・アプリケーションが単一の全二重接続を使用して通信できるようにします。	
websocket-1.1	Web ブラウザーまたはクライアント・アプリケーションと Web サーバー・アプリケーションが単一の全二重接続を使用して通信できるようにします。	

表 70. Java EE 7 Full Platform 用にサポートされる Liberty フィーチャー

Liberty フィーチャー	Liberty フィーチャーの説明	CICS でのフィーチャーの使用
appClientSupport-1.0	Liberty サーバーによるクライアント・モジュールの処理およびリモート・クライアント・コンテナのサポートを可能にします。	<b>ヒント:</b> アプリケーション・クライアント・モジュールは、クライアントとサーバーの両方で実行されます。クライアントは、アプリケーションのクライアント固有のロジックを実行します。コードの他の部分はサーバー上のクライアント・コンテナ内で実行され、サーバーで実行されているビジネス・ロジックからクライアントにデータを送信します。詳細については、アプリケーション・クライアントの準備と実行を参照してください。
batch-1.0	JSR-352 で定義されている Java Batch 1.0 API のサポートを可能にします。このフィーチャーは、エンタープライズ・バンドル・アーカイブ (EBA) にパッケージ化された Java バッチ・アプリケーションはサポートしていません。	<b>制約事項:</b> DB2 JDBC タイプ 2 の接続は、バッチ・パーシスタンスではサポートされていません。

表 70. Java EE 7 Full Platform 用にサポートされる Liberty フィーチャー (続き)

Liberty フィーチャー	Liberty フィーチャーの説明	CICS でのフィーチャーの使用
concurrent-1.0	管理対象の実行プログラムの作成を可能にします。これにより、同時に実行可能な複数のタスクをアプリケーションが実行依頼できるようになります。	制約事項: トランザクション・プロパティ 「ManagedTask.SUSPEND」は、Liberty JVM サーバーでサポートされていません。 制約事項: 新しいスレッドのトランザクションに関連付けられるユーザー ID は、常に親トランザクションに関連付けられているユーザー ID です。 制約事項: ManagedThreadFactory を使用すると、CICS 対応 Java スレッドではなく標準 Java スレッドが作成されます。
ejb-3.2	EJB 3.2 仕様に従って作成された Enterprise JavaBeans のサポートを可能にします。	
ejbHome-3.2	EJB 2.x API のサポートを提供します。	
ejbPersistentTimer-3.2	永続 EJB タイマーのサポートを提供します。	制約事項: DB2 JDBC タイプ 2 の接続は、永続 EJB タイマーではサポートされていません。
ejbRemote-3.2	リモート EJB インターフェースのサポートを提供します。	
jacc-1.5	Java Authorization Contract for Containers (JACC) バージョン 1.5 のサポートを可能にします。	Java Authorization Contract for Containers (JACC) 許可プロバイダーの開発
jaspic-1.1	Java Authentication SPI for Containers (JASPIC) により、Java EE アプリケーション・サーバーがカスタム認証を使用できるようになります。JASPIC プロバイダーは JSR-196 で定義されています。JASPIC プロバイダーと TAI が同じサーバーに構成されている場合、TAI は無効になります。JASPIC は標準 Java EE テクノロジーであるため、Java EE アプリケーションには TAI よりも移植可能性に優れたソリューションです。	
j2eeManagement-1.1	JEE アプリケーション・サーバー内のアプリケーションを管理およびモニターするための一連のインターフェースを提供します。	

表 70. Java EE 7 Full Platform 用にサポートされる Liberty フィーチャー (続き)

Liberty フィーチャー	Liberty フィーチャーの説明	CICS でのフィーチャーの使用
javaMail-1.5	アプリケーションで JavaMail 1.5 API を使用で きるようにします。	
javaee-7.0	Java EE 7.0 Full Platform をサポートする Liberty フ ィーチャーを結合します。	
jaxb-2.2	Java クラスと XML 表現を マップするためのサポート を提供します。	
jaxws-2.2	SOAP Web サービスのサポ ートを提供します。	
jca-1.7	アプリケーションからエン タープライズ情報システム (EIS) にアクセスするための リソース・アダプターの構 成を有効にします。	785 ページの『Java EE コネクター・アーキテクチャー (JCA)』 制約事項: JCA API javax.resource.spi.BootstrapContext.createTimer() によって作成されたスレッド内では、JCICS API および DB2 JDBC タイプ 2 の接続機能の使用はサポートされ ません。同じ効果を持つ機能として、同時 API (javax.enterprise.concurrent. ManagedScheduledExecutorService) を使用します。
jcaInboundSecurity-1.0	javax.resource.spi.work.SecurityContext 抽象クラスを拡張すること で、JCA インバウンド・リ ソース・アダプターがセキ ュリティー・コンテキスト を渡せるようにします。	
mdb-3.2	EJB 3.2 仕様に従って作成さ れたメッセージ駆動型 Enterprise JavaBeans の使 用を可能にします。MDB に よって、Java EE コンポー ネント内のメッセージの非 同期処理が可能になりま す。	
wasJmsClient-2.0	アプリケーションが JMS API を使用して、Liberty でホストされるメッセー ジ・キューにアクセスでき るようにします。	778 ページの『Java Message Service (JMS)』
wasJmsSecurity-1.0	組み込みメッセージング・ サーバーがクライアントか らのアクセスを認証して許 可できるようにします。	778 ページの『Java Message Service (JMS)』

表 70. Java EE 7 Full Platform 用にサポートされる Liberty フィーチャー (続き)

Liberty フィーチャー	Liberty フィーチャーの説明	CICS でのフィーチャーの使用
wasJmsServer-1.0	サーバー内で組み込みメッセージング・サーバーを使用できるようにします。アプリケーションは、wasJmsClient フィーチャーを使用して、メッセージを操作できます。	778 ページの『Java Message Service (JMS)』

表 71. 拡張プログラミング・モデルに関する Liberty のサポート対象フィーチャー

Liberty フィーチャー	Liberty フィーチャーの説明	CICS でのフィーチャーの使用
json-1.0	Java 環境用の一連の JSON ハンドリング・クラスを提供する、JavaScript Object Notation (JSON4J) ライブラリーを利用できるようにします。	
jta-1.1	Java Transaction API (JTA) をサポートします。 注: Java Transaction API は、保護 Liberty フィーチャーです。	766 ページの『Java Transaction API (JTA)』
jta-1.2	Java Transaction API (JTA) をサポートします。 注: Java Transaction API は、保護 Liberty フィーチャーです。	766 ページの『Java Transaction API (JTA)』
	MongoDB Java Driver のサポートを提供し、サーバー構成でリモート・データベース・インスタンスを構成できるようにします。アプリケーションは、MongoDB API を介してこれらのデータベースとやり取りします。	

表 72. エンタープライズ OSGi に関する Liberty のサポート対象フィーチャー

Liberty フィーチャー	Liberty フィーチャーの説明	CICS でのフィーチャーの使用
blueprint-1.0	OSGi blueprint container 仕様を使用した OSGi アプリケーションをデプロイするためのサポートを有効にします。	制約事項: トランザクション属性「NotSupported」は、Liberty JVM サーバーではサポートされません。
osgi.jpa-1.0	このフィーチャーは、OSGi 機能が組み込まれた、blueprint-1.0 フィーチャーおよび jpa-2.0 フィーチャーに置き換えられました。これらのフィーチャーが両方ともサーバーに追加されると、このフィーチャーが自動的に追加されます。	

表 72. エンタープライズ OSGi に関する Liberty のサポート対象フィーチャー (続き)

Liberty フィーチャー	Liberty フィーチャーの説明	CICS でのフィーチャーの使用
wab-1.0	エンタープライズ・バンドル (EBA) に含まれる Web アプリケーション・バンドル (WAB) のサポートを提供します。	754 ページの『OSGi アプリケーション・プロジェクトの作成』 注: このフィーチャーは、JVM システム・プロパティ <code>com.ibm.cics.jvmserver.wlp.wab=true</code> が設定されていれば、CICS によって自動的に追加されます。

表 73. MicroProfile 用にサポートされる Liberty フィーチャー

Liberty フィーチャー	Liberty フィーチャーの説明	CICS でのフィーチャーの使用
microProfile-1.0	Enterprise Java の Micro Profile をサポートする Liberty フィーチャーを結合します。	
microProfile-1.2	Enterprise Java のための Micro Profile 1.2 をサポートする Liberty フィーチャーを結合します。	このフィーチャーには Java 8 が必要です。
mpConfig-1.1	構成にアクセスするための統一メカニズムが規定されており、複数ソースの単一ビューが提供されます。	このフィーチャーには Java 8 が必要です。
mpFaultTolerance-1.0	Enterprise Java のための MicroProfile Fault Tolerance API のサポートを提供します。	このフィーチャーには Java 8 が必要です。 制約事項: MicroProfile Fault Tolerance 1.0 は、トランザクション (UOW や JTA など) を処理するように設計されていません。CICS リソースの更新は、 @Bulkhead、@CircuitBreaker、@Fallback、@Retry、または @Timeout の注釈が付けられたメソッドでは実行できません。
mpHealth-1.0	Enterprise Java のための MicroProfile Health API のサポートを提供します。	このフィーチャーには Java 8 が必要です。
mpJwt-1.0	Web アプリケーションまたはマイクロサービスは、構成されているユーザー・レジストリーの代わりに、またはそれに加えて、JSON Web Token (JWT) を使用してユーザーを認証することが可能になります。	このフィーチャーには Java 8 が必要です。  属性 <code>ignoreApplicationAuthMethod</code> のデフォルト値は <code>false</code> です。これは、Liberty によって受信されるすべての要求の HTTP ヘッダーに JWT トークンが含まれている必要があることを示します。  属性 <code>mapToUserRegistry</code> のデフォルト値は <code>false</code> です。CICS セキュリティーと統合する場合は、この値を <code>true</code> に設定します。

表 73. MicroProfile 用にサポートされる Liberty フィーチャー (続き)

Liberty フィーチャー	Liberty フィーチャーの説明	CICS でのフィーチャーの使用
mpMetrics-1.0	Enterprise Java のための MicroProfile Metrics API のサポートを提供します。	このフィーチャーには Java 8 が必要です。

表 74. 操作に関する Liberty のサポート対象フィーチャー

Liberty フィーチャー	Liberty フィーチャーの説明	CICS でのフィーチャーの使用
batchManagement-1.0	Java バッチ・コンテナの管理バッチ・サポートを提供します。これには、バッチ REST 管理インターフェース、ジョブ・ロギングのサポート、および外部スケジューラーの統合のためのコマンド・ライン・ユーティリティーが含まれます。	
distributedMap-1.0	DistributedMap API を介してアクセスできるローカル・キャッシュ・サービスを提供します。	
localConnector-1.0	JVM にビルドされたローカル JMX コネクタを使用して、サーバー内の JMX リソースにアクセスできるようにします。	780 ページの『Java Management Extensions API (JMX)』
monitor-1.0	JMX クライアントを使用した Liberty のランタイム・コンポーネントのパフォーマンス・モニターを可能にします。	780 ページの『Java Management Extensions API (JMX)』
osgiConsole-1.0	ランタイムのデバッグを支援する OSGi コンソールを使用できるようにします。	Troubleshooting Java applications
restConnector-1.0	REST ベースのコネクタを介した JMX クライアントによるリモート・アクセスを可能にします。SSL およびユーザー・セキュリティ構成が必要です。	780 ページの『Java Management Extensions API (JMX)』
sessionDatabase-1.0	JDBC を使用するデータ・ソースへの HTTP セッションのパーシスタンスを有効にします。	
webCache-1.0	Web 応答のローカル・キャッシングを可能にします。これには distributedMap フィーチャーが含まれ、応答時間とスループットを改善するために、Web アプリケーション応答の自動キャッシングを行います。	

表 74. 操作に関する Liberty のサポート対象フィーチャー (続き)

Liberty フィーチャー	Liberty フィーチャーの説明	CICS でのフィーチャーの使用
	IBM MQ でホストされるメッセージ・キューへの JMS 2.0 API を介したアクセスをアプリケーションに提供します。	<p><b>制約事項:</b> JMS アプリケーションがクライアント・モードの転送を使用して IBM MQ に接続する場合にのみサポートされます。Liberty 用の IBM MQ Resource Adapter V9.0.1 が必要です。</p> <p><b>重要:</b> この制約事項は、CICS 標準モードの Liberty にも適用されます。</p>

表 75. セキュリティーに関する Liberty のサポート対象フィーチャー

Liberty フィーチャー	Liberty フィーチャーの説明	CICS でのフィーチャーの使用
appSecurity-1.0	サーバー・ランタイム環境とアプリケーションを保護するためのサポートを提供します。appSecurity-2.0 は appSecurity-1.0 を置き換えるものです。	Liberty JVM サーバーのセキュリティーの構成 (Configuring security for a Liberty JVM server)
appSecurity-2.0	サーバー・ランタイム環境とアプリケーションを保護するためのサポートを提供します。appSecurity-2.0 は appSecurity-1.0 を置き換えるものです。	Liberty JVM サーバーのセキュリティーの構成 (Configuring security for a Liberty JVM server)
jwt-1.0	ランタイムで JWT トークンを作成できます。	
ldapRegistry-3.0	LDAP サーバーをユーザー・レジストリーとして使用するためのサポートが有効になります。LDAP パージョン 3.0 をサポートする任意のサーバーを使用できます。複数の LDAP レジストリーを構成してから統合して、単一の論理レジストリー・ビューを実現できます。	分散 ID マッピングを使用した Liberty JVM サーバーのセキュリティーの構成
oauth-2.0	Web アプリケーションが、ユーザーの認証および許可のために OAuth 2.0 を統合できるようになります。	<p>OAuth 2.0 を使用した許可</p> <p>パーシスタント OAuth 2.0 サービスの構成</p>
openidConnectClient-1.0	Web アプリケーションは、構成されているユーザー・レジストリーの代わりまたはユーザー・レジストリーに追加して、ユーザーを認証するために OpenID Connect クライアント 1.0 を統合できます。	
openidConnectServer-1.0	Web アプリケーションは、構成されているユーザー・レジストリーの代わりまたはユーザー・レジストリーに追加して、ユーザーを認証するために OpenID Connect サーバー 1.0 を統合できます。	

表 75. セキュリティーに関する Liberty のサポート対象フィーチャー (続き)

Liberty フィーチャー	Liberty フィーチャーの説明	CICS でのフィーチャーの使用
ssl-1.0	Secure Sockets Layer (SSL) 接続および SAF 鍵リングのサポートを提供します。	RACF を使用した Liberty JVM サーバー用の SSL (TLS) の構成  Liberty JVM サーバーでの SSL (TLS) クライアント証明書認証のセットアップ  Java 鍵ストアを使用した Liberty JVM サーバー用の SSL (TLS) の構成

表 76. z/OS に関する Liberty のサポート対象フィーチャー

Liberty フィーチャー	Liberty フィーチャーの説明	CICS でのフィーチャーの使用
	Liberty が z/OS リソース・リカバリー・サービス (RRS)、アプリケーション・サーバーのトランザクション・マネージャー、およびリソース・マネージャー間でトランザクション・アクティビティを同期化し、管理することを可能にします。	制約事項: zosTransaction-1.0 がサポートされるのは、CICS 標準モードの Liberty でバインディング・モードの転送を使用して IBM MQ に接続する JMS アプリケーションのみです。

表 77. Java EE 6 Web Profile 用にサポートされる Liberty フィーチャー

Liberty フィーチャー	Liberty フィーチャーの説明	CICS でのフィーチャーの使用
beanValidation-1.0	JavaBeans を検証するためのアノテーション・ベースのモデルを提供します。	
beanValidation-1.1	JavaBeans を検証するためのアノテーション・ベースのモデルを提供します。	
cdi-1.0	EJB や Managed Bean などのコンポーネントを、JSP や EJB といった他のコンポーネントに注入するためのメカニズムを提供します。	
ejbLite-3.1	EJB 仕様の EJB Lite サブセットに記載されている Enterprise JavaBeans のサポートを有効にします。	制約事項: トランザクション属性「NotSupported」は、Liberty JVM サーバーではサポートされません。
jndi-1.0	Liberty のサーバー構成における、単一の Java Naming and Directory Interface (JNDI) エントリ定義のサポートを提供します。	
jpa-2.0	アプリケーション管理およびコンテナ管理の JPA を使用したアプリケーションのサポートを有効にします。	
jsf-2.0	JavaServer Faces (JSF) フレームワークを使用する Web アプリケーションのサポートを提供します。	

表 77. Java EE 6 Web Profile 用にサポートされる Liberty フィーチャー (続き)

Liberty フィーチャー	Liberty フィーチャーの説明	CICS でのフィーチャーの使用
jsp-2.2	サーブレットおよび JavaServer Pages (JSP) アプリケーションのサポートを有効にします。	752 ページの『Java EE アプリケーションと Liberty アプリケーション』
servlet-3.0	Java サーブレット仕様に書き込まれる HTTP サーブレットのサポートを提供します。	752 ページの『Java EE アプリケーションと Liberty アプリケーション』 制約事項: サーブレット・ログインおよびログアウト API を使用すると、ユーザー ID が CICS タスクに伝搬されません。
webProfile-6.0	Java EE 6 Web プロファイルをサポートするのに必要な Liberty フィーチャーの便利な組み合わせを提供します。	

表 78. Java EE 6 Technologies 用にサポートされる Liberty フィーチャー

Liberty フィーチャー	Liberty フィーチャーの説明	CICS でのフィーチャーの使用
jaxb-2.2	Java クラスと XML 表現をマップするためのサポートを提供します。	
jaxrs-1.1	Liberty での Java API for RESTful Web Services (JAX-RS) のサポートを提供します。	
jaxws-2.2	SOAP Web サービスのサポートを提供します。	
jca-1.6	アプリケーションからエンタープライズ情報システム (EIS) にアクセスするためのリソース・アダプターの構成を有効にします。	785 ページの『Java EE コネクタ・アーキテクチャー (JCA)』 制約事項: JCA API <code>javax.resource.spi.BootstrapContext.createTimer()</code> によって作成されたスレッド内では、JCICS API および DB2 JDBC タイプ 2 の接続機能の使用はサポートされません。代わりに、同じ効果を持つ機能として、同時 API ( <code>javax.enterprise.concurrent.ManagedScheduledExecutorService</code> ) を使用します。
jcaInboundSecurity-1.0	<code>javax.resource.spi.work.SecurityContext</code> 抽象クラスを拡張することで、JCA インバウンド・リソース・アダプターがセキュリティ・コンテキストを渡せるようにします。	
jdbc-4.0	アプリケーションからデータベースにアクセスするためのデータ・ソースの構成を使用可能にします。	注: DB2 JCC ドライバー内の <code>jdbc-4.0</code> 実装と <code>jdbc-4.1</code> 実装を同時に使用することはできません。

表 78. Java EE 6 Technologies 用にサポートされる Liberty フィーチャー (続き)

Liberty フィーチャー	Liberty フィーチャーの説明	CICS でのフィーチャーの使用
jms-1.1	Java Message Service API を使用してメッセージング・システムにアクセスするためのリソース・アダプターの構成を使用可能にします。	778 ページの『Java Message Service (JMS)』
jmsMdb-3.1	JMS メッセージ駆動型 Enterprise JavaBeans の使用を可能にします。MDB によって、Java EE コンポーネント内のメッセージの非同期処理が可能になります。	
mdb-3.1	メッセージ駆動型 Enterprise JavaBeans の使用を可能にします。MDB によって、Java EE コンポーネント内のメッセージの非同期処理が可能になります。	
wasJmsClient-1.1	アプリケーションが JMS API を使用して、Liberty でホストされるメッセージ・キューにアクセスできるようにします。	Java Message Service (JMS)
wasJmsSecurity-1.0	組み込みメッセージング・サーバーがクライアントからのアクセスを認証して許可できるようにします。	Java Message Service (JMS)
wasJmsServer-1.0	サーバー内で組み込みメッセージング・サーバーを使用できるようにします。アプリケーションは、 <b>wasJmsClient</b> フィーチャーを使用して、メッセージを操作できます。	Java Message Service (JMS)

これらのフィーチャーの機能について詳しくは、Liberty の資料 (Liberty の概要) を参照してください。Liberty の制約事項について詳しくは、ランタイム環境での既知の制約事項を参照してください。

以下の表では、Liberty フィーチャーを CICS のサービスの品質に統合するための一連の CICS フィーチャーを示します。Liberty JVM サーバー・モードは、JVM プロファイルに CICS\_WLP\_MODE を指定することによって設定できます。

表 79. CICS Liberty フィーチャー

CICS フィーチャー	CICS Liberty のモード	説明	CICS フィーチャーの使用
cicsts:core-1.0	統合モード	コア CICS フィーチャー、および Java Transaction API (JTA) 1.0 を提供します。	このフィーチャーは、統合モードの CICS Liberty を使用する場合に必要です。 <b>制約事項:</b> このフィーチャーを追加または削除する前に、JVM サーバーを無効にする必要があります。
cicsts:defaultApp-1.0	統合モード・モードおよび標準モード	Liberty サーバーが実行中であり、サーバー構成に関する情報を提供することを確認します。FileViewer サブレットを使用して、JVM プロファイル、JVM サーバー・ログ、Liberty の server.xml、およびメッセージ・ログを参照します。	CICS デフォルト Web アプリケーションの構成
cicsts:distributedIdentity-1.0	統合モード・モードおよび標準モード	配布 ID のマッピングのサポートを提供します。	分散 ID マッピングを使用した Liberty JVM サーバーのセキュリティの構成
cicsts:jcaLocalEci-1.0	統合モード	CICS プログラムを呼び出ための、ローカル用に最適化された JCA ECI リソース・アダプターを提供します。	787 ページの『JCA ローカル ECI リソース・アダプターの使用』 <b>制約事項:</b> このフィーチャーを追加または削除する前に、JVM サーバーを無効にする必要があります。
cicsts:jdbc-1.0	統合モード・モードおよび標準モード	アプリケーションが JDBC を使用してローカル CICS DB2 データベースにアクセスするためのサポートを提供します。このフィーチャーは、jdbc-4.0 および jdbc-4.1 によって置き換えられています。ただし、DriverManager で直接使用する場合を除きます。	データベースへの接続の取得 <b>制約事項:</b> このフィーチャーを追加または削除する前に、JVM サーバーを無効にする必要があります。
cicsts:link-1.0	統合モード	Liberty JVM サーバーで実行される Java EE アプリケーションを、CICS トランザクションの初期プログラムとして、または CICS プログラムから <b>LINK</b> 、 <b>START</b> 、または <b>START CHANNEL</b> コマンドを使用して開始するためのサポートを提供します。	759 ページの『CICS プログラムからの Java EE アプリケーションへのリンク』

表 79. CICS Liberty フィーチャー (続き)

CICS フィーチャー	CICS Liberty のモード	説明	CICS フィーチャーの使用
cicsts:security-1.0	統合モード・モードおよび標準モード	スレッド ID の伝搬を含め、Liberty セキュリティーと CICS セキュリティーの統合を提供します。	Liberty JVM サーバーのセキュリティの構成 (Configuring security for a Liberty JVM server) 制約事項: このフィーチャーを追加または削除する前に、JVM サーバーを無効にする必要があります。
cicsts:standard-1.0	標準モード	ユーザーが、他のプラットフォームのアプリケーションを変更することなく CICS に移植してデプロイできるようにします。標準モードは、Java EE Full Platform を対象に作成され、このプラットフォームに依存するアプリケーションをホスティングするのに理想的ですが、CICS との完全な統合は必要ありません。	CICS 標準モードの Liberty: CICS と完全に統合せずに Java EE 7 Full Platform をサポート
cicsts:zosConnect-1.0	統合モード	z/OS Connect を CICS Liberty JVM サーバーと統合します。	z/OS Connect EE の構成 制約事項: このフィーチャーを追加または削除する前に、JVM サーバーを無効にする必要があります。
cicsts:zosConnect-2.0	統合モード	z/OS Connect を CICS Liberty JVM サーバーと統合します。	z/OS Connect EE の構成 制約事項: このフィーチャーを追加または削除する前に、JVM サーバーを無効にする必要があります。

## Java アプリケーションからのデータへのアクセス

Db2 および VSAM のデータのアクセスと更新を行うことができる Java アプリケーションを作成できます。または、他の言語のプログラムにリンクして、Db2、VSAM、および IMS にアクセスすることができます。

CICS のデータにアクセスするための Java アプリケーションを作成する際に、次のいずれかの手法を使用できます。CICS リカバリー・マネージャーがデータ保全性を維持します。

### リレーショナル・データへのアクセス

次のいずれかの方法を使用して、Db2 のリレーショナル・データにアクセスするための Java アプリケーションを作成できます。

- 構造化照会言語 (SQL) コマンドを使用してデータにアクセスするプログラムにリンクする JCICS **LINK** コマンド。
- 適切なドライバーが使用可能な場合は、Java Data Base Connectivity (JDBC) または Structured Query Language for Java (SQLJ) 呼び出しを使用して、データに直接アクセスします。Db2 に適切な JDBC ドライバーが使用可能です。JDBC および SQLJ アプリケーション・プログラミング・インターフェースの使用について詳しくは、Java プログラムから Db2 データにアクセスするための JDBC および SQLJ の使用を参照してください。
- 基礎のアクセス機構として JDBC または SQLJ を使用する JavaBeans。このような JavaBeans を開発するには、適切な Java 統合開発環境 (IDE) を使用できます。

## DL/I データへのアクセス

IMS の DL/I データにアクセスするには、Java アプリケーションで JCICS **LINK** コマンドを使用して、EXEC DLI コマンドを発行してデータにアクセスする中間プログラムにリンクする必要があります。

## VSAM データへのアクセス

VSAM データにアクセスするには、Java アプリケーションで次のいずれかの方法を使用できます。

- VSAM に直接アクセスする場合は、JCICS ファイル制御クラス。
- CICS ファイル制御コマンドを出してデータにアクセスするプログラムにリンクする JCICS **LINK** コマンド。

---

## Java からの構造化データとの対話

多くの場合、CICS Java プログラムは、当初は他のプログラミング言語用に設計されたデータと対話します。例えば、Java プログラムは、COBOL コピーブックで定義された COMMAREA を使用して COBOL プログラムにリンクしたり、アセンブラ言語 DSECT を使用してデータが定義される VSAM ファイルからレコードを読み取ったりすることができます。

### 構造化データの Java へのインポート

インポーターを使用して、他の言語からの構造化レコード・データとの対話を容易にする Java クラスを生成できます。インポーターは、言語構造ソースに含まれるデータ型をマップし、Java アプリケーションで基礎となるレコード構造内の個々のフィールドを容易に設定および取得できるようにします。

IBM Record Generator for Java または Rational の Java EE コネクター (J2C) ツールを使用すると、データと対話して Java クラスを生成できます。これにより、CICS で Java とその他のプログラム間でデータを受け渡すことができます。

### IBM Record Generator for Java V3.0.0

IBM Record Generator for Java はスタンドアロン・ユーティリティーであり、COBOL コピーブックまたはアセンブラー DSECT のコンパイルから生成される関連データ (ADATA) ファイルに基づいて、Java ヘルパー・クラスを生成します。次

に、Java アプリケーションで、これらの Java ヘルパー・クラスを COBOL 固有またはアセンブラ言語固有のレコード構造間でのデータ・マーシャルに使用できます。

詳しくは、IBM Record Generator for Java V3.0.0を参照してください。





## Rational J2C ツール

Rational J2C ツール、リソース・アダプター、およびファイル・インポーターを使用して、J2C 成果物を作成できます。この成果物を使用すると、CICS などのエンタープライズ情報システムに接続するエンタープライズ・アプリケーションを作成できます。Rational J2C ツールを使用するには、Rational Application Developer for WebSphere Software または IBM Developer for z Systems<sup>®</sup> が必要です。

J2C ツールの CICS/IMS データ・バインディング・ウィザードでは、カスタマイズ可能な Eclipse ベースのウィザードを使用して COBOL、PL/I、または C の各アプリケーション・プログラムのデータ構造にマップする Java クラスを生成します。次に、Java アプリケーションで、これらのヘルパー・クラスを言語固有のレコード構造間でのデータ・マーシャルに使用できます。

詳しくは、エンタープライズ情報システムへの接続 (Rational Application Developer for WebSphere ソフトウェア製品資料)を参照してください。

関連情報:

-  IBM Redbooks: IBM CICS と JVM サーバー: Java アプリケーションの開発とデプロイ
-  IBM Record Generator for Java を使用して COBOL から Java レコードを構築する
-  COBOL Importer の概要 (Rational Application Developer for WebSphere ソフトウェア製品資料)
-  Rational J2C Tools を使用して COBOL から Java レコードを生成する

---

## OSGi JVM サーバーで JZOS Toolkit API を使用する Java アプリケーションの開発

IBM JZOS Toolkit は、パッケージ com.ibm.jzos 内のクラスで構成され、このパッケージは、IBM Java SDK for z/OS に付属して単一の JAR ファイル `ibmjzos.jar` で配布されます。

これらのクラスにより、z/OS 上の Java アプリケーションは、バイト配列フィールドを Java データ型にマッピングする際に、従来の z/OS データ・セットとデータ・ファイルに直接アクセスでき、z/OS システム・サービスとコンバーター・クラスにアクセスできます。

## 始める前に

JZOS Toolkit API をワークステーションにダウンロードしていない場合は、ibmjzos.jar ファイルを z/OS 上の関連するバージョンの IBM Java SDK からワークステーションに転送します。

## 手順

1. ターゲット・プラットフォームを設定します。開発環境で JCICS API を使用する準備を行うには、ローカルで解決できるように Eclipse ターゲット・プラットフォームを設定します。OSGi 開発環境のターゲット・プラットフォーム定義は、ワークスペース内のアプリケーションが作成されるプラグインを定義するために使用されます。CICS Explorer の場合、Eclipse メニューの「ウィンドウ」>「設定」>「プラグイン開発」>「ターゲット・プラットフォーム」を使用します。「追加」をクリックし、提供されているテンプレートから、ご使用のランタイム環境用の CICS TS リリースを選択します。ワークスペースにターゲット・プラットフォームを適用することを忘れないでください。
2. JZOS Toolkit 用の OSGi ラッパー・バンドルを作成します。IBM CICS SDK for Java EE and Liberty プラグインを使用している場合は、「ファイル」>「インポート」>「OSGi バンドルへの Java アーカイブ」を選択して、新しい OSGi バンドル・プロジェクトを作成します。新しく作成されたバンドルにより、Java アプリケーションに必要な使用可能なすべての JZOS Toolkit パッケージ (com.ibm.jzos、com.ibm.jzos.fields、com.ibm.jzos.wlm など) がエクスポートされることを確認します。これにより、これらのパッケージを Eclipse ワークスペース内の他の OSGi プロジェクトによってインポートできるようになります。次に例を示します。

```
Export-Package: com.ibm.jzos,
 com.ibm.jzos.fields
```

3. CICS Java アプリケーションを作成します。
  - a. ウィザードの「ファイル」>「新規」>「その他のプラグイン・プロジェクト (Other Plug-in Project)」を使用して、Eclipse 内で OSGi バンドル・プロジェクトを作成します。
  - b. Java パッケージ com.ibm.cicsdev.jzos.sample を作成し、クラス ZFilePrint を追加します。
  - c. 次のコード例をコピーします。これにより、//INPUT DD が指す MVS データ・セットが開き、出力が CICS 一時記憶域キューに書き込まれます。

```
package com.ibm.cicsdev.jzos.sample;

import com.ibm.jzos.ZFile;
import com.ibm.jzos.ZUtil;
import com.ibm.cics.server.TSQ;

public class ZFilePrint
{
 public static void main(String[] args) throws Exception
 {
 ZFile zFile = new ZFile("//DD:INPUT", "rb,type=record,noseek");
 TSQ tsqQ = new TSQ();
 tsqQ.setName("JZOSTSQ");

 try
 {
 byte[] recBuf = new byte[zFile.getLrecl()];
```

```

 int nRead;
 String encoding = ZUtil.getDefaultPlatformEncoding();

 while ((nRead = zFile.read(recBuf)) >= 0)
 {
 String line = new String(recBuf, 0, nRead, encoding);
 tsqQ.writeString(line);
 }
 }
 finally
 {
 zFile.close();
 }
}
}

```

4. JCICS および JZOS パッケージのバンドル・マニフェストに、以下の Import-Package ステートメントを追加します。JCICS インポートでは、アプリケーションが動作するバージョンの範囲を指定するというベスト・プラクティスに従う必要があります。通常、この範囲は、次の API の互換性を破る変更の直前までになります。JCICS の場合、これはバージョン 2.0.0 であるため、この例では範囲 `com.ibm.cics.server;version="[1.401.0,2.0.0)"` を使用します。これは、JCICS `TSQ.writeString()` メソッドをサポートするために必要な最小レベルです。JZOS パッケージは、バージョン管理されたバンドルから取得されません。バージョンなしで基盤の JAR ファイルからのものがランタイムに表示されるため、バージョンを参照せずに `com.ibm.jzos` をリストできます。これにより、任意の使用可能なバージョン (0.0.0 を含む) が選択可能になります。

```

Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: com.ibm.cicsdev.jzos.sample
Bundle-SymbolicName: com.ibm.cicsdev.jzos.sample
Bundle-Version: 1.0.0
Bundle-RequiredExecutionEnvironment: JavaSE-1.7
Import-Package: com.ibm.cics.server;version="[1.401.0,2.0.0)",
 com.ibm.jzos

```

5. CICS-MainClass: 定義をバンドル・マニフェストに追加して、`com.ibm.cicsdev.jzos.sample.ZFilePrint` クラスの **MainClass** サービスを登録します。これにより、Java クラスを CICS プログラム定義を使用してリンクできます。現時点で、マニフェストの内容は次の例のようになっているはずです。

```

Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: com.ibm.cicsdev.jzos.sample
Bundle-SymbolicName: com.ibm.cicsdev.jzos.sample
Bundle-Version: 1.0.0
Bundle-RequiredExecutionEnvironment: JavaSE-1.7
Import-Package: com.ibm.cics.server;version="[1.401.0,2.0.0)",
 com.ibm.jzos
CICS-MainClass: com.ibm.cicsdev.jzos.sample.ZFilePrint

```

## タスクの結果

これで、アプリケーションをテストする準備ができました。以下のように、CICS バンドル・プロジェクトを使用して、CICS OSGi JVM サーバーにアプリケーションをデプロイできます。

1. Eclipse 内に CICS バンドル・プロジェクトを作成し、「新規 **OSGi** バンドル・プロジェクトを含める (**New OSGi Bundle Project Include**)」メニューを使用して、OSGi バンドル・プロジェクトを追加します。
2. 「**z/OS UNIX** ファイル・システムへのバンドル・プロジェクトのエクスポート」メニューを使用して、バンドル・プロジェクトを zFS にデプロイします。
3. この zFS ロケーションを参照する CICS バンドル定義を作成してインストールします。
4. JVMClass 属性内の CICS-MainClass: com.ibm.cicsdev.jzos.sample.ZFilePrint を指定する CICS プログラム定義を作成してインストールします。
5. アプリケーションを実行する前に、有効な MVS データ・セットを参照する MVS DD を CICS JCL で定義してから、CICS 領域を再始動する必要があります。次に例を示します。

```
//INPUT DD DISP=SHR,DSN=CICS.USER.INPUT
```

6. 3270 コンソールからアプリケーションを実行する必要がある場合は、ステップ 4 で定義したプログラムを参照するトランザクション定義を作成します。

開始時に、Java クラス ZFilePrint は JZOS Toolkit API を使用して定義された MVS データ・セットを読み取り、その内容を JCICS API を使用して CICS 一時記憶域キューに書き込みます。

---

## Java プログラムから **IBM MQ** へのアクセス

CICS で実行される Java プログラムでは、IBM MQ classes for Java または IBM MQ classes for JMS を使用して IBM MQ にアクセスできます。IBM MQ classes for JMS は、CICS で実行される Java アプリケーションから IBM MQ にアクセスする場合の優先インターフェースです。IBM MQ classes for Java は引き続きサポートされますが、新しいアプリケーションでは IBM classes for JMS を使用する必要があります。

CICS が IBM MQ で動作する仕組みの概要については、CICS と IBM MQを参照してください。

IBM MQ classes for Java は、ネイティブ IBM MQ API である Message Queue Interface (MQI) をカプセル化します。これらのクラスでは、IBM MQ の C++ および .NET インターフェースと類似のオブジェクト・モデルが使用されています。また、JMS で使用可能な機能にとどまらず、IBM MQ の完全な機能一式を活用できます。IBM MQ classes for JMS は、メッセージング・システムとして IBM MQ 用の JMS インターフェースを実装しています。

CICS では、以下の 3 つの異なる JVM サーバー環境で IBM MQ クラスへのアクセスがサポートされます。

- CICS 統合モードの Liberty JVM サーバー。この JVM サーバーでは、IBM MQ classes for JMS がサポートされます。また、管理対象 JMS 接続ファクトリーと MDB サポート、および統合 CICS のトランザクションとセキュリティが提供されます。IBM MQ classes for Java はサポートされません。
- CICS 標準モードの Liberty JVM サーバー。この JVM サーバーでは、IBM MQ classes for JMS がサポートされます。また、管理対象 JMS 接続ファクト

リーと MDB サポートが提供されますが、統合 CICS トランザクションは含まれません。IBM MQ classes for Java はサポートされません。

- OSGi JVM サーバー。この JVM サーバーでは、IBM MQ classes for JMS がサポートされます。また、非管理対象 JMS 接続ファクトリーおよび統合 CICS のトランザクションとセキュリティがサポートされます。IBM MQ classes for Java もバインディング・モードでのみサポートされます。

さらに、CICS から IBM MQ に接続するには、以下の 3 つの方法があります。

- MQ クライアント・モード: IBM MQ キュー・マネージャーへの TCP/IP ネットワーク接続
- MQ バインディング・モード: IBM MQ RRS アダプターを使用した、キュー・マネージャーへのローカル・クロスメモリー・インターフェース
- CICS-MQ アダプターおよび MQCONN: CICS-MQ アダプターを使用した、キュー・マネージャーへのローカル・クロスメモリー・インターフェース

表 80 は、どの JVM サーバーでどの IBM MQ クラスをサポートしているか、および使用される接続オプションを示しています。

表 80. Java アプリケーションから IBM MQ にアクセスするための CICS サポートの要約

MQ 接続	CICS 標準モードの Liberty JVM サーバー	CICS 統合モードの Liberty JVM サーバー	OSGi JVM サーバー
クライアント・モード	<ul style="list-style-type: none"> <li>• IBM MQ classes for JMS: JMS 1.1 および JMS 2.0</li> <li>• IBM MQ classes for Java: サポートされません</li> </ul> <p>詳しくは、827 ページの『CICS Liberty JVM サーバーでの IBM MQ classes for JMS の使用』を参照してください。</p>	<ul style="list-style-type: none"> <li>• IBM MQ classes for JMS: JMS 1.1 および JMS 2.0</li> <li>• IBM MQ classes for Java: サポートされません</li> </ul> <p>詳しくは、827 ページの『CICS Liberty JVM サーバーでの IBM MQ classes for JMS の使用』を参照してください。</p>	サポートされていない
バインディング・モード	<ul style="list-style-type: none"> <li>• IBM MQ classes for JMS: JMS 1.1 および JMS 2.0</li> <li>• IBM MQ classes for Java: サポートされません</li> </ul> <p>詳しくは、827 ページの『CICS Liberty JVM サーバーでの IBM MQ classes for JMS の使用』を参照してください。</p>	サポートされていない	<ul style="list-style-type: none"> <li>• IBM MQ classes for JMS: サポートされません</li> <li>• IBM MQ classes for Java: サポートされます</li> </ul> <p>詳しくは、837 ページの『OSGi JVM サーバーでの IBM MQ classes for Java の使用』を参照してください。</p>

表 80. Java アプリケーションから IBM MQ にアクセスするための CICS サポートの要約  
(続き)

MQ 接続	CICS 標準モードの Liberty JVM サーバー	CICS 統合モードの Liberty JVM サーバー	OSGi JVM サーバー
CICS-MQ アダプター および MQCONN	適用外	適用外	<ul style="list-style-type: none"> <li>IBM MQ classes for JMS: JMS 1.1 および JMS 2.0</li> <li>IBM MQ classes for Java: サポートされません</li> </ul> <p>詳しくは、832 ページの『OSGi JVM サーバーでの IBM MQ classes for JMS の使用』を参照してください。</p>

## CICS Liberty JVM サーバーでの IBM MQ classes for JMS の使用

CICS Liberty JVM サーバーで実行される Java プログラムでは、JMS を使用して IBM MQ にアクセスできます。IBM MQ JMS 機能が CICS Liberty JVM サーバーにインストールされている場合、JMS 要求は MQ メッセージング・プロバイダーによって処理されます。JMS 2.0 機能のサポートにより、従来の (JMS 1.1) インターフェースおよび単純化された (JMS 2.0) インターフェースへのアクセスが提供されます。CICS は、適切なレベルの JMS をサポートし、適切なバージョンの IBM MQ classes for JMS を使用しているレベルの IBM MQ キュー・マネージャーに接続する必要があります。

概要については、仕組み: IBM MQ classes for JMSを参照してください。IBM MQ での JMS の実装方法については、IBM MQ 資料のIBM MQ classes for JMS の使用を参照してください (IBM MQ classes for JMS JavaDocや、メッセージ、アプリケーション機能、および MQ 機能についてはIBM MQ classes for JMS アプリケーションの作成など)。JMS 仕様のレベルを比較するには、Java Message Service Specificationを参照してください。

CICS Liberty 環境では、IBM MQ メッセージング・プロバイダーにより、IBM MQ キュー・マネージャーに対して作成された JMS 接続が以下のようにサポートされます。

- CICS 統合モードの Liberty JVM サーバーでは、JMS アプリケーションが MQ クライアント・モードの転送を使用してキュー・マネージャーに接続できます。MQ バインディング・モードの使用はサポートされません。このタイプの CICS Liberty JVM サーバーでは、統合 CICS のトランザクションとセキュリティーとともに JMS サポートが提供されます。
- CICS 標準モードの Liberty JVM サーバーでは、JMS アプリケーションは、MQ バインディング・モードまたはクライアント・モードのいずれかの転送を使用してキュー・マネージャーに接続できます。このタイプの CICS Liberty JVM サーバーでは、統合 CICS トランザクションを含まない JMS サポートが提供されます。

Java アプリケーションでは、以下の 2 つの方法のいずれかを使用して IBM MQ と通信します。

- メッセージ駆動型 Bean (MDB)
- JMS 接続ファクトリーを使用するサブルーティ

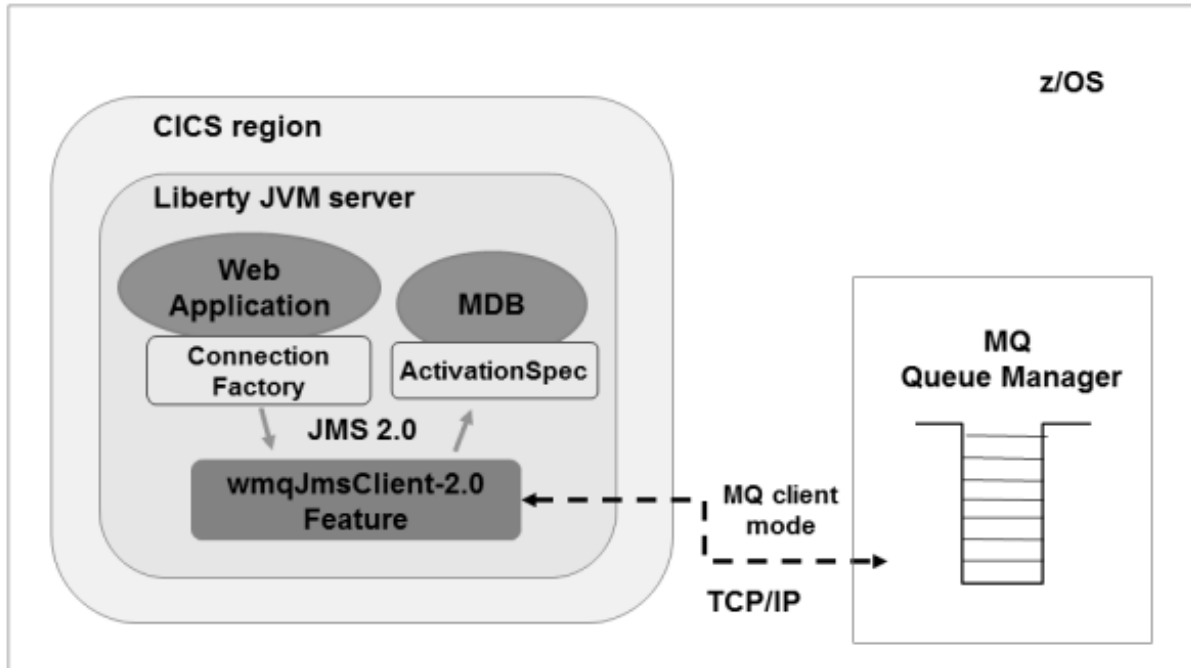


図 138. JMS を使用して IBM MQ に接続し、CICS Liberty JVM サーバーで実行するアプリケーション： この図は、CICS Liberty JVM サーバーから IBM MQ と対話するために使用可能な 2 つのタイプのプログラミングを示しています。示されている CICS Liberty JVM サーバーは、CICS 標準モードまたは CICS 統合モードのいずれかです。示されている MQ クライアント・モード接続に加えて、CICS 標準モードの Liberty JVM サーバーではバインディング・モードでも接続できます。

### 確認事項

- MQ への意図している接続が CICS Liberty JVM サーバーのご使用のバージョンでサポートされている。 825 ページの『Java プログラムから IBM MQ へのアクセス』の 826 ページの表 80を参照してください。
- CICS の接続先である IBM キュー・マネージャーによってサポートされる JMS のレベル。IBM MQ for z/OS バージョン 7.1 では、JMS 1.1 のみがサポートされます。IBM MQ バージョン 8.0 以上では、JMS 1.1 と JMS 2.0 の両方がサポートされます。
- バインディング・モードの転送 (CICS 標準モードの Liberty でのみサポートされます) を使用する場合は、以下のようになります。
  - バインディング・モードを使用して MQ キュー・マネージャーに接続する JMS アプリケーションでは、同じ CICS 領域にインストールされたすべての CICS MQCONN リソースで指定されているキュー・マネージャーとは異なるキュー・マネージャーを指定する必要があります。
  - Liberty と IBM MQ の両方が同じサーバー上にデプロイされます。

- CICSExecutorService を使用して開始されたすべての CICS タスクは、クライアント・モードの転送を使用してキュー・マネージャーに接続する必要があります。
- プログラミングに関していくつかの制約事項があり、これについては 831 ページの『JMS プログラミングに関する考慮事項 (Liberty JVM サーバー)』で説明されています。

## 次に行うこと

アプリケーションで JMS を使用するには、以下を実行する必要があります。

- 開発環境で、IBM MQ 製品のコンポーネントまたは FixCentral からの JAR ファイルとして、IBM MQ classes for JMS にアクセスできることを確認します (これを行う方法については、IBM MQ classes for JMS の使用を参照してください)。
- 管理対象 JMS 接続ファクトリーまたはメッセージ駆動型 Bean (MDB) のいずれかを使用するアプリケーションを開発します。詳しくは、『Liberty JVM サーバーでの IBM MQ classes for JMS を使用したプログラミング』を参照してください。
- アプリケーションを CICS バンドル・プロジェクトに追加し、zFS にエクスポートして、Liberty JVM サーバーにインストールします。
- CICS Liberty JVM サーバー環境を構成します。server.xml の構成に加えて、Liberty から IBM MQ に接続するために必要な IBM MQ リソース・アダプターをセットアップする必要があります。詳しくは、JMS をサポートするための Liberty JVM サーバーの構成を参照してください。

## Liberty JVM サーバーでの IBM MQ classes for JMS を使用したプログラミング

JMS を CICS Java アプリケーションで使用する IBM MQ とメッセージを交換する場合、2 つのオプションがあります。IBM MQ キュー・マネージャーからの着信メッセージを受信するメッセージ駆動型 Bean (MDB)、または JMS 接続ファクトリーを使用して JMS メッセージを送受信するサーブレットのいずれかを使用できます。

### JMS 接続ファクトリーの使用

このタイプのアプリケーションの開発方法を示すチュートリアルについては、CICS Developer Center: CICS Liberty 用の MQ JMS アプリケーションの開発を参照してください。このチュートリアルには、ダウンロード可能なサンプル・サーブレットとサポート・コードへのリンクが含まれています。

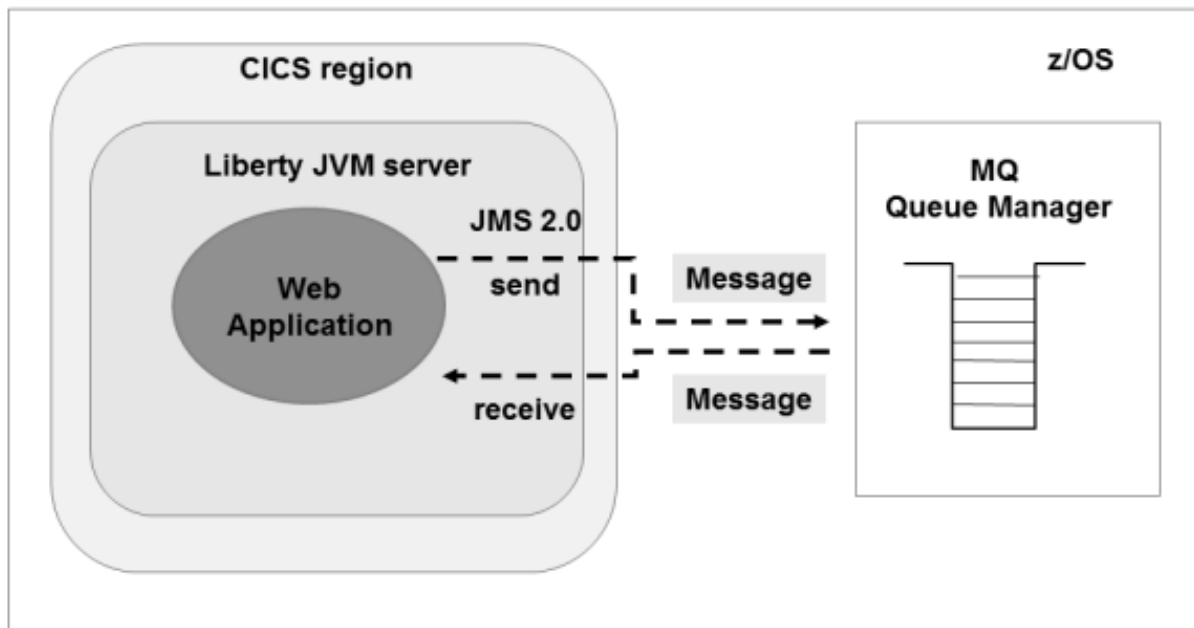


図 139. JMS 接続ファクトリーを使用する Java アプリケーションを介した IBM MQ へのアクセス

### メッセージ駆動型 Bean (MDB) の使用

この形式のプログラミングでは、MDB に関連付けられているキューにメッセージが着信すると、MDB の `onMessage()` メソッドが呼び出されます。次に、`javax.jms.Message` オブジェクトが、さらに処理を行うために MDB への入力として渡されます。MDB は EJB のタイプであるため、コンテナ管理または Bean 管理のいずれかの Java トランザクションを使用できます。

CICS 開発者センターの CICS Developer Center: CICS Liberty 用の MQ JMS アプリケーションの開発は、この種のアプリケーションの開発方法のチュートリアルです。このチュートリアルには、ダウンロード可能なサンプル・サーブレットとサポート・コードへのリンクが含まれています。

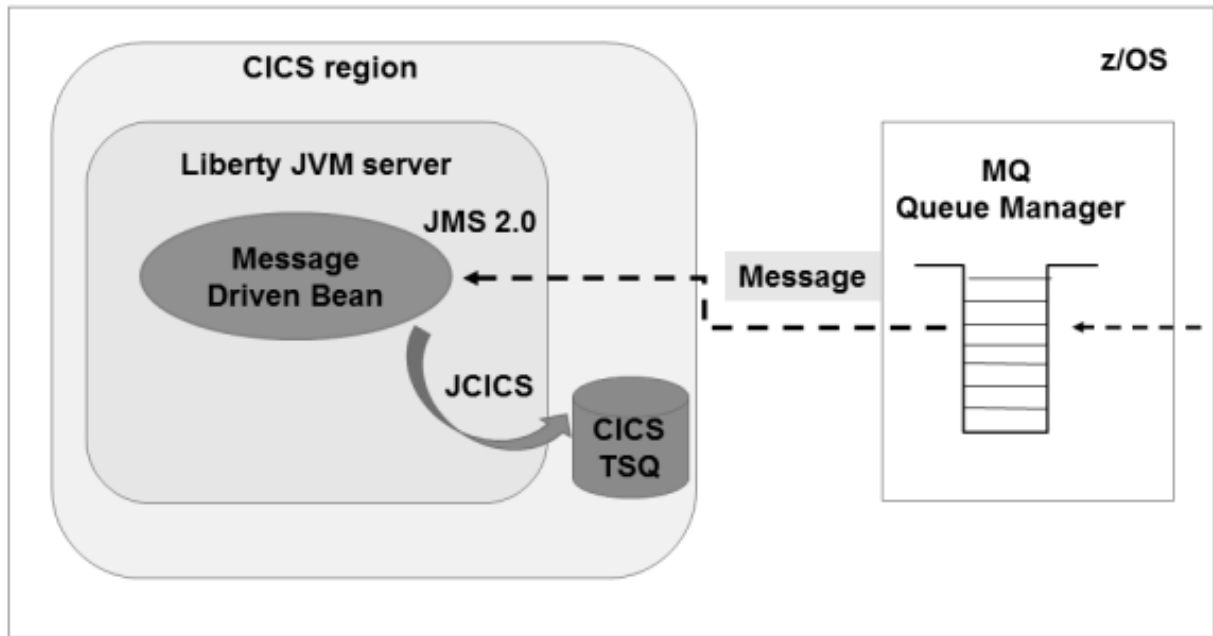


図 140. MDB を使用する Java アプリケーションを介した IBM MQ へのアクセス

### JMS プログラミングに関する考慮事項 (Liberty JVM サーバー)

- `runAsCICS()` メソッドを使用して `CICSExecutorService` に実行依頼される処理には、JMS 要求が含まれていないこと。
- MDB 要求が実行される CICS トランザクション ID のデフォルトは `CJSU` (JVM サーバーの分類されていない要求プロセッサ) です。これは、システム・プロパティ `com.ibm.cics.jvmserver.unclassified.tranid` を使用して、JVM サーバーごとに変更できます。
- Liberty JVM サーバーで JMS を使用する場合、IBM MQ classes for JMS で送受信されるメッセージは、Liberty トランザクション・マネージャーを使用して調整されます。CICS によって管理されるリカバリー可能リソースへの更新を同じ作業単位で調整するには、アプリケーションで、Java Transaction API (JTA) を `UserTransaction.begin()` メソッドによって明示的に、または EJB コンテナ管理トランザクションによって暗黙的に使用する必要があります。UOW を完了するには、`UserTransaction.commit()` メソッドまたは `rollback()` メソッドを使用します。UOW をコミットまたはロールバックするために、以下のオブジェクトで `EXEC CICS SYNCPOINT` コマンド (混合言語アプリケーション) または `commit()` メソッドと `rollback()` メソッドを使用することは、サポートされません。
  - `javax.jms.Session` (JMS 1.1 API)
  - `javax.jms.JmsContext` (JMS 2.0 API)
  - `com.ibm.cics.server.Task`

JTA については詳しくは、Java Transaction API (JTA)を参照してください。

## OSGi JVM サーバーでの IBM MQ classes for JMS の使用

OSGi JVM サーバーで実行される Java プログラムは、JMS を使用して IBM MQ にアクセスできます。CICS Java アプリケーションが JMS 要求を行うと、その要求は MQ メッセージング・プロバイダーによって処理されます。クラシック (JMS 1.1) インターフェースおよび単純化 (JMS 2.0) インターフェースの使用がサポートされています。ただし、CICS が、適切なレベルの JMS をサポートできるレベルの IBM MQ キュー・マネージャーに接続されており、適切なバージョンの IBM MQ classes for JMS を使用していることが条件です。

概要については、仕組み: IBM MQ classes for JMSを参照してください。IBM MQ での JMS の実装方法については、IBM MQ 資料のIBM MQ classes for JMS の使用を参照してください (IBM MQ classes for JMS JavaDocや、メッセージ、アプリケーション機能、および MQ 機能についてはIBM MQ classes for JMS アプリケーションの作成など)。JMS 仕様のレベルを比較するには、Java Message Service Specificationを参照してください。

CICS 環境では、IBM MQ classes for JMS を使用すると、OSGi JVM サーバーを介して接続を作成できます。これにより、非管理対象 JMS 接続ファクトリーおよび統合 CICS のトランザクションとセキュリティがサポートされます。管理対象の JMS 接続ファクトリーまたは MDB をアプリケーションで使用する場合は、代わりに CICS Liberty JVM サーバーを使用します。

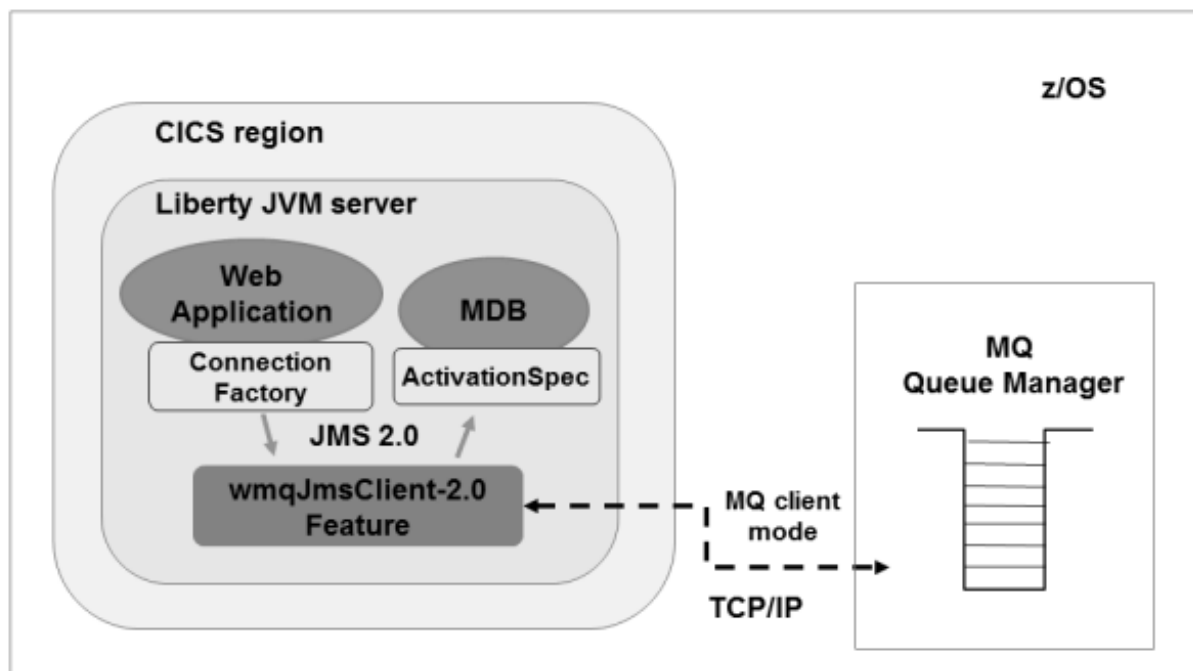


図 141. JMS を使用して IBM MQ に接続し、CICS OSGi サーバーで実行するアプリケーション

### 確認事項

- IBM MQ への接続。OSGi JVM サーバーでは、ローカル・キュー・マネージャーへのバインディング・モードの接続のみがサポートされます。

- CICS の接続先である IBM キュー・マネージャーによってサポートされる JMS のレベル。IBM MQ for z/OS バージョン 7.1 では、JMS 1.1 のみがサポートされます。IBM MQ バージョン 8.0 以上では、JMS 1.1 と JMS 2.0 の両方がサポートされます。
- CICS MQCONN リソースを定義した。
- プログラミングに関していくつかの制約事項があり、これについては『OSGi JVM サーバーでの IBM MQ classes for JMS を使用したプログラミング』で説明されています。

## 次に行うこと

アプリケーションで JMS を使用するには、以下を実行する必要があります。

- 開発環境で、IBM MQ 製品のコンポーネントまたは FixCentral からの JAR ファイルとして、IBM MQ classes for JMS にアクセスできることを確認します (これを行う方法については、IBM MQ 資料の IBM MQ classes for JMS の使用を参照してください)。
- 非管理対象 JMS 接続ファクトリーを使用するアプリケーションを開発します。詳しくは、『OSGi JVM サーバーでの IBM MQ classes for JMS を使用したプログラミング』を参照してください。
- アプリケーションを CICS バンドル・プロジェクトに追加し、zFS にエクスポートして、OSGi JVM サーバーにインストールします。
- IBM MQ に接続する CICS-MQ アダプターを構成します。詳しくは、CICS-MQ アダプターのセットアップを参照してください。
- CICS OSGi サーバー環境を構成します。詳しくは、JMS をサポートするための OSGi JVM サーバーの構成を参照してください。

## OSGi JVM サーバーでの IBM MQ classes for JMS を使用したプログラミング

JMS を CICS Java アプリケーションで使用して IBM MQ とメッセージを交換するには、JMS 接続ファクトリーを使用します。

このタイプのアプリケーションの開発方法を示すチュートリアルについては、CICS Developer Center: CICS OSGi JVM サーバーでの MQ JMS の使用を参照してください。

### JMS プログラミングに関する考慮事項 (OSGi JVM サーバー)

- XA 接続ファクトリー (例えば `com.ibm.mq.jms.MQXAConnectionFactory`) の使用はサポートされていません。
- JVM サーバー環境で IBM® MQ classes for JMS によって送受信されるメッセージは常に、現行スレッドでアクティブな CICS® の作業単位 (UOW) と関連付けられます。その UOW は、`com.ibm.cics.server.Task` オブジェクトでコミット・メソッドまたはロールバック・メソッドを呼び出すことによってのみ完了できます。また、CICS が正常に終了する場合にも完了しますが、その場合 UOW は暗黙的にコミットされます。(これには 1 つの例外があり、このリストの次の項目で説明されています。) `Connection.createSession` または `ConnectionFactory.createContext` のいずれかのメソッドを呼び出す際に、

transacted 引数と acknowledgeMode 引数の値は無視されます。また、以下のメソッドはサポートされておらず、これら呼び出すと、セッション・ケースで `IllegalStateException` になります。

- `javax.jms.Session.commit()`
- `javax.jms.Session.recover()`
- `javax.jms.Session.rollback()`

以下は、JMS コンテキスト・ケースで `IllegalStateRuntimeSession` になります。

- `javax.jms.JMSContext.commit()`
- `javax.jms.JMSContext.recover()`
- `javax.jms.JMSContext.rollback()`
- 上記のトランザクション性に対する例外は、セッションまたは JMS コンテキストが以下に示すいずれかのメカニズムを使用して作成された場合、以下のようになります。

- `Connection.createSession(false, Session.AUTO_ACKNOWLEDGE)`
- `Connection.createSession(Session.AUTO_ACKNOWLEDGE)`
- `ConnectionFactory.createContext(JMSContext.AUTO_ACKNOWLEDGE)`

このセッションの動作、または JMS コンテキストは、以下のようになります。

- 送信されるメッセージはすべて、CICS UOW の外で転送されます。つまり、ターゲット宛先で即時に、または指定された送達遅延間隔が完了したときに使用可能になります。
- セッションまたは JMS コンテキストを作成した接続ファクトリーで `syncPointAllGets` プロパティが指定されていない場合、非永続メッセージは CICS UOW の外で受信されます。
- 永続メッセージは、常に CICS UOW 内で受信されます。

混合言語アプリケーションの場合、非 Java™ プログラムから発行された EXEC CICS SYNCPOINT コマンドは、Java プログラムによる IBM MQ の更新を含め、作業単位全体をコミットします。

- JMS では、`javax.jms.MessageListener`、`javax.jms.ExceptionListener`、JMS 2 を使用する場合には `javax.jms.CompletionListener` など、多数のさまざまなリスナー・インターフェースがサポートされます。これらのどのインターフェースを使用する場合も、MQ JMS では、CICS 環境でサポートされない複数のスレッドが使用されます。これらのうちいずれかのリスナーを登録しようとすると、`JMSEException` または `JMSRuntimeException` が発生します。
- MQ JMS は、CICS での IBM MQ に対するネイティブ・サポートを基盤として、IBM MQ セキュリティー・サポートが利用されます。このセキュリティ・サポートについては、ここを参照してください。その結果、ユーザー ID またはパスワードを指定しているときに接続または JMS コンテキスト・オブジェクトのいずれかを作成しようとすると、`JMSEException` または `JMSRuntimeException` が発生します。
- CICS MQCONN リソースを定義する必要があります。MQ JMS が接続するキュー・マネージャーまたはキュー共用グループの名前は、この MQCONN 定義から取得されます。キュー・マネージャーまたはキュー共用グループをプログラムで指定しようとしても、無効になります。

- 以下の方式で、接続ファクトリーおよび宛先の IBM MQ 実装を作成して構成することができます。
  - JNDI を使用した管理対象オブジェクトの取得
  - IBM JMS 拡張機能の使用
  - IBM MQ JMS 拡張機能の使用

たいていの JMS ユーザーは、JNDI リポジトリを使用して、事前構成された一式の接続ファクトリーおよび宛先を見つけます。CICS では、JNDI 実装は提供されていません。また、LDAP は OSGi 環境では使用できません。

使用可能なオプション、およびバンドル・アダプターの start メソッドを使用して OSGi に初期コンテキスト・ファクトリーと IBM MQ オブジェクト・ファクトリーを登録する方法の例について詳しくは、接続ファクトリーおよび宛先の作成および構成を参照してください。

CICS と IBM MQ キュー・マネージャーの間の接続は、CICS アドレス・スペースのユーザー ID を使用して管理されます。キューへのリソース・アクセスは、トランザクション・ユーザー ID によって許可されます。そのため、接続ファクトリーでユーザー ID およびパスワードを指定することはサポートされていません。

- CICS で MQ JMS を使用するすべてのアプリケーションでは、アプリケーションが実行されるたびに必ず、すべての JMS リソースを MQConnectionFactory から再作成する必要があります。つまり、セッション・インスタンス、メッセージ・コンシューマー、またはその他すべての MQ JMS オブジェクトを、アプリケーションの実行間で共用できるようにアプリケーション・コードの静的変数に格納することはできません。この制約事項が存在するのは、CICS-MQ アダプターにより、キュー入力ハンドルなどのすべてのリソースが、これらを作成したトランザクションの完了時にタイディアップされるためです。これらのリソースのいずれかを同じトランザクションの別の実行または別のトランザクションで使用しようとすると、JMS 例外が発生します。
- JMS 仕様の観点から、IBM MQ classes for JMS では、JVM サーバーを、進行中の JTA トランザクションが常に存在する Java™ EE 準拠のアプリケーション・サーバーとして扱います。例えば、CICS では `javax.jms.Session.commit()` を呼び出すことができません。JMS 仕様では、JTA トランザクションが進行中の間は、JEE EJB または Web コンテナでこれと呼び出せないことになっているためです。その結果、CICS での JMS API には制約事項が存在します。

従来の JMS API (JMS 1.1) には、以下の制約事項が適用されます。

- `javax.jms.Connection.createConnectionConsumer(javax.jms.Destination, String, javax.jms.ServerSessionPool, int)` は常に `JMSEException` をスローします。
- `javax.jms.Connection.createDurableConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int)` は常に `JMSEException` をスローします。
- 接続の既存のセッションがアクティブの場合、`javax.jms.Connection.createSession` の 3 つのバリエーションすべては常に `JMSEException` をスローします。

- `javax.jms.Connection.createSharedConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int)` は常に `JMSEException` をスローします。
- `javax.jms.Connection.createSharedDurableConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int)` は常に `JMSEException` をスローします。
- `javax.jms.Connection.setClientID()` は常に `JMSEException` をスローします。
- `javax.jms.Connection.setExceptionListener(javax.jms.ExceptionListener)` は常に `JMSEException` をスローします。
- `javax.jms.Connection.stop()` は常に `JMSEException` をスローします。
- `javax.jms.MessageConsumer.setMessageListener(javax.jms.MessageListener)` は常に `JMSEException` をスローします。
- `javax.jms.MessageConsumer.getMessageListener()` は常に `JMSEException` をスローします。
- `javax.jms.MessageProducer.send(javax.jms.Destination, javax.jms.Message, javax.jms.CompletionListener)` は常に `JMSEException` をスローします。
- `javax.jms.MessageProducer.send(javax.jms.Destination, javax.jms.Message, int, int, long, javax.jms.CompletionListener)` は常に `JMSEException` をスローします。
- `javax.jms.MessageProducer.send(javax.jms.Message, int, int, long, javax.jms.CompletionListener)` は常に `JMSEException` をスローします。
- `javax.jms.MessageProducer.send(javax.jms.Message, javax.jms.CompletionListener)` は常に `JMSEException` をスローします。
- `javax.jms.Session.run()` は常に `JMSRuntimeException` をスローします。
- `javax.jms.Session.setMessageListener(javax.jms.MessageListener)` は常に `JMSEException` をスローします。
- `javax.jms.Session.getMessageListener()` は常に `JMSEException` をスローします。

単純化された JMS API (JMS 2.0) には、以下の制約事項が適用されます。

- - `javax.jms.JMSContext.createContext(int)` は常に `JMSRuntimeException` をスローします。
  - `javax.jms.JMSContext.setClientID(String)` は常に `JMSRuntimeException` をスローします。
  - `javax.jms.JMSContext.setExceptionListener(javax.jms.ExceptionListener)` は常に `JMSRuntimeException` をスローします。
  - `javax.jms.JMSContext.stop()` は常に `JMSRuntimeException` をスローします。
  - `javax.jms.JMSProducer.setAsync(javax.jms.CompletionListener)` は常に `JMSRuntimeException` をスローします。
  - `javax.jms.JMSConsumer.getMessageListener()` は常に `JMSRuntimeException` をスローします。

- `javax.jms.JMSCConsumer.setMessageListener(javax.jms.MessageListener)` は常に `JMSRuntimeException` をスローします。

### JMS 要求を処理中の CICS の異常終了

IBM MQ classes for JMS とバインディング・モードの転送を使用すると、IBM MQ MQI コマンドが発行されます。MQI コマンドの処理中に発生した CICS の異常終了は、Java 例外に変換されないため、CICS Java アプリケーションによってキャッチされません。

この状況では、CICS トランザクションが異常終了し、最後の同期点までロールバックされます。

## OSGi JVM サーバーでの IBM MQ classes for Java の使用

OSGi JVM サーバーで実行される Java プログラムでは、IBM MQ によって提供される IBM MQ classes for Java を使用して、IBM MQ にアクセスできます。IBM MQ classes for Java では、Message Queue Interface (MQI) の Java バリエーションが提供されます。このバリエーションを使用すると、CICS アプリケーションで CICS によって保持される MQ 接続を使用して、キューに対してメッセージの書き込みと取得を行うことができます。CICS アプリケーションでの IBM MQ classes for Java に対するサポートは、IBM MQ for z/OS 7.1 から提供されます。

CICS 環境では、IBM MQ が提供するクラスは、バインディング・モードでの MQ への接続のみを許可します。クライアント・モードでリモート・キュー・マネージャーへの接続を使用しようとする、例外が発生します。バインディング・モードでは、呼び出し要求が IBM MQ の MQI 呼び出しに変換され、既存の CICS-MQ アダプターによって通常どおりに処理されます。変換された要求は、他のプログラム (COBOL プログラムなど) からの MQI 要求とまったく同じ方式で CICS-MQ アダプターに到着します。そのため、IBM MQ にアクセスする Java プログラムと他のプログラムの間に、動作の違いはありません。

ご使用のアプリケーションで IBM MQ classes for Java を使用するには、以下を実行する必要があります。

- 開発環境で、MQ classes for Java にアクセスできることを確認します。ご使用のワークステーションに IBM MQ がインストールされていない場合は、IBM MQSupportPacs から入手してください。ここでは、IBM MQ クライアントを無料でダウンロードする資格があります。
- アプリケーションを CICS バンドル・プロジェクトに追加し、zFS にエクスポートして、JVM サーバーにインストールします。
- 適切なレベルの IBM MQ Java およびネイティブ・ライブラリーを使用して、CICS JVM サーバー環境を構成します。これらは、CICS STEPLIB に指定されている IBM MQ ライブラリーのレベルと一致する必要があります。詳細については、IBM MQ classes for Java をサポートするための OSGi JVM サーバーの構成を参照してください。

IBM MQ classes for Java については、IBM MQ 資料の IBM MQ classes for Java の使用を参照してください。クラスのリストは、IBM MQ の資料の IBM MQ

classes for JMS JavaDoc にあります。チュートリアルについては、CICS Developer Center: CICS OSGi JVM サーバーでの MQ JMS の使用を参照してください。

## WebSphere MQ 要求に関わる作業単位のコミット

CICS JVM サーバー環境で IBM MQ classes for Java によって送受信されるメッセージには、常に CICS 作業単位 (UOW) が関連付けられます。

その UOW を完了するには、`com.ibm.cics.server.Task` オブジェクトのコミット・メソッドまたはロールバック・メソッドを呼び出すしかありません。あるいは、CICS タスクが正常に終了すれば、UOW は暗黙的にコミットされます。MQQueueManager に対するトランザクション制御メソッドの使用は、サポートされません。

混合言語アプリケーションの場合、非 Java プログラムから発行された **EXEC CICS SYNCPOINT** コマンドは、Java プログラムによる IBM MQ の更新を含め、作業単位全体をコミットします。

## IBM MQ 要求を処理中の CICS の異常終了

IBM MQ classes for Java を使用すると、IBM MQ MQI コマンドが発行されます。MQI コマンドの処理中に発生した CICS の異常終了は、Java 例外に変換されないため、CICS Java アプリケーションによってキャッチされません。

この状況では、CICS トランザクションが異常終了し、最後の同期点までロールバックされます。

---

## CICS 内の Java アプリケーションからの接続

CICS 環境内の Java プログラムは、TCP/IP ソケットをオープンし、外部プロセスと通信することができます。Java プログラムをゲートウェイとして使用すると、他の言語の CICS プログラムからは使用できない可能性がある他のエンタープライズ・アプリケーションに接続することができます。例えば、リモート・サブレットまたはデータベースと通信する Java プログラムを作成できます。

この接続が CICS に統合されて、分散トランザクションや ID 伝搬などのエンタープライズ・サービス品質を提供する場合があります。また、CICS によって提供される分散トランザクションやその他のサービスなしに接続を使用できる場合もあります。必要な接続のタイプによっては、CICS で本来はサポートされないエンタープライズ・アプリケーションとの接続を可能にするサード・パーティー・ベンダー製品が使用できる場合があります。

一般に、CICS 環境における JVM の機能は、バッチ・モード JVM とほぼ同じです。バッチ・モード JVM は、CICS 環境の外部ではスタンドアロン・プロセスとして実行され、通常は、UNIX システム・サービスのコマンド行から、または JCL ジョブで開始されます。バッチ・モード JVM で作動可能な大部分のアプリケーションは、同じ範囲で CICS における JVM でも実行できます。例えば、サード・パーティーの JDBC ドライバーを使用して IBM 以外のデータベースと通信するバッチ・モード Java アプリケーションを作成する場合、同じアプリケーションがおそらく、CICS における JVM で作動します。ベンダー提供のコード (IBM 以外の

JDBC ドライバーなど) を CICS における JVM で使用したい場合は、ベンダーに問い合わせ、そのコードが CICS における JVM で実行されることをサポートするかどうかを判別してください。

CICS での Java アプリケーションの動作について詳しくは、703 ページの『CICS での Java ランタイム環境』を参照してください。

CICS 環境における JVM で実行されるバッチ・モード・アプリケーションは、通常、CICS の機能を利用しません。例えば、CICS の Java プログラムが、サード・パーティーの JDBC ドライバーを使用して IBM 以外のデータベース内のレコードを更新する場合、CICS はこのアクティビティを認識せず、現行の CICS トランザクションに更新を組み込もうとしません。

---

## JCA ローカル ECI サポート

JCA ローカル ECI リソース・アダプターを使用するよう構成されている Liberty JVM サーバーに、JCA ECI アプリケーションをデプロイすることができます。このトピックは、CICS 統合モードの Liberty にのみ適用されます。

アプリケーションの開発について詳しくは、785 ページの『Java EE コネクター・アーキテクチャー (JCA)』を参照してください。既存の CICS Transaction Gateway アプリケーションの移植の詳細については、788 ページの『JCA ECI アプリケーションを Liberty JVM サーバーに移植する』を参照してください。JCA の構成については、787 ページの『JCA ローカル ECI リソース・アダプターの構成』を参照してください。

CICS TS JCA ローカル ECI リソース・アダプターに備わっている JCA ECI プログラミング・インターフェースは、クラス定義から生成される Javadoc で記述されています。Javadoc は、JCA ローカル ECI Javadoc 情報から入手できます。

アプリケーション開発に必要なライブラリーと OSGi バンドルは、IBM CICS SDK for Java に備わっています。

---

## JVM サーバーで実行する既存のアプリケーションのパッケージ化

プールされた JVM で Java アプリケーションを実行している場合、JVM サーバーで実行するようにそれらのアプリケーションを移動することができます。JVM サーバーは、同じ JVM 内で Java アプリケーションに対する複数の要求を処理できるので、同じワークロードの実行に必要な JVM 数を減らすことができます。

Java アプリケーションを 1 つ以上の OSGi バンドルとしてパッケージ化する必要があります。アプリケーションのパッケージ化に、3 つの方法の 1 つを使用できます。

### 既存の Java プロジェクトのプラグイン・プロジェクトへの変換

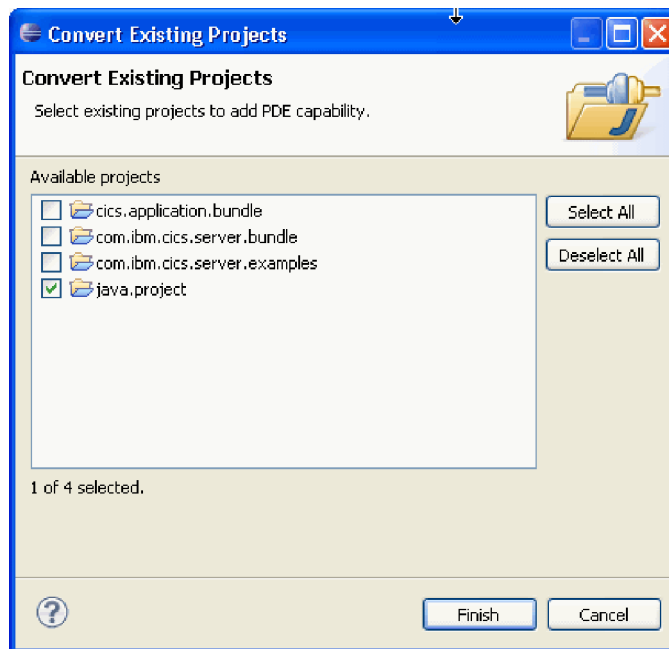
既存の Java プロジェクトがある場合は、OSGi プラグイン・プロジェクトに変換できます。OSGi バンドルは、プールされた JVM 環境および JVM サーバーで実行できます。

## このタスクについて

このタスクは、ワークスペースに既存の Java プロジェクトがあり、OSGi プラグイン・プロジェクトに変換することを想定しています。

### 手順

1. 「パッケージ・エクスプローラー」ビューで、プラグイン・プロジェクトに変換する Java プロジェクトを右クリックして、「構成 (Configure)」 > 「プラグイン・プロジェクトに変換 (Convert to Plug-in Projects)」をクリックします。「既存のプロジェクトを変換 (Convert Existing Projects)」ダイアログが表示されます。



ダイアログには、ワークスペースのすべての Java プロジェクトのリストが含まれます。変換に選んだものが選択されます。選択内容を変更することも、複数の Java プロジェクトを選択してプラグイン・プロジェクトに変換することもできます。

2. 「終了」をクリックします。Java プロジェクトがプラグイン・プロジェクトに変換されます。プロジェクト名は変更されませんが、プロジェクトにマニフェスト・ファイルとビルド・プロパティ・ファイルが含まれるようになりました。
3. 必須: ここでプラグイン・マニフェスト・ファイルを編集して、JCICS API の依存関係を追加する必要があります。これらのステップを実行しない場合、バンドルをエクスポートしてインストールできますが、動作しません。

注: CICS TS バージョン 4.2 より前の CICS では、Java クラス・ライブラリー dfjcics.jar を Java ビルド・パスに追加する必要がありました。CICS TS バージョン 4.2 では、OSGi がユーザーの代わりにビルド・パスを管理します。次のステップを実行する前に、現在のビルド・パスを編集して dfhjcics.jar の参照をすべて削除する必要があります。dfhjcics.jar へ参照をすべて削除しない場合、実行時に NoSuchMethodException エラーが発生します。

- a. 「パッケージ・エクスプローラー」ビューで、プロジェクト名を右クリックして、「プラグイン・ツール」 > 「マニフェストを開く」をクリックします。 マニフェスト・ファイルがマニフェスト・エディターで開きます。
- b. 重要: CICS TS バージョン 4.2 より前の CICS のバージョンでは、JCICS という Java クラス・ライブラリーが dfjcics.jar JAR ファイルの中に提供されています。 CICS TS バージョン 4.2 では、このライブラリーは com.ibm.cics.server.jar ファイル内に提供されています。プロジェクトのマニフェストに宣言 **Import-Package: dfhjcics.jar;** が含まれている場合は、この宣言を削除してから残りのステップを続行する必要があります。
- c. 「依存関係 (Dependencies)」タブを選択して、「インポートされたパッケージ (Imported Packages)」セクションで、「追加」をクリックします。「パッケージの選択 (Package Selection)」ダイアログが開きます。
- d. パッケージ com.ibm.cics.server を選択して、「OK」をクリックします。 パッケージが「インポートされたパッケージ (Imported Packages)」リストに表示されます。
- e. オプション: アプリケーションで必要な場合には、前述のステップを繰り返して、以下のパッケージをインストールします。

#### **com.ibm.record**

VisualAge に付属の Java レコード・フレームワークから IByteBuffer を使用するレガシー・プログラム用の Java API。以前は dfjcics.jar ファイル内にありました。

- f. 「ファイル」 > 「保管」を選択してマニフェスト・ファイルを保管します。

## タスクの結果

既存の Java プロジェクトのプラグイン・プロジェクトへの変換が正常に完了しました。

## 次のタスク

この時点で、マニフェスト・ファイルを更新して CICS-MainClass 宣言を追加する必要があります。詳しくは、関連リンクを参照してください。

## JAR ファイルの内容の OSGi プラグイン・プロジェクトへのインポート

既存の JAR ファイルからプラグイン・プロジェクトを作成できます。この方法は、アプリケーションが既にスレッド・セーフであり、リファクタリングや再コンパイルが必要ない場合に役立ちます。OSGi バンドルは、プールされた JVM 環境および JVM サーバーで実行できます。

### このタスクについて

このタスクでは、既存の JAR ファイルから新しい OSGi プラグイン・プロジェクトを作成します。JAR ファイルは、ローカル・ファイル・システム上に存在している必要があります。

## 手順

1. Eclipse メニュー・バーで、「ファイル」 > 「新規」 > 「プロジェクト (Project)」をクリックして、新規ウィザードを開きます。
2. 「プラグイン開発」フォルダーを展開して、「既存の JAR アーカイブからのプラグイン (Plug-in from Existing JAR Archives)」をクリックします。「次へ」をクリックします。「JAR の選択 (JAR selection)」ダイアログが開きます。
3. 変換する JAR ファイルを見つけます。ファイルが Eclipse ワークスペース内にある場合は、「追加」をクリックします。ファイルがコンピューター上のフォルダーにある場合は、「外部を追加 (Add External)」をクリックして、JAR ファイルを参照します。必要なファイルを選択して「オープン」をクリックし、「JAR の選択 (Jar selection)」ダイアログで追加します。「次へ」をクリックします。「プラグイン・プロジェクトのプロパティ (Plug-in Project Properties)」ダイアログが開きます。

**New Plug-in from Existing JAR Archives**

**Plug-in Project Properties**  
Enter the data required to generate the plug-in.

Project name:

☒ Use default location

Location:

**Plug-in Properties**

Plug-in ID:

Plug-in Version:

Plug-in Name:

Plug-in Provider:

☐ Analyze library contents and add dependencies

Execution Environment:

**Target Platform**

This plug-in is targeted to run with:

☐ Eclipse version:

☒ an OSGi framework:

☒ Unzip the JAR archives into the project

☐ Update references to the JAR files

**Working sets**

☐ Add project to working sets

Working sets:

4. 「プロジェクト名」フィールドで、作成するプロジェクトの名前を入力します。プロジェクト名は必須です。

5. 必要に応じて、「プラグインのプロパティ (Plug-in Properties)」セクションの次のフィールドを入力します。

#### プラグイン ID

プラグイン ID はプロジェクト名から自動的に生成されます。ただし、必要に応じて ID を変更できます。

#### プラグイン名

プラグイン名はプロジェクト名から自動的に生成されます。ただし、必要に応じて名前を変更できます。

#### 実行環境

このフィールドは、プラグインの実行に必要な JRE の最小レベルを指定します。CICS ランタイム・ターゲット・プラットフォームの実行環境に一致する Java のレベルを選択します。

6. 「ターゲット・プラットフォーム」セクションで、「**OSGI フレームワーク (an OSGI framework)**」を選択して、メニューから「標準」を選択します。
7. 「**JAR アーカイブをプロジェクトに unzip する (Unzip the JAR archives into the project)**」が選択されていることを確認して、「終了」をクリックします。Eclipse がプラグイン・プロジェクトをワークスペースに作成します。
8. 必須: ここでプラグイン・マニフェスト・ファイルを編集して、JCICS API の依存関係を追加する必要があります。これらのステップを実行しない場合、バンドルをエクスポートしてインストールできますが、動作しません。
  - a. 「パッケージ・エクスプローラー」ビューで、プロジェクト名を右クリックして、「プラグイン・ツール」 > 「マニフェストを開く」をクリックします。マニフェスト・ファイルがマニフェスト・エディターで開きます。
  - b. 「依存関係 (**Dependencies**)」タブを選択して、「インポートされたパッケージ (**Imported Packages**)」セクションで、「追加」をクリックします。「パッケージの選択 (Package Selection)」ダイアログが開きます。
  - c. パッケージ `com.ibm.cics.server` を選択して、「**OK**」をクリックします。パッケージが「インポートされたパッケージ (**Imported Packages**)」リストに表示されます。
  - d. オプション: アプリケーションで必要な場合には、前述のステップを繰り返して、以下のパッケージをインストールします。

#### `com.ibm.record`

VisualAge に付属の Java レコード・フレームワークから `IByteBuffer` を使用するレガシー・プログラム用の Java API。以前は `dfjcics.jar` ファイル内にありました。

- e. 「ファイル」 > 「保管」を選択してマニフェスト・ファイルを保管します。

## タスクの結果

既存の JAR ファイルから OSGi プラグイン・プロジェクトを作成しました。

## 次のタスク

この時点で、マニフェスト・ファイルを更新して `CICS-MainClass` 宣言を追加する必要があります。詳しくは、関連リンクを参照してください。

## バイナリー JAR ファイルの OSGi プラグイン・プロジェクトへのインポート

既存のバイナリー JAR ファイルからプラグイン・プロジェクトを作成できます。この方式は、ライセンスの制限がある状態、またはバイナリー・ファイルを抽出できない状態で便利です。ただし、JAR ファイルを含む OSGi バンドルは、プールされた JVM 環境ではサポートされません。

### このタスクについて

このタスクでは、既存のバイナリー JAR ファイルから新しい OSGi プラグイン・プロジェクトを作成します。JAR ファイルは、ローカル・ファイル・システム上に存在している必要があります。

### 手順

1. Eclipse メニュー・バーで、「ファイル」 > 「新規」 > 「プロジェクト (Project)」をクリックして、新規ウィザードを開きます。
2. 「プラグイン開発」フォルダーを展開して、「既存の JAR アーカイブからのプラグイン (Plug-in from Existing JAR Archives)」をクリックします。「次へ」をクリックします。「JAR の選択 (JAR selection)」ダイアログが開きます。
3. 変換する JAR ファイルを見つけます。ファイルが Eclipse ワークスペース内にある場合は、「追加」をクリックします。ファイルがコンピューター上のフォルダーにある場合は、「外部を追加 (Add External)」をクリックして、JAR ファイルを参照します。必要なファイルを選択して「オープン」をクリックし、「JAR の選択 (Jar selection)」ダイアログで追加します。「次へ」をクリックします。「プラグイン・プロジェクトのプロパティー (Plug-in Project Properties)」ダイアログが開きます。

4. 「プロジェクト名」フィールドで、作成するプロジェクトの名前を入力します。プロジェクト名は必須です。
5. 必要に応じて、「プラグインのプロパティ (Plug-in Properties)」セクションの次のフィールドを入力します。

#### プラグイン ID

プラグイン ID はプロジェクト名から自動的に生成されます。ただし、必要に応じて ID を変更できます。

#### プラグイン名

プラグイン名はプロジェクト名から自動的に生成されます。ただし、必要に応じて名前を変更できます。

#### 実行環境

このフィールドは、プラグインの実行に必要な JRE の最小レベルを指定します。CICS ランタイム・ターゲット・プラットフォームの実行環境に一致する Java のレベルを選択します。

6. 「ターゲット・プラットフォーム」セクションで、「**OSGI フレームワーク (an OSGI framework)**」を選択して、メニューから「標準」を選択します。
7. 「**JAR アーカイブをプロジェクトに unzip する (Unzip the JAR archives into the project)**」が選択されていないことを確認して、「終了」をクリックします。Eclipse がプラグイン・プロジェクトをワークスペースに作成します。プロジェクトにはバイナリー JAR ファイルが含まれますが、プロジェクトはブールされた JVM 環境ではサポートされません。
8. 必須: ここでプラグイン・マニフェスト・ファイルを編集して、JCICS API の依存関係を追加する必要があります。これらのステップを実行しない場合、バンドルをエクスポートしてインストールできますが、動作しません。
  - a. 「パッケージ・エクスプローラー」ビューで、プロジェクト名を右クリックして、「プラグイン・ツール」 > 「マニフェストを開く」をクリックします。マニフェスト・ファイルがマニフェスト・エディターで開きます。
  - b. 「依存関係 (**Dependencies**)」タブを選択して、「インポートされたパッケージ (**Imported Packages**)」セクションで、「追加」をクリックします。「パッケージの選択 (**Package Selection**)」ダイアログが開きます。
  - c. パッケージ `com.ibm.cics.server` を選択して、「**OK**」をクリックします。パッケージが「インポートされたパッケージ (**Imported Packages**)」リストに表示されます。
  - d. オプション: アプリケーションで必要な場合には、前述のステップを繰り返して、以下のパッケージをインストールします。

#### **com.ibm.record**

VisualAge に付属の Java レコード・フレームワークから  
IBuffer を使用するレガシー・プログラム用の Java API。以前  
は `dfjcics.jar` ファイル内にありました。

- e. 「ファイル」 > 「保管」を選択してマニフェスト・ファイルを保管します。

## タスクの結果

ワークスペースでのプラグイン・プロジェクトの作成が正常に完了しました。

## 次のタスク

この時点で、マニフェスト・ファイルを更新して `CICS-MainClass` 宣言を追加する必要があります。詳しくは、関連リンクを参照してください。

---

## 第 12 章 リカバリーのための開発

リカバリーの計画局面では、ご使用のアプリケーション、システム定義、内部資料、およびテスト計画について考慮する必要があります。

---

### アプリケーション設計上の考慮事項

アプリケーション設計段階における可能な限り早期のリカバリー可能性について考えます。このトピックでは、考慮すべき設計のいくつかの局面について説明します。

#### リカバリー要件に関連する質問

説明を簡単にするために、以下は、単一のアプリケーションについての質問とします。

注: 新しいアプリケーションを既存のシステムに追加する場合は、その追加によるシステム全体への影響を考慮する必要があります。

質問 1: そのアプリケーションは、システム内のデータを更新しますか?

そのアプリケーションが更新を実行しない場合 (つまり、照会専用のアプリケーションの場合)、リカバリー機能および再始動機能が CICS に含まれている必要はありません。 (ただし、それらが読み取り不能になった場合のために、更新されていないデータ・セットのバックアップ・コピーを作成しておく必要があります。) 残りの質問は、そのアプリケーションが更新を行う場合を想定しています。

質問 2: このアプリケーションは、他のオンライン・アプリケーションがアクセスするデータ・セットを更新しますか?

「はい」の場合、その業務では、その更新をオンラインにして、その後すぐに (つまり、そのアプリケーションが更新を行った直後に) 他のアプリケーションが使用できるようにする必要がありますか? これは、他のアプリケーションが使用するインベントリー・データ・セット (データベースなど) を常にできるだけ最新の状態にしておくことが重要な、オンラインの注文入力システムで必要になる可能性があります。

あるいは、更新を一時的に保管して、後で (おそらくはオフラインのバッチ・プログラムを使用して) データ・セットを変更するために使用することができますか? これは、他のアプリケーションがすぐには必要としないデータのみを記録するアプリケーションの場合に、受け入れられる可能性があります。

質問 3: このアプリケーションは、バッチ・アプリケーションがアクセスするデータ・セットを更新しますか?

「はい」の場合は、バッチ・アプリケーションがオンライン・アプリケーションと同時にデータ・セットにアクセスするかどうかを設定します。バツ

---

1. これらの質問のコンテキストでは、「データ・セット」という用語にはデータベースが含まれます。

チ・アプリケーションによって行われるアクセスが読み取り専用に限られていれば、そのデータ・セットをオンライン・アプリケーションとバッチ・アプリケーションで共用することができます。ただし、読み取り保全性は保証されない可能性があります。オンライン・アプリケーションとバッチ・アプリケーションの両方から同時にデータ・セットを更新する場合は、DL/I または Db2 を使用することを検討してください。これらは、読み取りと書き込みの両方の保全性を確保します。

**質問 4:** このアプリケーションは機密データにアクセスしますか？

機密データが含まれるファイル、およびそれらのファイルへのアクセス権を持つアプリケーションは、この段階で明確に識別する必要があります。障害の発生後にサービスを再開するときには、サインオン・メッセージで ID を再度要求することで、確実に許可されているユーザーのみが機密データにアクセスできるようにする必要があります。

**質問 5:** データ・セットが使用不可になった場合、リカバリーを実行する間はすべてのアプリケーションを終了する必要がありますか？

データ・セットのリカバリーが行われている間、いずれかのアプリケーションに対して低下したサービスを保持する必要がある場合は、そのための手順を組み込む必要があります。

**質問 6:** 更新されるファイルのどれが重要とみなされますか？

その業務にとって非常に重要なファイルを、常にリカバリー可能にしておく必要があるものとして識別します。

**質問 7:** 可用性と比較してデータ保全性はどれぐらい重要ですか？

ロックされたレコードをその業務ではどのぐらいの期間待つことができるかを検討します。また、通常の再同期プロセスがオーバーライドされる場合の、データ保全性のリスクと比較した重要度について検討します。

許容できる待ち時間は、データの重要値、および影響があると予測されるユーザーの数によって異なります。非常に価値のあるデータや、稀にしかアクセスされないデータの場合は、価値の低いデータや、業務に不可欠な多数のプロセスからアクセスされるデータの場合よりも、許容可能な待ち時間は長くなります。

**質問 8:** 障害が発生した場合、その業務ではどのぐらいの期間、アプリケーションが使用不能でも許容できますか？

障害発生後のシステムのサービス休止をその業務で許容できる（おおよその）最大時間を示します。それは数分または数時間でしょうか。許容する時間は、障害のタイプと、オンライン・アプリケーションなしで業務を継続できる方法を勘案して決める必要があります。

**質問 9:** 障害の発生後、ユーザーはどのようにデータの入力を続行または再開しますか？

これは、必要になるプログラミングの量に影響するため、リカバリー要件記述の重要な部分です。端末ユーザーの再始動手順は、例えば以下のように、何が実行可能かに大きく左右されます。

- ユーザーは、他の方法（例えば、手動など）で業務を継続できる必要がありますか？

- ユーザーがソース資料 (紙の文書) をまだ保持していて、データの入力 (または再入力) の続行が可能ですか? ソース資料が一時的なもの (例えば、電話で受け取ったものなど) の場合には、より複雑な手順が必要になる可能性があります。
- ユーザーがソース資料をまだ保持している場合でも、データの量がその再入力を不可能にしていますか?

これらの要因によって、ユーザーが作業を再開するポイントが定義されます。これは、システム障害の前に到達したポイントにできるだけ近いポイントとすることができます。最良のポイントは、進行トランザクションを活用して決めることができます。また、アプリケーション内の早いポイントにすることもできます (トランザクションの開始時でも可)。注: ここでいう進行トランザクションとは、ユーザーのためにアプリケーションによって実行された最後のアクションをユーザーが判別できるようにするトランザクションを意味します。

これらの考慮事項を、外部設計記述に含めてください。

**質問 10:** ユーザーは一日のうちのどの時間帯にオンライン・アプリケーションが使用可能であることを期待しますか?

これは、アプリケーション (オンラインおよびバッチ) がコンピューター使用可能時間の大部分を必要とする場合に、重要な考慮事項になります。このような場合、リカバリーのための予防作業 (例えばバックアップ・コピーの作成など) のスケジューリングが難しくなる可能性があります。RLS の静止機能および静止解除機能を参照してください。

## リカバリー要件ステートメントの検証

リカバリー要件に関する問題を考慮した上で、アプリケーションおよびリカバリー要件の公式ステートメントを生成します。

設計またはプログラミング作業を開始する前に、以下を含むすべての利害関係者がステートメントに同意する必要があります。

- ビジネス管理担当者
- データ管理担当者
- エンド・ユーザー、およびコンピューターとオンライン・システム・オペレーションの担当者を含む、アプリケーションを使用するユーザー

## ユーザーの再開手順の設計

システム障害後にユーザーがアプリケーションでの作業を再開する方法を決定します。

### このタスクについて

考慮すべき点は、以下のとおりです。

- サインオン・メッセージでユーザーが自身をシステムに再度識別させる必要性 ( 847 ページの『リカバリー要件に関連する質問』の質問 4 で説明しているように、セキュリティ要件を満たすために必要です)。
- 実行済みの処理と実行されていない処理を知るために、適切な情報をユーザーが使用できるかどうか。進行トランザクションの可能性を検討します。

- 作業の再開時にどの程度の量のキー入力が必要になるか ( 847 ページの『リカバリー要件に関連する質問』の質問 9 で説明しているように、データの再キー入力の実現可能性を判別するために必要です)。

ユーザーの再開手順 (進行トランザクションが使用されている場合はこれも含まれます) を設計するときは、入力データ項目が確実にそれぞれ 1 回だけ処理されるようにするための予防措置を組み込んでください。

## ユーザーの待機手順

システム障害が長引いた場合にアプリケーションの作業をどのように続行できるかを決定します。

例えば、注文入力アプリケーションの場合、(制限された時間内であれば) 手動による方法でオフラインで受注を継続することが現実的な可能性があります。そのような方法を計画している場合は、オフラインのデータを後でシステムに入力する方法を指定します。キャッチアップ機能を提供する必要があるかもしれません。

注: ユーザーが、インテリジェント・ワークステーション、またはプログラマブル・コントローラーが接続されている端末で作業を行っている場合には、MVS ホスト・システムの CICS 領域にアクセスせずに、データの収集を続行できる可能性があります。

## アプリケーションとユーザーの間の通信

アプリケーションごとに、ユーザーが作業する端末のタイプを指定します。

通信の問題を克服するために、例えば以下のような特別な手順を提供するかどうかを決定します。

- ユーザーが代替端末で作業を継続できるようにします (ただし、再度サインオンするなど、適切なセキュリティ上の予防措置を使用します)。
- ユーザーの端末がプログラマブル・コントローラーに接続されている場合は、そのコントローラー (またはそのコントローラーに含まれるプログラム) が提供できるリカバリー・アクションを判別します。
- ユーザーのプリンターが (ハードウェアまたは通信の問題が原因で) 使用不可になった場合は、コンピューター・センターのプリンターなどの代替品をスタンバイとして使用することを検討します。

## セキュリティ

緊急時再始動または通信が切れた場合のセキュリティ手順を決定します。例えば、機密データにリスクがある場合は、ユーザーが再度サインオンし、パスワードを再チェックさせる必要があることを指定します。

障害が単一の端末 (または少数の端末) に限定されており、ユーザーが代替端末を使用する必要がある場合のセキュリティ要件に留意してください。

注: ユーザーのサインオン状態は、持続セッションの再開後は保持されません。

---

## リカバリー関連機能のシステム定義

多数のシステム定義を使用して、CICS アプリケーションに必要なリカバリー・サポートおよび再始動サポートが CICS 領域によって確実に提供されるようにすることが推奨されています。

### システム・リカバリー・テーブル (SRT)

**SRT** システム初期設定パラメーターでシステム・リカバリー・テーブルを指定することが推奨されています。このシステム初期設定パラメーターでテーブルの接尾部を指定しないと、そのパラメーターはデフォルトの YES に設定されます。これは、CICS が、接尾部なしのテーブル (おそらくロード・ライブラリーには存在しません) のロードを試行することを意味します。生成済みのサンプル・テーブル DFHSRT1\$ が、CICSTS55.CICS.SDFHLOAD 内に用意されています。このテーブルがニーズに合っている場合は、SRT=1\$ を指定してください。このテーブルは、CICS が自動的に処理する標準装備のリストに、追加のシステム異常終了コードを追加します (基本の DFHSRT TYPE=INITIAL マクロおよび TYPE=FINAL マクロのみを定義した場合でも)。

追加システムまたは独自のユーザー項目を追加する場合は、サンプル・テーブルを変更してください。SRT の変更については、System recovery table (SRT)を参照してください。

### リカバリーのためのリソース定義

CICS 領域に、**GRPLIST** システム初期設定パラメーターで指定されているリストの 1 つとして DFHLIST が組み込まれていることを確認してください。DFHLIST には、以下のグループが含まれています。これらは基本のリカバリー機能を提供します。

- DFHRSEND
- DFHSTAND
- DFHVTAM

これらのグループの内容については、提供されたリソース定義、グループ、およびリストを参照してください。個々のリソースのリカバリー可能性については、CICS 管理リソースのリカバリーの構成を参照してください。

### システム・ログ・ストリームと汎用ログ・ストリーム

各 CICS 領域のシステム・ログ・ストリームを定義することは、すべてのリカバリー要件の中で最も基本的なことです。トランザクションまたはシステムの異常終了が発生した際にデータ保全性を維持するため、また、CICS がウォーム・リスタートおよび緊急時再始動を実行できるようにするために、システム・ログは必須です。

CICS は、そのすべてのシステム・ロギングおよび汎用ロギングの要件に対して、MVS システム・ロガーのサービスを使用します。CICS ログ・マネージャーは、システム・ログおよび汎用ログのデータを、MVS システム・ロガーに定義されているログ・ストリームに書き込みます。詳しくは、ロギングおよびジャーナリングを参照してください。

### ファイル

RLS モードでアクセスされるように定義されている VSAM ファイルについては、ICF カタログ内のリカバリー属性を、IDCAMS を使用して定義し

ます。非 RLS モードでアクセスされるように定義されている VSAM ファイルについては、これをサポートする DFSMS のレベルを指定することで、CSD ファイル・リソース定義内、または ICF カタログ内のリカバリー属性を定義することができます。BDAM ファイルについては、FCT 内のリカバリー属性を定義します。

#### 一時データ・キュー

区画内キューをリカバリー可能にする場合は、各キューの定義でリカバリー処理オプションを指定します。リカバリー処理オプション RECOVSTATUS の定義については、SESSIONS 属性を参照してください。

#### 一時記憶域キュー

一時記憶域キューをリカバリー可能にする場合は、それらの TSMODEL リソース定義で RECOVERY(YES) オプションを指定します。リカバリー処理オプションの定義については、TSMODEL 属性を参照してください。

#### プログラム・リスト・テーブル (PLT)

DFHPLT マクロを使用して、CICS の初期設定または制御されたシャットダウンの中で実行される各プログラムを指定する、プログラム・リスト・テーブルを作成します。

初期設定時に PLT を使用するには **PLTPI** システム初期設定パラメーターを指定し、シャットダウン時に PLT を使用するには、**PLTSD** システム初期設定パラメーターを指定します。また、CICS のシャットダウン・コマンドでシャットダウン PLT を指定することもできます。

リカバリー目的で起動されるグローバル・ユーザー出口プログラムがある場合は、CICS の初期設定の第 2 段階で使用可能にする必要があります。これらのグローバル・ユーザー出口リカバリー・プログラムは、PLTPI の最初の部分で指定されるアプリケーション・プログラムで使用可能にすることができます。

プログラム・リスト・テーブルの定義については、Program list table (PLT)を参照してください。

#### トランザクション・リスト・テーブル (XLT)

通常のシャットダウンの最初の静止段階で端末から開始できるトランザクションを指定する方法は 2 つあります。

- DFHXLT マクロを使用して、そのトランザクションを指定するトランザクション・リスト・テーブルを作成します。
- トランザクション・リソース定義で SHUTDOWN(ENABLED) 属性を指定します。

---

## 資料およびテスト計画

内部設計時に、定義されたリカバリーと再始動プログラム、出口、およびプロシージャの文書化とテストの実行方法を検討します。

リカバリーと再始動のプログラムおよびプロシージャは、通常、例外条件に関連しているため、通常の条件を扱うものよりテストが難しくなる可能性があります。ただし、設計対象の機能を確実に処理できるように、可能な限りテストする必要があります。

CICS 機能 (実行診断機能 (CEDF) やコマンド・インタープリター (CECI) など) は、例外条件を作成し、それらの条件に対するプログラムやシステムの対応を解釈するのに役立ちます。

インストール済みの CICS システム、アプリケーション・プログラム、オペレーター、および端末ユーザーは、例外条件に対処できなければなりません。したがって、設計者および実装者は予想される例外条件を予測し、リカバリー・プロセスにおけるオペレーターとユーザーの処置を文書化する必要があります。文書には、持続する問題やエラーのエスケープ・プロシージャを含める必要があります。

文書化されたプロシージャを必要とする条件には以下のようなものがあります。

- プロセッサでの電源障害
- CICS の障害
- データ・セットの物理的な障害
- トランザクションの異常終了
- 電話回線の損失やプリンターのサービス休止などの通信障害
- 通信障害またはバックアウト障害による作業単位の中断

障害に対処しなければならない場合があるすべての利用者によって、制御された環境で、リカバリーと再始動のプロシージャをテストし、リハーサルすることが重要です。これは、臨時のオペレーターが関与するインストール済み環境では特に重要です。

---

## リカバリーのためのプログラミング

アプリケーション・プログラムを設計する場合は、CICS が提供しているリカバリー機能を組み込むことができます。例えば、バックアウト・リカバリーのためのグローバル・ユーザー出口を使用することができます。

このセクションでは、以下のトピックを取り上げます。

- 『リカバリーのためのアプリケーションの設計』
- 856 ページの『プログラム設計』
- 863 ページの『トランザクション障害およびシステム障害の管理』
- 868 ページの『アプリケーション・プログラムにおけるリソースのロック (エンキュー)』
- 876 ページの『トランザクション・バックアウトのユーザー出口』

## リカバリーのためのアプリケーションの設計

このコンテキストでは、アプリケーションは、ユーザー組織の特定のニーズを満たすように設計された、1 つ以上のトランザクションのセットを指します。トランザクションは、設計担当者が 1 つのエンティティーと見なすことにした、アプリケーション内のアクションのセットを指します。これは、通常は CICS に接続された端末からその ID でトランザクションを呼び出すことによって CICS 領域内で開始される実行単位に対応しています。

アプリケーションの設計者は、アプリケーションを (存在する場合) どのようにしてトランザクションに分割するか、およびそのトランザクションを単一の作業単位で構成するか複数の作業単位で構成するかを決める必要があります。

トランザクションは、1 つの作業単位に対応しているのが理想です (必須ではありません)。ビジネス・アプリケーションをトランザクションに対応する作業単位に分割すると、リカバリー・プロセス全体が単純化されます。

標準的なビジネス・アプリケーションの例としては、注文入力システムがあります。標準的な注文入力アプリケーションには、顧客からの単一の注文を処理するために必要なすべてのプロセスが組み込まれています。それらのプロセスは、以下のように、処理単位のセットとして設計されます。

1. 顧客の名前と住所をチェックし、注文番号を割り振る。
2. 注文された品目の詳細を記録し、在庫ファイルを更新する。
3. 送り状や配送に関する文書を印刷する。

合意したリカバリー要件記述に応じて、注文された品目の注記の詳細や、単一の大きなトランザクションか複数のトランザクション (その注文に含まれる品目ごとに 1 つのトランザクション) のいずれかとしてファイルを更新する方法を設計できます。

## アプリケーションをトランザクションに分割する

アプリケーションをトランザクションに分割する方法を指定します。

### 手順

1. 各トランザクションに名前を付け、その機能を、端末ユーザーが理解できる言葉で記述します。アプリケーションには、以下のような、障害からリカバリーするためのトランザクションを組み込むことができます。
  - 進行トランザクション は、アプリケーション全体の進行状況をチェックします。このような機能は、トランザクションが失敗した後や緊急時再始動後だけでなく、通常の運用中の任意の時点で使用することができます。例えば、端末ユーザーが中断された作業を再開するのに適切な再始動点を見つけるように設計することができます。これは特に疑似会話に関連します。
  - キャッチアップ機能 は、ユーザーがシステム障害の発生中に他の手段で蓄積させておくことを余儀なくされたデータを入力するためのものです。
2. 各処理単位でアクセスできるファイルおよびデータベースを指定します。そのアクセスできるファイルおよびデータベースのうち、読み取り専用ではなく更新されるものを指定します。
3. アプリケーションの処理単位によって更新されるそれらのファイルおよびデータベースに更新を適用する方法を指定します。ここで考慮すべき要素には、整合性や更新の即時性などがあります。
  - a. データの保全性を確保するための相互ステップで行う必要がある更新を (もしあれば) 指定します。例えば、注文入力アプリケーションでは、在庫ファイルからある数量を差し引くと同時に、同じ数量を配送予定ファイルに確実に追加するようにしなければならない可能性があります。
  - b. 新しく入力されたデータを、いつ、ファイルまたはデータベースに適用する必要があるか、または適用できるかを指定します。

- アプリケーションの処理単位は、ユーザーからデータを受け取ると、すぐにファイルおよびデータベースを更新します。
- アプリケーションの処理単位は、後のアクション (例えば、同じアプリケーション内の後の処理単位や、夜間に実行されるバッチ・アプリケーションなど) に備えて更新を蓄積させておきます。バッチ・オプションを選択する場合は、その数の更新を完了するのに十分な時間がそのバッチ作業に与えられていることを確認してください。

この情報は、アプリケーションの処理単位の内部設計を行う際に使用してください。

4. アプリケーションの処理単位から別の処理単位に渡される必要があるデータを指定します。例えば、注文入力アプリケーションでは、ある 1 つの処理単位によって注文品目が蓄積される可能性があります。在庫ファイルの更新は、別の処理単位が行います。この場合には、明らかに、1 つ目の処理単位によって蓄積されたデータを、2 つ目の処理単位に渡す必要があります。この情報は、各処理単位にどのようなリソースが必要かを判断する際に使用してください。

#### 処理単位間の関係:

アプリケーションの処理単位から別の処理単位に渡される必要があるデータを指定します。

例えば、注文入力アプリケーションでは、ある 1 つの処理単位によって注文品目が蓄積される可能性があります。在庫ファイルの更新は、別の処理単位が行います。この場合には、明らかに、1 つ目の処理単位によって蓄積されたデータを、2 つ目の処理単位に渡す必要があります。

この情報は、各処理単位にどのようなリソースが必要かを判断する際に必要になります ( 858 ページの『トランザクション間でデータを渡すためのメカニズム』を参照してください)。

### SAA 互換アプリケーション

システム・アプリケーション体系 (Systems Application Architecture<sup>®</sup>, SAA) 互換アプリケーションを実装する必要がある場合、SAA 共通プログラミング・インターフェース (CPI) のリソース・リカバリー・エレメントを標準 CICS アプリケーション・プログラム・インターフェース (API) の代わりに使用することができます。

SAA リソース・リカバリー・インターフェースの CICS 実装環境で提供されるリソース・リカバリー機能は、CICS API で提供されるものと同じです。したがって、アプリケーションが SAA 互換である必要がある場合にのみ、CICS API から SAA リソース・リカバリー・コマンドに変更する必要があります。

SAA リソース・リカバリー・インターフェースを使用するには、EXEC CICS SYNCPOINT コマンドの代わりに、アプリケーションで SAA リソース・リカバリー・コマンドを含める必要があります。本書では、CICS API リソース・リカバリー・コマンドについてのみ説明しています。SAA リソース・リカバリー・インターフェースについては、Systems Application Architecture Common Programming Interface Resource Recovery Reference を参照してください。

## プログラム設計

このセクションでは、CICS のリカバリー機能を効率的に使用するようにプログラムを設計する方法について説明します。

### トランザクションを作業単位に分割する

ユーザーは、アプリケーションの処理単位をどのような形（トランザクション、作業単位、およびプログラム）で実装するかを決める必要があります。

#### このタスクについて

以下のアドバイスに従ってアプリケーションの処理単位の計画を立てることをお勧めします。

#### 手順

1. ユーザーに表示するダイアログをサポートするプログラムでは、単一の読み取り端末と単一の書き込み端末のみを組み込むように各作業単位を実装することを検討します。この方法を使用すれば、ユーザーの再始動手順を単純化することができます（857 ページの『ユーザーに表示するダイアログの処理』も参照してください）。

短い作業単位のほうが望ましい、以下のようないくつかの理由があります。

- データ・リソースがロックされる時間が短い。そのため、他のタスクがそのリソースの解放を待たなければならない可能性が減ります。
- (動的トランザクション・バックアウトまたは緊急時再始動時の) バックアウトの処理時間が短縮される。
- 障害の発生後にトランザクションが再始動したときのユーザーによる再入力が少なくなる。

再キー入力がほとんど、またはまったくなくようにすることが可能なアプリケーション（847 ページの『リカバリー要件に関連する質問』の質問 9 で説明しています）では、作業単位を短くして、入力されたすべてのデータができるだけ早くコミットされるようにすることが極めて重要です。

2. トランザクションを多数の作業単位に分割するかどうかを決める際には、リカバリーおよび再始動への影響を考慮します。

動的トランザクション・バックアウトやトランザクション再始動などの CICS 機能が最も効率的に処理を行うのは、単一の作業単位のみを持つトランザクションの場合です。しかし、複数の作業単位を持つトランザクションが必要な場合もあります。例えば、ファイルまたはデータベースの一連の更新を必ず単一の作業単位でコミットする必要があるものの、そのトランザクションのそれ以降の処理は 1 つ以上の作業単位で続行される場合などです。

3. ファイルまたはデータベースの複数の更新を手順に入れておく必要がある場合は、そのアプリケーションがそれらの更新を同じ作業単位で行うことを確認してください。この方法により、確実にそれらの更新がすべてまとめてコミットされるようにします。その作業単位が中断された場合、それらの更新はまとめてバックアウトされて、整合性が保たれます。

## ユーザーに表示するダイアログの処理

アプリケーションは、ユーザーといくつかの対話（入出力）を行うことが必要な場合があります。

CICS は、そのようなシチュエーションで使用するプログラムを設計するための、以下の基本的な技法を提供しています。

- 会話型処理
- 疑似会話型処理

### 会話型処理:

会話型処理では、ユーザーが出力を読み取り入力を行うのにかかる時間を含め、すべての端末対話にわたるタスクとしてトランザクションの実行が続行されます。

タスクは、実行中に他のタスクで必要になる可能性のあるリソースを保持します。以下に例を示します。

- タスクは、長期間にわたり、ストレージを占有し、データベース・レコードをロックします。また、障害や後続のバックアウトの発生時には、障害発生時点までに行われたファイルおよびデータベースに対するすべての更新をバックアウトする必要があります（トランザクションが作業単位に細かく分割されている場合を除く）。
- トランザクションが DL/I を使用し、スケジュールされた PSB の数が最大許容数に達した場合、PSB をさらにスケジュールする必要があるタスクは待機する必要があります。

会話型処理は通常は好まれません、データ保全性を維持するために、ユーザーとの複数の対話で行った複数のファイル更新またはデータベース更新を相互に関連付ける必要がある（つまり、すべて一緒にコミットする必要があるか、すべて一緒にバックアウトする必要がある）場合は会話型処理が必要になることがあります。

### 疑似会話型処理:

疑似会話型処理では、ユーザーとの連続する端末対話は別個のタスクとして処理され、通常はそれぞれ 1 つの作業単位で構成されます。

この方法では、タスク間またはトランザクション間の通信（858 ページの『トランザクション間でデータを渡すためのメカニズム』を参照）が必要になることがあり、アプリケーション・プログラミングが会話型処理の場合よりも少し複雑になることがあります。

ただし、各タスクの終了時に、更新はコミットされ、タスクに関連付けられているリソースは他のタスクで使用するために解放されます。そのため、通常、疑似会話型トランザクションは会話型トランザクションより優先されます。

ユーザーとの複数の端末対話が相互に関連する場合、更新対象データがリカバリー可能リソースに蓄積してから、単一のタスク（会話の最後の対話など）でデータベースに適用する必要があります。障害発生時には、緊急時再始動または動的トランザクション・バックアウトで個々のステップ中に行われた更新のみがバックアウトされます。アプリケーションは会話の適切な時点で再始動を行う必要があります（その際に画面フォーマットが再作成される場合があります）。

ただし、他のタスクが、更新情報が受け入れられてから、データベースに適用されるまでの間にデータベースを更新しようとする可能性があります。他のアプリケーションが、ご自分のアプリケーションによる更新の妨げになる場合には、データベースを更新できないようにご自分のアプリケーションを設計してください。

## トランザクション間でデータを渡すためのメカニズム

トランザクションが前のトランザクションによって作成された作業データにアクセスする必要があるアプリケーションでは、どのようなメカニズムでトランザクション間のデータの受け渡しを行うかを決める必要があります。

トランザクション間のデータの受け渡しについては 2 つのオプションがあります。

- 主記憶域
- CICS のリカバリー可能リソース

### 主記憶域

主記憶域の利点は、リカバリーが重要ではない場合、または、同じタスクにサービス提供しているプログラム間でデータを受け渡す場合にのみ実現されます。

トランザクション間でデータを受け渡すために使用できる主記憶域には、以下のものがあります。

- 通信域 (COMMAREA)
- 共通作業域 (CWA)
- 一時記憶域 (メイン)
- 端末管理テーブル・ユーザー域 (TCTUA)

CICS は、これらの領域に対する変更をログに記録しません (このセクションで後述する場合を除く)。したがって、制御されていないシャットダウンが起きた場合、これらの領域のいずれかに保管されているデータは失われます。これにより、それらのデータは、緊急時再始動中にトランザクション間でデータを保持する必要があるアプリケーションには適さなくなります。また、これらの一部の記憶域は、トランザクション間の類縁性の原因となります。この類縁性は、動的トランザクション・ルーティングの妨げとなります。トランザクション間の類縁性を回避するには、COMMAREA または TCTUA のいずれかを使用します。トランザクション間の類縁性については、類縁性を参照してください。

そのプログラムに初めて制御が渡されたかどうかを示す際に COMMAREA 内のデータの有無に依存することがないように、プログラムを設計します (例えば、長さ 0 のデータをテストすることによって)。動的トランザクション・バックアウトおよび自動再始動が指定されている場合は、トランザクションの異常終了を検討してください。異常終了後、端末から、次のトランザクションに COMMAREA を渡すことができます (新しいトランザクションが関連付けられていない場合でも)。端末管理テーブル・ユーザー域 (TCTUA) にも同様の考慮事項が適用されます。

### CICS のリカバリー可能リソース

トランザクション間の通信に使用できる、バックアウトによってリカバリー可能なリソースには、以下のものがあります。

- 一時記憶域 (補助)

- 一時データ・キュー
- ユーザー・ファイル、および DL/I データベースと Db2 データベース
- データ・テーブル (ユーザー保守)
- カップリング・ファシリティ・データ・テーブル

タスクが異常終了した場合、CICS は、これらすべてのリソースを、未完了の作業単位が開始されたときの状況に戻すことができます。

#### 一時記憶域 (補助):

一時記憶域項目を使用して、トランザクション間通信を行うことができます。そのため、一時記憶域項目が端末 ID に固有のものである必要があります。端末が使用不可になると、トランザクション・シーケンスは、端末が再び使用可能になるまで中断されます。

指定された一時記憶域キューの読み取りと再読み取りは可能ですが、トランザクション・シーケンス間の通信が不要になったら、アプリケーション・プログラムはキューを削除する必要があります。

#### 一時データ・キュー:

一時データ (区画内) は、トランザクション間通信用の一時記憶域 (補助) に似ています。主な違いは、一時データ・キューの各レコードを一度だけ読み取ることができる、その後、そのレコードが使用できなくなることです。

未完了の作業単位の開始位置へのバックアウトを行うには、一時データを論理的にリカバリー可能として指定する必要があります。

#### ユーザー・ファイル、および DL/I データベースと Db2 データベース:

トランザクション間でデータ通信を行うためにファイルまたはデータベース・セグメントを専用化することができます。

トランザクションでは、専用ファイルまたはデータベース・セグメントに、特定機能の完了を記録することができます。進行中のトランザクション (実行された更新と実行されていない更新をユーザーに知らせるためのもの) で専用ファイルまたはセグメントを調べることができます。

物理的損傷が発生した場合、ユーザー VSAM ファイル、DL/I、および Db2 データベースを順方向リカバリーすることができます。

#### ユーザー保守のデータ・テーブル:

作業単位の障害発生後にリカバリー可能なユーザー保守データ・テーブル (UMT) は、トランザクション間でのデータの受け渡しに役立つことがあります。ただし、それらは順方向にリカバリーできず、CICS 再始動後にリカバリーできません。

#### カップリング・ファシリティ・データ・テーブル:

ロック・モデルを使用して更新され、作業単位の障害発生後にリカバリー可能なカップリング・ファシリティ・データ・テーブルは、トランザクション間でのデータの受け渡しに役立つことがあります。

UMT とは異なり、カップリング・ファシリティ・データ・テーブルは、CICS 障害、CFDT サーバー障害、または MVS 障害が発生した場合にリカバリー可能です。ただし、それらは順方向にはリカバリーできません。

## トランザクションのデッドロックを回避する設計

トランザクションのデッドロックを回避するようにプログラムを設計する必要があります。そのような状態を回避するためにプログラムで利用できる手法は多数あります。

### このタスクについて

以下の手法の使用を検討してください。

#### 手順

- あらかじめ合意した順序でファイルにアクセスするように、すべてのトランザクションを調整します。これは、インストール標準にとって適切な手法と考えられます。複数のパスから更新できるようにする場合には、特別な注意が必要です。
- 明示インストール・エンキュー標準を強制することで、すべてのアプリケーションが以下を行うようにします。
  1. 同じ文字ストリングによるエンキュー
  2. 同じ順序でそれらのストリングを使用する
- ファイル内のレコードには、常に同じ順序でアクセスします。例えば、複数のファイルまたはデータベース・レコードを更新する場合、必ず昇順でそれらにアクセスするようにします。

そのためには、以下のようになります。

1. 端末オペレーターは、常に既存のデータ・セットの順序でデータを入力します。

この方式では、端末オペレーターは特別な操作を行う必要があり、これをアプリケーションの制約の中で行うのは、現実的ではない場合があります。(例えば、電話で受けた注文は、製品番号の順序がランダムになることがあります。)

2. アプリケーション・プログラムで、まず、データ項目の順序がデータ・セットの順序と一致するように、入力トランザクションの内容をソートします。

この方式は、追加のアプリケーション・プログラムを必要としますが、端末オペレーターまたはアプリケーションに外部制約を課しません。

3. アプリケーション・プログラムが、トランザクションに入力された各データ項目の処理後に、SYNCPOINT コマンドを発行します。

この方式に必要な追加プログラミングは、2 番目の方式よりも少なくなります。ただし、同期点を発行するということは、以前に処理されたトランザクション内のデータ項目は、トランザクション全体が終了する前にシステムまたはトランザクションの障害が発生した場合にもバックアウトされないことを意味します。これはそのアプリケーションにとって妥当ではない場合があります。トランザクションのデータ項目のうちどれが処理され、どれが CICS によってバックアウトされたかについて疑問を生じさせます。トランザクショ

ン全体をバックアウトする必要がある場合は、同期点を発行しないようにするか、トランザクションごとに 1 つのデータ項目だけを入力するようにします。

3 つの方式のうち、2 番目 (プログラミングによってデータ項目を昇順にソートする方式) が、最も広く受け入れられます。

データ・セットの更新を、基本パスと 1 つ以上の代替索引パスから、または複数の代替索引パスからできるようにする場合、順序付けレコードを更新すると、トランザクション・デッドロックに対する保護が提供されないことがあります。保護されない原因は、おそらく、異なる基本キー・シーケンスがすべて昇順 (または降順) になっていないためです。複数のパスから更新できるようにする場合で、複数のレコードを更新する必要がある場合は、常に単一のパスまたは基本パスを使用してください。このような手順を、インストール標準に規定してください。

### インターバル制御機能の **START** 要求の影響

インターバル制御機能の **START** 要求は、別のタスク (例えば、**START** を発行したタスクによって蓄積されている更新の実行など) を開始します。これにより、ユーザーは、更新が適用されるまで待たずに、データの蓄積を続行することができます。

**START** 要求の **PROTECT** オプションは、**START** を発行したタスクが作業単位の途中で失敗した場合に、新しいタスクの開始時間が過ぎていても開始されないようにします。( **PROTECT** オプションについて詳しくは、**START** 要求のリカバリーを参照してください。 )

開始済みタスクの失敗の可能性についても検討してください。プログラムに異常終了処理を組み込んでいなければ、その失敗を認識するのはマスター端末のみです。異常終了処理では、失敗の原因をできる限り分析し、タスクを再始動することが適切であればそうする必要があります。ユーザーまたはマスター端末オペレーターのいずれかが確実に適切な操作を行って、更新を繰り返せるようにします。例えば、ユーザーがタスクを再開できるようにすることができます。

また別の解決策としては、開始済みのトランザクションが、独自の **TRANSID** を指定した **START** コマンドを発行します。 **RETURN** コマンドを発行する直前に、そのトランザクションは **START** コマンドを取り消す必要があります。こうすると、開始済みのタスクは、失敗した場合に自動的に再始動します。( **START** コマンドに指定されている間隔が短すぎると、トランザクションは、最初の起動がまだ実行されている間に再度起動される可能性があります。これを防ぐために、十分な長さの間隔が指定されていることを確認してください。 )

### 自動タスク開始の影響 (**TD** トリガー・レベル)

区画内一時データ宛先のリソース定義で **TRANSID** オペランドを指定すると、トリガー・レベルに達したときに、指定されたトランザクションが開始されます。宛先タイプは論理的にリカバリー可能と指定してください。これにより、一時データ・レコードがコミットされてから、タスクがこれらのレコードを実行および使用するようになります。

## ユーザーに大量のデータを表示した場合の影響

理想的には、ファイルまたはデータベースを更新するトランザクションは、それらの更新が (ユーザーの同期点またはタスクの終了によって) コミットされるまで、ユーザーに対する確認を保留する必要があります。

アプリケーションが、一度に表示しきれないような大量のデータ (ブラウズに必要なデータなど) で構成されるものについて応答を必要とする場合には、以下のようないくつかの手法を使用できます。

- BMS による端末ページング
- 一時データ・キューの使用

### BMS による端末ページング:

アプリケーション・プログラム (**SEND PAGE BMS** コマンドを使用) は、オペレーター・ページ・コマンドを使用して、後続の表示用の一時記憶域キューに出力データのページを作成します。

このようなキューは、もちろん、一時記憶域のリカバリーで説明されているように、リカバリー可能として指定する必要があります。

アプリケーション・プログラムは、タスクが完了したことと、出力データが端末ページの形式で使用可能であることを示すコミット済み出力メッセージをユーザーに送信する必要があります。

ユーザーがデータのページを表示しているときに非制御終了が発生した場合、それらのページは失われません (BMS の一時記憶域がリカバリー可能として指定されていることが前提)。緊急時再始動後に、ユーザーは、CSPG CICS 提供のトランザクションと端末ページング・コマンドを使用して、端末ページングを再開できます。(CSPG について詳しくは、CSPG - ページ検索を参照してください)。

### 一時データ・キューの使用:

いくつかのタスクが大量のデータを単一の端末 (例えば、ユーザーが開始した複数ページ・レポートを受信するプリンター) に送信する場合、端末での受信準備ができるまで、(ディスク上の) データをキューに入れておく必要がある場合があります。

### このタスクについて

このようなキューイングは、端末に関連付けられている一時データ・キューで行うことができます。その後、端末が使用可能であるときにトリガーされる特殊なトランザクションは、データをフォーマットして表示できます。

### リカバリーと再始動が目的の場合:

- 一時データ・キューは論理的にリカバリー可能として指定する必要があります。
- データを表示するトランザクションが失敗すると、動的トランザクション・バックアウトが呼び出されます。

ただし、トランザクションが実行される端末がプリンターである場合、動的トランザクション・バックアウト (おおよどどのような手段によるものでもトランザク

ションの再始動)が発生すると、出力の部分的重複が生じて、特別なユーザー手順が必要になる可能性があります。最良の解決策は、各作業単位がプリンターのページまたはフォームに対応していることを確認することです。

## トランザクション障害およびシステム障害の管理

トランザクション障害、およびシステムの制御されていないシャットダウンを管理するために役立てることができる機能は、多数存在します。

### このタスクについて

それらの機能によって、以下のことが保証されます。

1. ファイルおよびデータベースは、整合性と一貫性のある状態に維持されます。
2. プログラムが失敗した場合は、診断情報および警告情報が作成されます。
3. トランザクション間の通信は、障害の影響を受けません。

CICS が実行するアクションについては、作業単位のリカバリーと異常終了処理およびオペレーティング・システムの異常終了およびプログラム・チェックの処理で説明されています。

### トランザクション障害

トランザクションが失敗した場合、異常終了の処理中と処理後に CICS 機能呼び出すことができます。

次の機能が含まれます。

- CICS 条件処理
- HANDLE ABEND コマンド、およびユーザー出口コード
- SYNCPOINT ROLLBACK コマンド
- 動的トランザクション・バックアウト (DTB)
- DTB 後のトランザクション再始動
- プログラム・エラー・プログラム (DFHPEP)

これらの機能を個別または一緒に使用することができます。内部設計フェーズ時に、どの機能を使用するかを指定し、どんな追加の (アプリケーションまたはシステム) プログラミングが必要となる可能性があるかを判別します。

コマンドに RESP オプションを指定すると、テスト可能な条件 ID が返されます。あるいは、特定の条件が発生した場合に制御が渡されるラベルを指定するために、トランザクション・プログラムのローカル・コンテキストで HANDLE CONDITION コマンドを使用します。

例えば、(デフォルトの処置が単にタスクの異常終了であるときに) ファイルの入出力エラーが発生した場合、特にファイルのいずれかがアプリケーションに不可欠である場合には、CICS の強制終了を決定できるマスター端末オペレーターにエラーを通知することができます。

ご使用のシステムに、RESP オプションまたは HANDLE CONDITION コマンドの使用に関する標準がある場合があります。新規アプリケーションごとにそれらを確認してください。

## HANDLE ABEND コマンド

**HANDLE ABEND** コマンドは、トランザクション内のルーチンに、または別々にコンパイルされたプログラムに、タスクの異常終了時に制御を渡すことができます。

異常終了処理コードで行う可能性のある作業には次のようなものがあります。

- タスクが異常終了する前に (CICS で提供されるものに加えて) 診断情報を取り込み、メッセージをマスター端末とユーザーに送信する。
- 開始要求の取り消しなど、クリーンアップ処置を実行する (PROTECT オプションが使用されていない場合)。
- 異常終了の前に実行される明示的なジャーナル処理の影響を逆転させるためにジャーナル・レコードを書き込む。

ご使用のシステムに **HANDLE ABEND** コマンドの使用に関する標準がある場合があります。新規アプリケーションごとにそれらを確認してください。

## EXEC CICS SYNCPOINT ROLLBACK コマンド

**ROLLBACK** はトランザクション内で役立つ場合があります。例えば、いくつかのデータベース更新が開始された後、同期点でコミットされる前に、トランザクションが論理的に矛盾する入力を検出した場合などです。

使用を決定する前に、以下の点を考慮してください。

- ロールバックにより、タスク全体ではなく、現行の作業単位のみで実行されたりリカバリー可能リソースへの更新がバックアウトされます。
- **SYNCPOINT** コマンドは、**ROLLBACK** オプションの指定の有無に関係なく、新規の作業単位を開始します。
- トランザクションが異常終了し、そのトランザクションで処理を続行しない場合は、**EXEC CICS ABEND** を発行して、動的トランザクション・バックアウトで更新をバックアウトし、データ保全性を確保できます。ロールバックは、作業単位の影響を無効にしてから、アプリケーションで制御を再取得する場合にのみ使用します。

**SYNCPOINT** コマンドのプログラミング情報については、**SYNCPOINT**を参照してください。

## 動的トランザクション・バックアウト

動的トランザクション・バックアウト (DTB) は、リカバリー可能リソースを含むすべてのトランザクションに対して自動的に行われます。トランザクション・リソース定義で指定できるオプションではありません。DTB のアクションは、トランザクション・バックアウトで説明されています。

以下を覚えておいてください。

- リカバリー可能リソースにアクセスするトランザクションの場合、DTB は論理データ保全性を維持するために役立ちます。
- 更新されるリソースはリカバリー可能である必要があります。
- DTB は、プログラム・レベルの異常終了出口 (存在する場合) がクリーンアップまたは論理リカバリーを試行した後にのみ行われます。

## DTB 後のトランザクション再始動

DTB が指定されているトランザクションごとに、自動トランザクション再始動も指定することを考慮してください。例えば、DL/I データベースにアクセスする（プログラム分離デッドロックが発生する可能性がある）トランザクションの場合、通常は自動トランザクション再始動が指定されます。

トランザクション再始動が指定されている場合でも、タスクは、特定のデフォルト条件（タスクの異常終了にリストされています）でのみ自動的に再始動します。これらの条件の変更が不可欠である場合は、再始動プログラム DFHREST を変更して行うことができます。

## プログラム・エラー・プログラム (DFHPEP) の使用

独自の機能を組み込むかどうかを決定します。例については、CICS 提供の PEP を参照してください。（DFHPEP は、タスクの異常終了で説明されているように、タスクの異常終了時に呼び出されます。）

## システム障害

緊急時再始動後にアプリケーションをどのように再始動するかを指定します。

再始動プロセスを自動化するかどうかに応じて、アプリケーションおよびシステムのプログラミングで以下の機能を提供することができます。

- 以下を処理するためのトランザクション・バックアウト処理のユーザー出口
  - BDAM ファイルまたは VSAM-ESDS ファイルに追加されたレコードの論理的な削除 (XFCLDEL グローバル・ユーザー出口点について詳しくは、出口 XFCLDEL、ファイル制御論理削除出口を参照)
  - ファイル制御ログ・レコードのバックアウト (XFCBOUT グローバル・ユーザー出口点について詳しくは、出口 XFCLDEL、ファイル制御論理削除出口を参照)
  - トランザクション・バックアウト中のファイル・エラー (XFCBFAIL グローバル・ユーザー出口点について詳しくは、出口 XFCBFAIL、ファイル制御バックアウト障害出口を参照)
  - 緊急時再始動中にシステム・ログから読み取ったユーザー・リカバリー・レコード (XRCINPT グローバル・ユーザー出口点について詳しくは、出口 XRCINPT を参照)
- 実行された更新と実行されなかった更新をユーザーが知るために役立つ進行トランザクション。この目的のために、既存のファイルまたはデータベースから特定のタイプの最終レコードまたはセグメントを検索するアプリケーション・コードを書くことができます。

## 異常終了の処理とプログラム・レベルの異常終了出口

プログラム・レベルの異常終了出口コードを作成することができます。これにより、プログラムは、発生した異常終了によって異なる処理を行うことができます。

例えば、以下のいずれかの方法でプログラムが処理を進めるようにすることができます。ただし、異常終了出口コードは最小限に抑えることをお勧めします。

- ・ タスクが異常終了する場合は、アプリケーションに依存する、そのタスク関連の情報を記録します。

ダンプを開始する場合は、その異常終了と同じプログラム・レベルの出口コードでそれを開始します。異常終了が発生したプログラム・レベルよりも高いプログラム・レベルでダンプを開始すると、貴重な診断情報が失われる可能性があります。

- ・ ローカル・リカバリーを試みてから、プログラムの実行を続けます。
- ・ 端末オペレーターにメッセージを送信します。例えば、入力データの誤りが異常終了の原因であると思われる場合に、これを行うことができます。

### プログラム・レベルの出口ルーチンまたは出口プログラムが使用できる情報

次の表は、プログラム・レベルの出口ルーチンまたは出口プログラムに ASSIGN コマンドが提供する情報を示しています。

コマンド	提供される情報
ADDRESS TWA	トランザクション作業域 (TWA) のアドレス
ASSIGN ABCODE	現在の CICS 異常終了コード
ASSIGN ABOFFSET	最後の ASRA、ASRB、または ASRD 異常終了のオフセット
ASSIGN ABPROGRAM	最後の異常終了時に失敗したプログラムの名前
ASSIGN ASRAINTRPT	最後の AICA、ASRA、ASRB、ASRD、または ASRE 異常終了の命令長コード (ILC) およびプログラム割り込みコード (PIC) データ
ASSIGN ASRAKEY	最後の AEYD、AEYF、AICA、ASRA、または ASRB 異常終了が発生したときの実行キー (もしあれば)
ASSIGN ASRAPSW	最後の AICA、ASRA、ASRB、ASRD、または ASRE 異常終了のプログラム状況ワード (PSW)
ASSIGN ASRAPSW16	最後の AICA、ASRA、ASRB、ASRD、または ASRE 異常終了の 16 バイトの PSW
ASSIGN ASRAREGS	最後の AICA、ASRA、ASRB、ASRD、または ASRE 異常終了の汎用レジスター
ASSIGN ASRAREGS64	最後の AICA、ASRA、ASRB、ASRD、または ASRE 異常終了の 64 ビットの汎用レジスター
ASSIGN ASRASPC	最後の AEYD、AEYF、AICA、ASRA、または ASRB 異常終了が発生したときに制御下にあったスペースのタイプ (もしあれば)
ASSIGN ASRASTG	最後の AEYD、AEYF、AICA、ASRA、または ASRB 異常終了が発生したときにアドレス指定されていたストレージのタイプ (もしあれば)
ASSIGN ORGABCODE	異常終了が繰り返される場合の、オリジナルの異常終了コード

### 考慮事項

CICS サービスの呼び出し時に異常終了が発生した場合、その同じサービスに対してさらに要求を発行すると、予測不能な結果が起きる場合があります。これは、ポイ

ンターと作業域の再初期化、および出口ルーチンでのストレージ域の解放が完了していない可能性があるためです。さらに、ASP<sub>x</sub> 異常終了 (同期点の処理中に発生するタスクの異常終了) は、アプリケーション・プログラムでは処理できません。

異常終了が発生すると動的にバックアウトされるトランザクションの場合は、RETURN コマンドで終わる出口コードの作成について考慮してください。このような出口コードを作成すると、トランザクションが正常に終了したことを CICS に通知するため、動的なトランザクション・バックアウトおよび自動によるトランザクション再始動 (該当する場合) が行われません。

出口プログラムはサポートされる任意の言語でコーディングできますが、出口ルーチンはその出口ルーチンが一部となっているプログラムと同じ言語である必要があります。

### さらに学習したい方に

CICS が開始した異常終了のトランザクション異常終了コード、それらの意味、および推奨されるアクションについては、トランザクション異常終了コードを参照してください。

プログラム・レベルの出口コードのコーディングに関連するプログラミング情報 (アドレス可能度やレジスターの使用など) については、プログラム・レベルの異常終了プログラムまたは異常終了ルーチンの作成を参照してください。背景情報については、異常終了のリカバリーを参照してください。

## IOERR 条件の処理

リカバリー可能リソースの IOERR 条件の処理を試行するプログラムは、RETURN コマンドまたは SYNCPOINT コマンドを発行せずに、ABEND コマンドを発行して終了する必要があります。RETURN コマンドまたは SYNCPOINT コマンドは、リカバリー・マネージャーに作業単位を完了させ、リカバリー可能リソースに対する変更をコミットします。

## START TRANSID コマンド

**START TRANSID** コマンドを使用して他のトランザクションを開始するトランザクションでは、論理データ保全性を維持する必要があります。

データ保全性は、以下のガイドラインに従うことで維持できます。

1. 常に、START TRANSID コマンドの PROTECT オプションを使用してください。そうすることで、START を発行するタスクがバックアウトされる場合に、新しいタスクが開始されることがないようにします。
2. 開始されたトランザクションに (データ・オプション FROM、RTERMID、RTRANSID、または QUEUE のいずれかで) データを渡す場合には、必ず、関連付けられた一時記憶域キューを、その TSMODEL リソース定義の中でリカバリー可能として定義します。いずれかのデータ・オプションを使用する際に START コマンドの REQID オプションを指定することで、一時記憶域キューの名前を選択することができます。REQID を使用しない場合に CICS によって生成される一時記憶域キュー名のフォーマットは、「DFRxxx」です。

一時記憶域キューをリカバリー可能として定義すると、START を発行するタスクが失敗してバックアウトされる場合に、別のタスクに渡されるデータは一時記憶域キューから確実に削除されます。START を発行するタスクがその同期点を完了した後にシステム障害が発生した場合、その START コマンドは保持されます。有効期限時刻に達しており、TERMID オプションで指定された端末が使用可能な場合、CICS は、リカバリー可能な START コマンドで指定されたトランザクションを、緊急時再始動後に開始します。

注: シーケンス内の次のトランザクションを同じ端末で開始することが目的の場合は、IMMEDIATE オプションで EXEC CICS RETURN TRANSID(...) を使用することを検討してください。これは端末をアンロックしません。また、動的トランザクション・ルーティング (DTR) 環境の場合、そのトランザクションは DTR の対象になります。

## PL/I プログラムとエラー処理

ON ユニットは、PL/I プログラムにおける標準的なエラー処理の方法です。実行時オプション STAE が指定されている場合、CICS プログラム制御サービスは、PL/I の ON ユニットをアクティブ化する出口ルーチンをセットアップします。

この出口ルーチンは以下を処理できます。

- すべての PL/I エラー
- PL/I プログラムおよび関連する CICS サービスで発生した CICS の異常終了
- プログラム・チェック

CICS では、PL/I 実行時オプションは、PLIXOPT 文字ストリングでのみ指定できます。

CICS 環境における PL/I コーディングの制約事項について詳しくは、ご使用のコンパイラーの適切な PL/I プログラマーの手引き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リソースのロック (エンキュー)

このトピックでは、データ保全性を保護するために CICS が提供する、ロック (エンキュー) 機能およびアクセス方式について説明します。

### このタスクについて

ロックには、以下の 2 つの形式があります。

1. CICS によって実行される暗黙ロック機能 (またはアクセス方式)。これは、トランザクションがデータの変更要求を出すときには必ず実行されます。これについては、以下で説明されています。
  - 869 ページの『ファイルの暗黙ロック』
  - 872 ページの『論理的にリカバリー可能な TD 宛先に対する暗黙エンキュー』
  - 873 ページの『リカバリー可能な一時記憶域キューに対する暗黙エンキュー』

- 873 ページの『DBCTL を使用する DL/I データベースに対する暗黙エンキュー』.
2. 明示エンキュー機能。これは、EXEC CICS コマンドを使用して要求します。これについては、874 ページの『(アプリケーション・プログラマーによる) 明示エンキュー』で説明されています。

注: データ・リソースを (暗黙または明示) ロックすると、障害の発生時にデータ保全性が保護されますが、複数のタスクが同じデータ・リソースを同時に操作しようとした場合に、パフォーマンスに影響を与えます。ただし、856 ページの『トランザクションを作業単位に分割する』で説明しているように、ロックがパフォーマンスに与える影響は、作業単位の短いアプリケーションを実装することで最小限に抑えられます。

### ファイルの暗黙ロック

このセクションでは、まず最初に、リカバリー不能ファイルの更新中に提供される暗黙ロックについて説明します。その後、リカバリー可能ファイルの更新中の拡張ロック・アクションについて説明します。

リカバリー不能ファイル:

リカバリー不能 (つまり、LOG=NO が FCT 項目で指定されている) BDAM ファイルの場合、CICS は、更新中のレコードをロックしません。

デフォルトでは、BDAM 排他制御を使用します。これは、物理ブロックで動作し、システム全体を制御しますが、持続するのは更新が完了するまでです。トランザクションは、BDAM 排他制御下で更新対象のレコードを読み取り、その後、データを変更しないことを決定した場合、BDAM 排他制御を解放する必要があります。これを行うには、EXEC CICS UNLOCK コマンドを発行します。これにより、CICS は RELEX マクロを発行します。

BDAM 排他制御が必要ない場合は、FCT のファイル項目で SERVREQ=NOEXCTL を指定します。

非 RLS モードでアクセスされたリカバリー不能 VSAM ファイルの場合、VSAM 排他制御により、更新中に制御インターバルがロックされます。RLS モードでアクセスされたリカバリー不能 VSAM ファイルの場合、SMSVSAM は更新中にレコードをロックします。

870 ページの図 142 は、リカバリー不能ファイルのロック範囲を示しています。

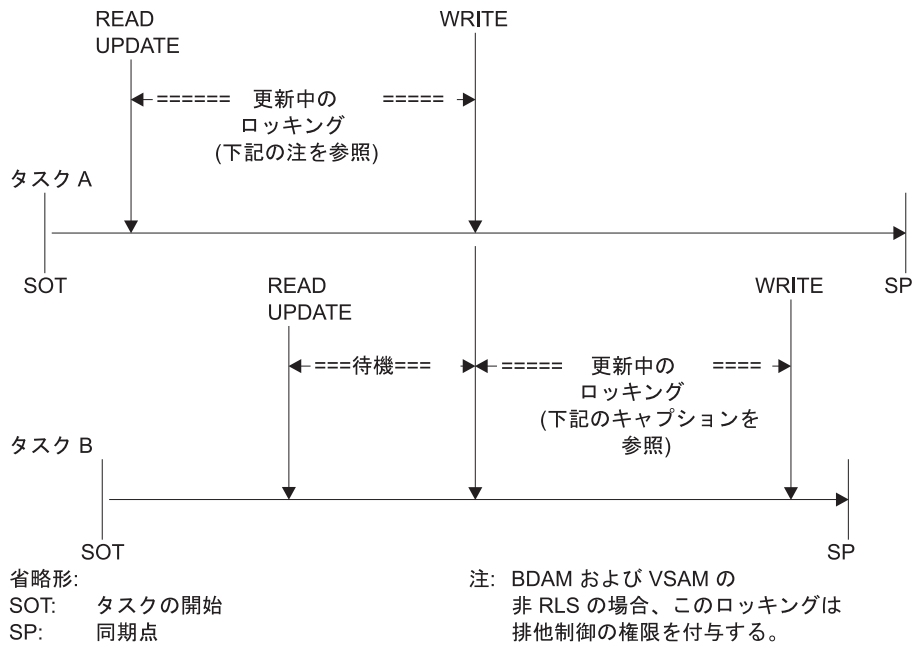


図 142. 更新中のリカバリー可能ファイルのロック： この図は、同じレコードまたは制御インターバルを更新する 2 つのタスクを示しています。タスク A には、READ UPDATE コマンドと WRITE コマンドの間で、レコードまたは制御インターバルのロックが付与されています。この期間中、タスク B は待機します。

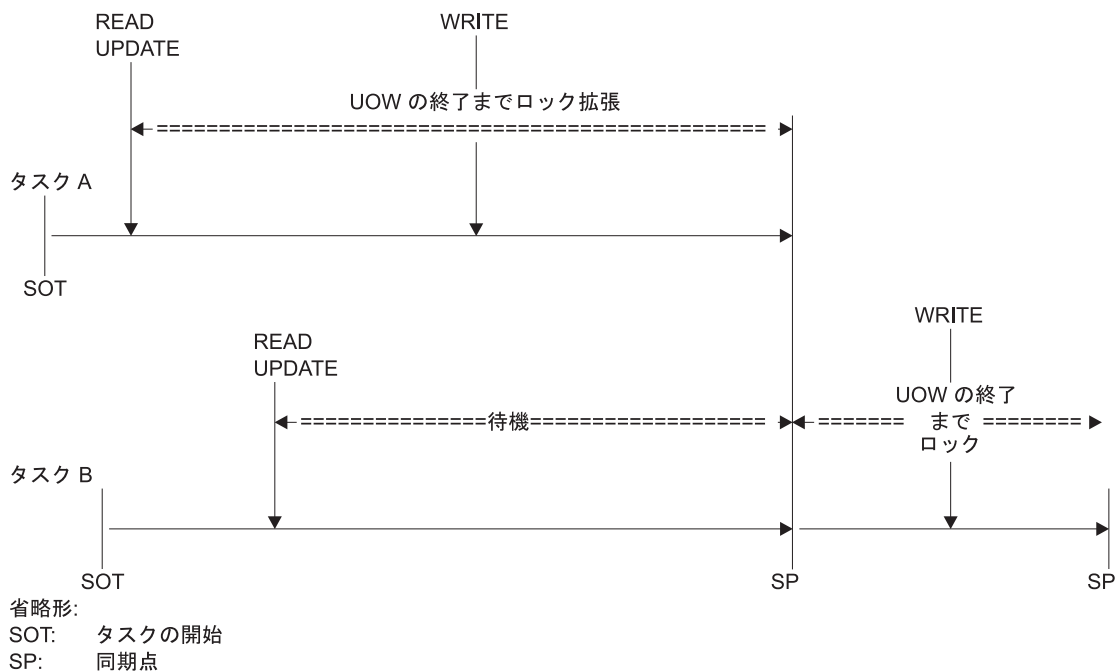


図 143. リカバリー可能ファイルの更新中のロック (リソースのエンキュー)： この図は、同じレコードまたは制御インターバルを更新する 2 つのタスクを示しています。タスク A には、更新が (UOW の終了時に) コミットされるまで、レコードの排他ロックが付与されています。この期間中、タスク B は待機します。

リカバリー可能ファイル:

リカバリー可能として指定された VSAM ファイルまたは BDAM ファイルの場合、ロック・アクションの期間が延長されます。VSAM ファイルの場合、拡張ロックは、制御インターバル全体ではなく、更新レコードでのみ行われます。

ロックの延長期間は、他のタスクによってバックアウトされるタスクが更新をコミットしないようにするために必要です。(870 ページの図 142 に示されている非拡張ロック・アクションがリカバリー可能ファイルの更新時に使用された場合に起こり得ることを考えてみてください。タスク B が変更をコミットし、同期点に達した直後にタスク A が異常終了した場合、タスク A の後続のバックアウトにより、ファイルはタスク A の開始時の状態に戻され、タスク B のコミット済み更新は失われます。)

この問題を回避するために、トランザクションがリカバリー可能ファイルを変更する (または更新前にリカバリー可能ファイルから読み取る) コマンドを発行するたびに、CICS は、変更がコミットされるまで (つまり、作業単位が終了するまで) 更新レコードを自動的にロックします。したがって、上記の例では、タスク A が作業単位の終了時にその変更をコミットするまで、タスク B はレコードにアクセスできませんでした。そのため、タスク B の更新がタスク A のバックアウトにより失われることはなくなります。非 RLS モードで開いたファイルの場合、CICS はエンキュー・ドメインを使用してこのロックを付与します。RLS モードで開いたファイルの場合、SMSVSAM がロックを付与し、CICS の要求に応じて作業単位の完了時にロックは解除されます。

このように自動ロックを起動するファイル制御コマンドは、以下のとおりです。

- READ (更新用)
- WRITE
- DELETE

注:

1. 前述のようにエンキューにより、トランザクション・デッドロックが発生することがあります (874 ページの『トランザクション・デッドロックの可能性』を参照)。
2. ロックの有効範囲は、アクセス方式、アクセスのタイプ、ロックの取得者によって異なります。
  - BDAM 排他制御は物理ブロックに適用されます。
  - 非 RLS VSAM 排他制御は制御インターバルに適用されます。
  - BDAM 用の CICS ロック (NOEXCTL を指定) はレコードにのみ適用されます。
  - 非 RLS VSAM 用の CICS ロックはレコードにのみ適用されます。
  - RLS 用の SMSVSAM ロックはレコードにのみ適用されます。
3. **VSAM 排他制御。**非 RLS モードで開いたリカバリー可能ファイルでの CICS エンキュー・アクションは、作業単位が終了するまで続きます。トランザクションが READ UPDATE コマンドを発行すると、レコードを含む制御インターバルの VSAM の排他制御は、CICS がそのレコードに対して ENQ を発行するのに十分な長さの期間だけ保持されます。CICS は、その後、CI の排他制御の解

放を VSAM に通知し、タスクが REWRITE (または UNLOCK、DELETE、SYNCPOINT) コマンドを発行するときのみ、これを再取得します。タスクでレコードの再書き込み準備ができるまで VSAM 排他 CI ロックを解放することで、トランザクション・デッドロックの可能性を最小限に抑えられます。

4. リカバリー可能ファイルの場合は、(代替キーで表される) 固有リソースを割り振るために固有キーの代替索引を使用しないでください。使用すると、以下の一連の状況でバックアウトが失敗する可能性があります。
  - a. タスクが (基本または別の代替索引を使用して) レコードを削除または更新し、代替索引キーが変更される。
  - b. 1 つ目のタスクの作業単位が終了する前に、2 つ目のタスクが元の代替索引キーで新規レコードを挿入するか、あるいは、既存の代替索引キーを元の代替索引キーのものに変更する。
  - c. 1 つ目のタスクが失敗し、バックアウトが試行される。

代替索引で重複キーが検出されてメッセージ DFHFC4701 が示され、X'F0' 障害コードも出されたため、バックアウトは失敗します。代替索引キーには、1 つ目のタスクの作業単位終了前の 2 つ目のタスクによるキーの取得を防ぐためのロックがありません。この操作に関するアプリケーション要件がある場合は、CICS エンキュー・メカニズムを使用して、作業単位が終了するまでにキーを予約する必要があります。

5. 読み取り中のデータが最新であることを確認するために、アプリケーション・プログラムは以下のことを行う必要があります。
  - 非 RLS モードでアクセスされたファイルの場合、(単なる READ ではなく) READ UPDATE コマンドを発行する。これにより、作業単位が終了するまでデータがロックされます。
  - RLS モードでアクセスされたファイルの場合は、整合性のある読み取り保全性オプションを使用する。

### 論理的にリカバリー可能な TD 宛先に対する暗黙エンキュー

リカバリー可能ファイルの場合と同様に、CICS は、(物理的にリカバリー可能なものではなく) 論理的にリカバリー可能な一時データ宛先に対して、エンキュー保護機能を提供します。

ただし、小さな違いが 1 つあります。CICS は、リカバリー可能な各宛先を、書き込み用と読み取り用の 2 つの別個のリカバリー可能リソースとみなします。

暗黙エンキューを起動する一時データ管理コマンドは、以下のとおりです。

- **WRITEQ TD**
- **READQ TD**
- **DELETEQ TD**

したがって、以下の例のようになります。

- タスクが特定の宛先に WRITEQ TD コマンドを発行すると、そのタスクはそのタスク (または作業単位) が終わるまで、その書き込みの宛先にエンキューされます。これによりタスクがエンキューされている間は、以下のようになります。
  - 同じ宛先に書き込もうとする別のタスクは、中断されます。

- 同じ宛先から読み取ろうとする別のタスクは、(現在完了していない作業単位で書き込まれているデータではなく) コミット済みのデータのみを読み取ることができます。
- タスクが特定の宛先に READQ TD コマンドを発行すると、そのタスクはそのタスク (または作業単位) が終わるまで、その読み取りの宛先にエンキューされます。これによりタスクがエンキューされている間は、以下ようになります。
  - 同じ宛先から読み取ろうとする別のタスクは、中断されます。
  - 同じ宛先に書き込もうとする別のタスクは、書き込みを許可され、タスク (または作業単位) が終わるまで書き込みの宛先にエンキューされます。
  - タスクが DELETEQ TD 要求を発行すると、そのタスクは読み取りと書き込みの両方の宛先にエンキューされます。これにより、タスクがエンキューされている間、他のタスクは、そのキューに対する読み取りまたは書き込みを行えません。

### リカバリー可能な一時記憶域キューに対する暗黙エンキュー

CICS は、VSAM データ・セットのリカバリー可能ファイルと同様の方法で、リカバリー可能一時記憶域キューに対するエンキュー保護機能を提供します。

ただし、小さな違いがあります。CICS のエンキューは、**READQ TS** コマンドを呼び出しません。そのため、あるタスクが一時記憶域キュー・レコードを読み取っている間も、別のタスクが同じレコードを更新することができます。これを防ぐためには、同時に実行中のタスクが同じ一時記憶 ID でキュー (複数可) の読み取りおよび変更を行うことが可能な一時記憶域キューに対して、明示エンキューを使用します。( 874 ページの『(アプリケーション・プログラマーによる) 明示エンキュー』を参照してください。)

暗黙エンキューを起動する一時記憶域管理コマンドは、以下のとおりです。

- WRITEQ TS
- DELETEQ TS

### DBCTL を使用する DL/I データベースに対する暗黙エンキュー

IMS™ プログラム分離スケジューリングは、タスクが DL/I データベース呼び出しでセグメントにアクセスすると、アクセスされたセグメントと同じデータベース・レコード内のすべてのセグメントに暗黙的にエンキューします。

エンキューされる期間は、使用されるアクセス方式によって異なります。

#### 直接方式 (HDAM、HIDAM)

セグメントに対して ISRT、DLET、または REPL 呼び出しが発行された場合、そのセグメントとすべての子セグメント (また、DLET 呼び出しの場合は親セグメントも) は、DL/I TERM 呼び出しが発行されるまで、エンキューされたままになります。タスクは、別のデータベース・レコードのセグメントにアクセスすることで、そのデータベース・レコードに含まれる他のすべてのセグメントからデキューされます。

#### 順次方式 (HSAM、HISAM、SHISAM)

タスクが任意のセグメントに対して ISRT、DLET、または REPL 呼び出しを発行した場合は、DL/I TERM 呼び出しが発行されるまで、データベース・レコード全体がエンキューされたままになります。ISRT、DLET、ま

たは REPL 呼び出しが発行されない場合、タスクは、別のデータベース・レコードのセグメントにアクセスすることで、そのデータベース・レコードからデキューされます。

上述のプログラム分離スケジューリングのルールは、セグメント検索指数で「Q」コマンド・コードを使用するか (このコマンドはエンキューを DL/I TERM 呼び出しの発行にまで拡張します)、または PCB で PROCOPT=EXCLUSIVE を使用することで (このマクロは、タスクが PSB をスケジュールに入れた期間全体について、指定されたセグメント・タイプの排他制御を指示します)、指定変更されます。

### (アプリケーション・プログラマーによる) 明示エンキュー

CICS はエンキュー・コマンドを提供します。それらのコマンドは、データを保護し、トランザクション・デッドロックを防ぐために、アプリケーション内で役に立ちます。

CICS に用意されている明示エンキュー・コマンドは以下のとおりです。

- **EXEC CICS ENQ RESOURCE**
- **EXEC CICS DEQ RESOURCE**

こうしたコマンドを使用して、以下の機能を実行できます。

- 共通作業域 (CWA) に書き込まれるデータを保護する。CICS では、それらのデータは自動的に保護されません。
- 複数のタスクで同時に更新する可能性のあるレコードをエンキューすることにより、トランザクションのデッドロックを防止する。
- 同時の読み取りおよび更新から、一時記憶域キューを保護する。

しかし、有効にするためには、すべてのトランザクションが同じ規則に従う必要があります。決められた ENQ コマンドと DEQ コマンドを使用せずに CWA にアクセスするトランザクションは、中断されず、保護が破られます。

タスクが ENQ RESOURCE(*data-area*) コマンドを発行すると、同じデータ領域パラメーターを指定して ENQ RESOURCE コマンドを発行する他のタスクはすべて、対応する DEQ RESOURCE(*data-area*) コマンドをそのタスクが発行するか、その作業単位が終了するまで、中断されます。

注: 複数のリソースに同時にエンキューを行うと、トランザクション間にデッドロックが生じる可能性があります。

### トランザクション・デッドロックの可能性

二重更新からリソースを保護する、エンキューおよびプログラム分離スケジューリングのメカニズムは、トランザクション・デッドロックとして知られる状態を引き起こす可能性があります。

875 ページの図 144で示しているように、トランザクション・デッドロックは、別のタスクによってエンキューされているリソースの解放を各タスクが待っているために、複数のタスクを実行できないことを意味します。(エンキュー、DL/I プログラム分離スケジューリング・アクション、または VSAM RLS ロック・アクションは、次の同期点に達するまでリソースを保護します。)

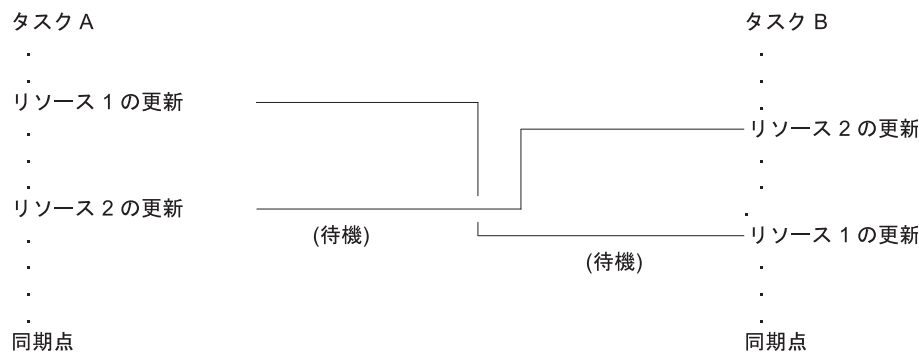


図 144. トランザクション・デッドロック (一般)

トランザクション・デッドロックが発生すると、一方のタスクは異常終了し、もう一方のタスクは続行します。

- トランザクション・デッドロック検出は、DTIMOUT トランザクション属性で制御されます。指定された間隔の間タスクが中断状態 (非アクティブ) のまま、SPURGE(YES) も指定されている場合、CICS はそのタスクの異常終了を開始します。CICS 提供のミラー・トランザクション CSMI、CSM1、CSM2、CSM3、および CSM5 には、DTIMOUT(NO) と SPURGE(NO) のデフォルト設定があります。
- デッドロックされたリソースがどちらも VSAM RLS リソースの場合、デッドロックの検出は VSAM によって行われます。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。

注: 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。VSAM は、DTIMOUT パラメーターまたは FTIMEOUT パラメーターによって定義されている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、リソース間のデッドロックを解決します。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。

- リソースがどちらも DL/I データベースの場合は、スケジューリング呼び出しを発行するタスクの結果として、DL/I 自体が潜在的なデッドロックを検出します。この場合、DL/I は CICS に、更新アクティビティーが最も少ないタスクを異常終了させます (異常終了コード ADCD)。
- デッドロックされたリソースがどちらも CICS リソースの場合 (ただし、どちらも VSAM リソースではない場合)、または一方が CICS でもう一方は DL/I の場合、CICS は、DTIMOUT 期間が先に終了したほうのタスクを異常終了させます。両方のタスクが同時にタイムアウトする可能性もあります。どちらのタスクにも DTIMOUT 期間が指定されていない場合は、どちらか一方が発信端末コマンドによって取り消されない限り、両方とも無期限に中断状態のままになります。

異常終了したタスクは、その後、トランザクション・バックアウトで説明しているように、動的トランザクション・バックアウトによってバックアウトされることがあります。(特定の条件においては、トランザクション・バックアウトで説明しているように、トランザクションが自動的に再始動する可能性があります。あるいは、端末オペレーターが、異常終了したトランザクションを再始動させる場合もあります。)

詳しくは、860 ページの『トランザクションのデッドロックを回避する設計』を参照してください。

## トランザクション・バックアウトのユーザー出口

動的トランザクション・バックアウト中、および緊急時再始動でのバックアウト中に実行されるグローバル・ユーザー出口プログラムに、独自のロジックを組み込むことができます。

ファイル制御リカバリー管理プログラム、およびユーザー・ログ・レコード・リカバリー・プログラムには出口があります (緊急時再始動時にのみ駆動される)。一時データおよび一時記憶域のバックアウトには出口はありません。

### 独自のコードを追加できる場所

緊急時再始動時に、プログラム・リスト・テーブルに指定する事後初期化プログラムに独自のコードを追加することができます。

#### このタスクについて

緊急時再始動中に実行されるグローバル・ユーザー出口プログラムに以下の機能を組み込むことができます。

- バックアウトされるファイル更新の詳細を含むファイル制御ログ・レコードの処理
- 論理的に削除されたものとしてレコードにフラグを付けることによる、削除をサポートしないデータ・セット・タイプ (VSAM ESDS および BDAM) への追加のバックアウト処理
- トランザクション・バックアウト中に発生するファイル・エラー状態の処理
- ユーザー・リカバリー・レコードの処理 (緊急時再始動中のユーザー・ログ・レコード・リカバリー・プログラムの XRCINPT 出口において)
- 非 RLS バッチ・プログラムが RLS 保持ロックをオーバーライドした場合の処理 (これは頻繁に発生すべきものではありません)

使用できる出口は以下のとおりです。

1. XRCINIT。ユーザー・ログ・レコード・リカバリー・プログラムの始めと終わりに開始されます。
2. XRCINPT。システム・ログからユーザー・ログ・レコードが読み取られるたびに開始されます。
3. XFCBFAIL。ファイル・バックアウト障害出口です。
4. XFCLDEL。ファイル論理削除出口です。
5. XFCBOVER。ファイル・バックアウトの非 RLS オーバーライド出口です。
6. XFCBOUT。ファイル・バックアウト出口です。

デフォルトの処置を望まない場合には、これらの出口のいずれかを使用して独自の処理を追加できますが、出口タスクはパージできないため、これらの出口には UERCPURG 戻りコードを設定しないでください。緊急時再始動時にこれらの出口を使用するには、以下の手順を使用します。

### 手順

- PLT 処理の最初の部分で PLT プログラムの出口を有効にします。
- システム初期設定パラメーター **TBEXITS** に出口を指定します。この場合、TBEXITS=(name1,name2,name3,name4,name5,name6) 形式を使用します。ここで、name1、name2、name3、name4、name5、および name6 は、XRCINIT、XRCINPT、XFCBFAIL、XFCLDEL、XFCBOVER、XFCBOUT 出口点のグローバル・ユーザー出口プログラムの名前です。

### 次のタスク

出口の汎用インターフェースのプログラミング情報、出口プログラムの作成方法、および出口ごとの入力パラメーターと戻りコードについては、Global user exit programsを参照してください。

### XRCINIT 出口

XRCINIT は、次のように、ウォーム・リスタートおよび緊急時再始動時に呼び出されます。

1. システム・ログから最初のユーザー・ログ・レコードが読み取られ、そのレコードが XRCINPT 出口に渡される前
2. システム・ログから最後のユーザー・ログ・レコードが読み取られ、XRCINPT に渡された後

XRCINIT 出口コードは、必ず UERCNORM の戻りコードで終了する必要があります。この出口に対して選択可能な処理オプションはありません。

CICS は、XRCINPT 出口に示されるユーザー作成のログ・レコードに関する情報を、グローバル・ユーザー出口パラメーター・リストに入れて渡します。パラメーターには、ユーザー・ログ・レコードの後処理を示すフラグ・バイトが含まれます。これは、検出されたユーザー・ログ・レコードが含まれていた作業単位の状態を示すか、またはそのレコードがアクティビティー・キーポイント・レコードであることを示すことができます。可能な値は、以下のことを示します。

- そのレコードはアクティビティー・キーポイント・レコードです。
- UOW がコミットされました。
- UOW がバックアウトされました。
- UOW が未完了です。
- UOW が未確定です。

### XRCINPT 出口

XRCINPT は、ウォーム・リスタート時および緊急時再始動時に、システム・ログから読み取られるユーザー・ログ・レコードごとに一度呼び出されます。

デフォルト・アクションでは、この出口では何も行われません。

ログ・レコードを無視する場合は、戻りコード UERCBYP で戻るようにします。これにより、レコード域が即時に解放され、システム・ログから新規レコードが読み取られます。このアクションでデータ保全性がリスクにさらされないように注意してください。

### **XFCBFAIL グローバル・ユーザー出口**

XFCBFAIL は、作業単位のバックアウト中にエラーが発生するたびに呼び出されます。

XFCBFAIL グローバル・ユーザー出口プログラムは、CICS バックアウト障害制御をバイパスするかあるいは呼び出すかを決定できます。CICS バックアウト障害制御によって実行される処理については、バックアウト障害からのリカバリーで説明されています。

### **XFCLDEL グローバル・ユーザー出口**

XFCLDEL は、VSAM ESDS、または BDAM データ・セットへの書き込み操作を実行した作業単位をバックアウトするときに呼び出されます。

### **XFCBOVER グローバル・ユーザー出口**

XFCBOVER は、CICS がコミットされていない更新はバックアウトしないことを決定しようとするたびに呼び出されます。これは、レコードが非 RLS バッチ・プログラムによって更新された可能性があるためです。

この状態は、ロックが保持されている場合でも、バッチ・プログラムが RLS データ・セット保護をオーバーライドして、データ・セットを開いた後に発生することがあります。

### **XFCBOUT グローバル・ユーザー出口**

XFCBOUT は、CICS がファイルの更新をバックアウトしようとしているときに呼び出されます。

### **トランザクション・バックアウト出口のコーディング**

出口の実行中は、端末管理サービス以外のすべての CICS サービスにアクセスできません。

#### **このタスクについて**

ただし、以下の制約事項を考慮してください。

- 出口プログラムはアセンブラー・コードで書かれている必要があります。
- それらは準再入可能である必要があります。それらは、出口プログラミング・インターフェース (XPI) を使用し、EXEC CICS コマンドを発行できます。
- 出口プログラムがファイル制御要求の結果として領域を獲得する場合、その領域の解放はその出口が行います。
- 出口は、RLS 以外のモードで開かれている、ストリング番号 1 の VSAM データ・セットを参照するファイルへのファイル制御要求を作成しようとはできません (出口の初期設定中に、そのファイルに対するアクションが指定されていない場合)。

- 出口内で獲得された、タスクにチェーニングされているストレージは、その内容が不要になったら、その出口によってすぐに解放される必要があります。
- 出口が使用されていない場合は、デフォルトのアクションが実行されます。
- 緊急時再始動のグローバル・ユーザー出口がリカバリー可能 リソースを変更しないようにすることを強くお勧めします。一時記憶域、一時データ、またはファイル制御を使用する場合は、これらのリソース・マネージャーもリカバリー状態になる可能性があります。したがって、これらのサービスにアクセスすることは、最良の場合でリカバリー・タスクの逐次化を引き起こし、最悪の場合はデッドロックを引き起こします。



---

## 第 13 章 変換およびコンパイル

古いコンパイラー (およびアセンブラー)の中には CICS コマンドを直接処理できないものもあります。プログラムを実行可能コードに変換するための追加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。

ほとんどのコンパイラーでは統合 CICS 変換プログラムを使用し、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します。統合 CICS 変換プログラム方式を使用すると、変換タスクの多くはコンパイル時に実行されるので、変換ステップを追加で実行しなくても済みます。変換ステップにおけるタスクについて詳しくは、884 ページの『変換のプロセス』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。

- 『統合 CICS 変換プログラム』
- 884 ページの『変換のプロセス』
- 887 ページの『CICS 提供の変換プログラム』
- 899 ページの『CICS 変換プログラムの使用』
- 901 ページの『変換プログラムのオプションの定義』
- 903 ページの『COPY ステートメントの使用』
- 903 ページの『CICS 提供のインターフェース・モジュール』
- 904 ページの『AMODE(24) および AMODE(31) アプリケーションでの 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に変換し、その言語に対応する CALL ステートメントを生成します。

CICS オンライン・プログラムをコンパイルする場合、以下の言語用の統合変換プログラムのバージョンがあります。

C  
C++  
COBOL  
PL/I

さらに、外部 CICS インターフェース (EXCI) コマンド・レベル API を使用する COBOL、C、C++、および PL/I のバッチ・プログラムをコンパイルする場合にも、統合変換プログラムを使用できます。

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・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呼び出しでは中間リストを必要とするという意味であり、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する必要があります。

統合変換プログラムを使用すると、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なりました。

言語環境プログラムに準拠した、統合変換プログラムをサポートする言語コンパイラは、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。

統合変換プログラムをサポートする言語コンパイラのリリースは、アプリケーション・プログラミング言語に関する CICS サポートの変更点にリストされています。その他のコンパイラまたはアセンブラを使用する場合は、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

言語コンパイラでは、統合 CICS 変換プログラムで利用できるさまざまなプロシージャを提供しています。それらのプロシージャは、Enterprise COBOL for z/OS、z/OS XL C/C++、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では、コンパイルのステップで STEPLIB 連結に CICSTS55.CICS.SDFHLOAD を追加する必要があります、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・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・オプション CICS、NODYNAM、および RENT を有効にする必要があります。NODYNAM は、統合変換プログラムに固有の制限ではありません。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。

注: コンパイラの LIB オプションは、COBOL 5 以降では不要になり、除去されました。以前のバージョンの COBOL の場合は、このオプションを手動で復元する必要があります。

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 以上を実行しており、統合変換プログラムを含むコンパイラを使用して COBOL プログラムを準備している場合、コンパイラでは SQL ステートメント・コプロセッサ (これにより、DBRM が生成されます) も提供されるため、個別の Db2 プリコンパイラを使用する必要はありません。SQL ステートメント・コプロセッサの使用方法について詳しくは、CICS Db2 プログラムの準備および Db2 for z/OS 製品資料内の『Db2 for z/OS のプログラミング』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

PL/I、COBOL、XL C、C++ のいずれかのコンパイラを使用している場合、CICS 変換プログラムのオプションを指定できます。

### このタスクについて

すべての変換プログラム・オプションについての説明は、901 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には適用されません。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、PL/I ではサポートされませんが、EXCI コマンド・レベル API を使用する COBOL、C、および C++ のバッチ・プログラムではサポートされます。

統合 CICS 変換プログラムで有効な変換プログラム・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

## 手順

- 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。以下に例を示します。

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

PL/I コンパイラー・オプションの指定の詳細については、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。以下に例を示します。

```
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・オプションの指定の詳細については、Enterprise COBOL for z/OS プログラミング・ガイドを参照してください。

- 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。以下に例を示します。

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

別の方法として、XOPTS キーワードまたは 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・オプションを指定することもできます。

C および C++ の各コンパイラー・オプションの指定について詳しくは、z/OS XL C/C++ ユーザーズ・ガイドを参照してください。

---

## 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言語ごとに変換プログラムを提供し、EXEC CICS、EXEC CPSM、および EXEC DLI 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、コンパイラーは直接 CICS 変換プログラムを呼び出して、これらのコマンドの変換を処理します。

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。図 145 には、この 3 つのステップが説明されています。

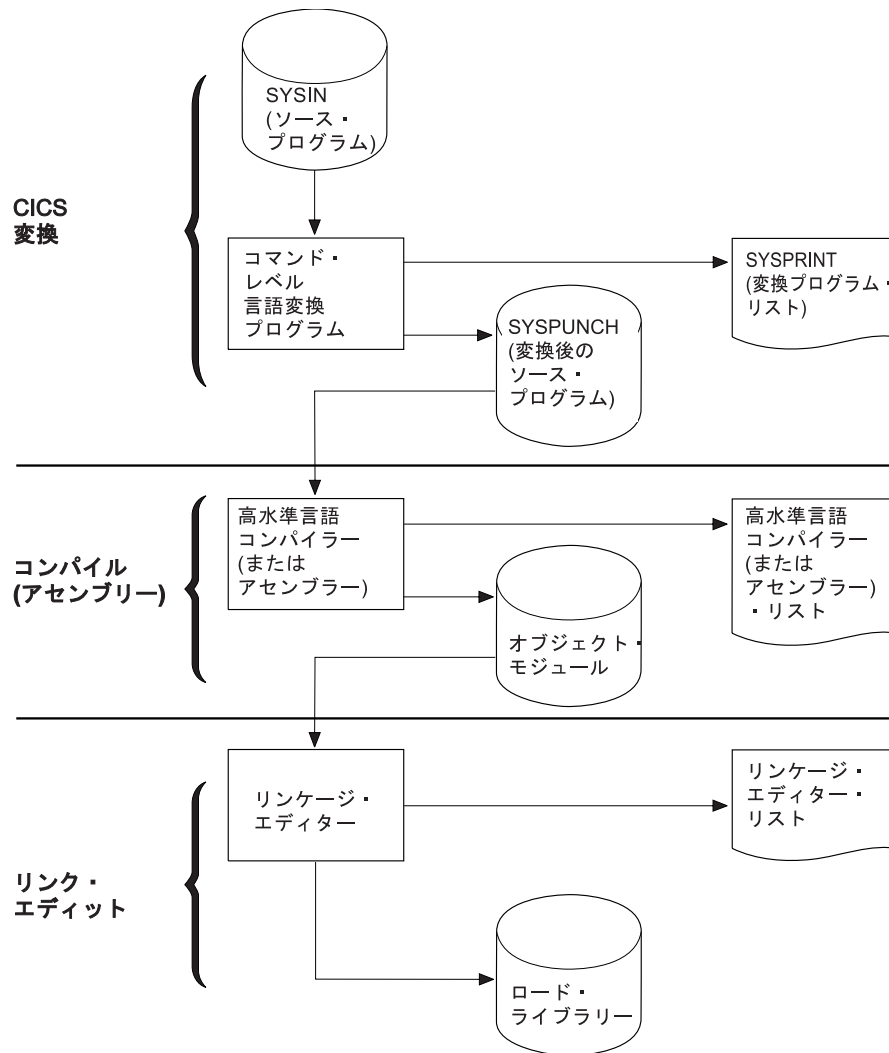


図 145. アプリケーション・プログラムの準備

システム管理者が、制限付きコマンド parmlib メンバー DFHAPIR 内の制限付き CICS API および制限付き SPI コマンドを識別する規則を定義している場合、変換時には、CICS 変換プログラムにより、制限付きコマンドとキーワードについてもソース・プログラムが検査されます。違反している場合、CICS 変換プログラムにより警告メッセージまたはエラー・メッセージが発行され、変換が失敗する可能性があります。詳しくは、特定の CICS API コマンドと SPI コマンドの使用を制御するを参照してください。

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・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になります。

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。1011 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、903 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

CICS 提供の変換プログラムは、CICSTS55.CICS.SDFHLOAD ライブラリーにインストールされています。

このライブラリー内の変換プログラムは次の表のとおりです。

表 81. 言語別の CICS 提供変換プログラム

言語	変換プログラム
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

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。

## このタスクについて

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・オプション

コマンド・レベルの言語変換プログラムには、ユーザーが作成する COBOL、C、C++、PL/I、およびアセンブラー・プログラムを変換するために選択できる一連のオプションがあります。ここでは、それぞれの変換プログラム・オプションについて説明し、それらのオプションと共に使用できる言語を紹介します。

### APOST

(COBOL のみ)

APOST は、リテラルをアポストロフィまたは引用符で区切ることを指定します。代替オプションは QUOTE であり、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。

提供されている COBOL コピーブックは、このオプションを使用して、単一引用符を含めて生成されます。提供されているコピーブックを、CICS コンポーネントとのインターフェースを確立するためにアプリケーションで使用する場合には、必ず、QUOTE オプションではなく、APOST オプションを設定してください。

### 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ように指定します。CBLCARD はデフォルトです。変更する場合は NOCBLCARD を指定します。COBOL コンパイラー LIB オプションは、COBOL バージョン 5 以上では必要ありません。CBLCARD 変換プログラム・オプションの結果としてのコンパイラー・オプションには、LIB オプションは含まれません。バージョン 5 より前の COBOL コンパイラーによって処理されるソースを変換する場合は、CBLCARD に依存せずに LIB オプションを指定する必要があります。

### 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示す唯一の方法は、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、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。COBOL2 と COBOL3 の両方を指定すると、変換プログラムは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。言語環境プログラムに準拠したどのコンパイラーが利用可能かについて詳しくは、679 ページの『第 9 章 プログラミング言語と言語環境プログラム (Language Environment)』を参照してください。

## **CPP**

(C++ のみ)

CPP は、変換プログラムで、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。代替オプションは 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。DBCS 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 を使用して COBOL プログラムを作成する方法について詳しくは、「Enterprise COBOL for z/OS 言語解説書」で DBCS 文字ストリングに関する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。NODEBUG は代替オプションです。

## 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。代替オプションは NOEDF です。

## EPILOG

(アセンブラ言語のみ)

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 を指定します。

EPILOG はデフォルトです。代わりに NOEPILOG を指定すると、変換プログラムでマクロ DFHEIRET が挿入されなくなります。(DFHEIRET マクロに関するプログラミング情報については、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。EXCI API 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 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。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、FEPI API を使用した開発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。これらのコマンドのプログラミング情報については、CICS API コマンド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。GRAPHIC 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。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、また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、意図しないシフトアウトおよびシフトイン・コードを防ぎます。

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**

**(アセンブラー言語のみ)**

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。LEASM オプションにより、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カスタマー・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 (Language Environment) 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はサポートされないため、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はこのカテゴリーに該当します。

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、908 ページの『LEASM を使用するアセンブラー言語プログラムのサンプル』を参照してください。

## **LENGTH**

**(COBOL、アセンブラー言語、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、MARGINS 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、以下のとおりです。

**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・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。

- |           |            |
|-----------|------------|
| <b>CS</b> | 中国語 (簡体字)。 |
| <b>EN</b> | (デフォルト) 英語 |
| <b>KA</b> | 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・オプションは、DFHYITVL プロシージャが指定します。RENT および NODYNAM が指定されていることを確認してください。

#### **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。NOCPSM 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 を指定します。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。または、EPILOG (デフォルト) を指定することもできます。

## **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、アセンブラー言語、および PL/I のみ)

アプリケーション・プログラムの CICS コマンドから LENGTH オプションが省略された場合、NOLength は、変換プログラムに対して、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。

NOLINKAGE オプションは、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ることを意味します。

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。代替オプションの 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 は、IBM サポート・センターから使用を求められた場合にのみ使用してください。

#### **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、代替オプションは VBREF です。

#### **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 オプションに許可される最大 *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 値のデフォルトが許可されている場合、使用されるコンパイラの入力マージンを変更する必要はありません。

#### **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* と *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・ステップには、同一の値を指定しなければなりません。

提供されている COBOL コピーブックが単一引用符 (APOST) を使用して生成されます。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

ターフェイス用に 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 を指定してください。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、以下のとおりです。

**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。これらのコマンドは、システム・プログラマーによって使用されます。これらのコマンドのプログラミング情報については、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 を指定してください。

### 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。このオプションは、それを使用するプログラムに制限を課すため、特別なシチュエーションでのみ使用してください。SYSEIB オプションで変換されたプログラムでは、以下のことが必要です。

- 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 オプションを使用すると、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します。コマンドには、必要に応じて RESP オプションを使用できます。

### 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 と呼ばれていましたが、これまでどおり受け入れられます。

### 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

言語変換プログラムは、ソース・プログラムを読み取り、新しいプログラムを作成します。ほとんどの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する言語で必要とされる形式の CALL ステートメントに変換されます。

呼び出しによって、CICS 提供「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・モジュールは後でロード・モジュールにリンク・エディットされ、実行時には、今度はそれらのインターフェース・モジュールが、要求されたサービスを呼び出します。

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。901 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。

選択するオプションは、以下の方法で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 (アセンブラ) 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。

表 82. 各言語について提供されているステートメント

言語	ステートメント
COBOL	CBL <sup>注</sup>
COBOL	PROCESS
C	#pragma
C++	#pragma
PL/I	* PROCESS
アセンブラ	*ASM または *PROCESS <sup>注</sup>

- 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。

ほとんどのインストール・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ため、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、プロシージャー名を DFHYITVL、そのプロシージャー内の変換ステップの名前を TRN とすると、次のようなステートメントを使用して、COBOL プログラムの変換プログラム・オプションを設定します。

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

あるメソッドを使用してオプションを指定し、別のメソッドを使用して同じオプションまたは競合するオプションを指定した場合は、言語ステートメントの指定が EXEC ステートメントの指定に優先します。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

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 RENT NODYNAM 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)
```

次の形式は無効です。

```
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:

**\*ASM および \*PROCESS**

1. アセンブラー言語プログラムの場合、\*ASM ステートメントには変換プログラム・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。

**CBL** COBOL コンパイラ LIB オプションは、COBOL バージョン 5 以上では必要ありません。CBLCARD 変換プログラム・オプションの結果としてのコンパイラ・オプションには、LIB オプションは含まれません。バージョン 5 より前の COBOL コンパイラによって処理されるソースを変換する場合は、CBLCARD に依存せずに LIB オプションを指定する必要があります。

## 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。

表 83に、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を配布したままの形式で使っている場合には、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 ( 887 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、899 ページの『CICS 変換プログラムの使用』の説明のとおりオプションを指定する必要があります。

## 変換プログラムのオプション・テーブル

変換プログラムのオプション、および変換プログラムの適用対象プログラム言語が、表形式で示されています。

表 83. 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				

表 83. 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
CPP			X		
CPSM または <b>NOCPSM</b>	X	X	X	X	X
DBCS	X				
<b>DEBUG</b> または NODEBUG	X	X	X	X	
DLI	X	X	X	X	X
<b>EDF</b> または NOEDF	X	X	X	X	X
<b>EPILOG</b> または NOEPILOG					X
EXCI	X	X	X	X	X
FEPI または <b>NOFEPI</b>	X	X	X	X	X
FLAG (I または <b>W</b> あるいは E または S)	X	X	X	X	
GDS		X	X		X
GRAPHIC				X	
LEASM					X
<b>LENGTH</b> または NOLENGTH	X			X	X
LINECOUNT(n)	X	X	X	X	X
<b>LINKAGE</b> または NOLINKAGE	X				
MARGINS(m,n)		X	X	X	
NATLANG	X	X	X	X	X
NUM または <b>NONUM</b>	X				
OPMARGINS(m,n[,c ])		X	X	X	
<b>OPSEQUENCE</b> (m,n) または NOOPSEQUENCE		X	X	X	
<b>OPTIONS</b> または NOOPTIONS	X	X	X	X	X
<b>PROLOG</b> または NOPROLOG					X
QUOTE または <b>APOST</b>	X				

表 83. 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
SEQ または NOSEQ	X				
SEQUENCE(m,n) または NOSEQUENCE		X	X	X	
SOURCE または NOSOURCE		X	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 (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 (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、コピーブックのソースに CICS コマンドが含まれている場合には、そのコピーブック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、コピーブックのソースに CICS コマンドが含まれている場合は、そのコピーブックが組み込まれるプログラムをコンパイルする前に、コピーブック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、組み込まれるコピーブック内にすべての 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 の下で実行されるアプリケーション・プログラムには、CICS 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。

アプリケーション・プログラムは、以下の機能を使用するために、スタブを必要とします。

- 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

## EXEC インターフェース・モジュール

ご使用の CICS アプリケーションには、それぞれ、CICS とのインターフェースが含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。CICSTS55.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。CICSTS55.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・リカバリーとのインターフェースが含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。CICSTS55.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。

---

## AMODE(24) および AMODE(31) アプリケーションでの EXEC インターフェース・モジュールの使用

AMODE(24) および AMODE(31) アプリケーションの場合、言語変換プログラムによって生成される CALL ステートメントで、ユーザー・コードと DFHEIP (CICS EXEC インターフェース・プログラム) との間の通信を実現する EXEC インターフェース・モジュールが呼び出されます。

言語変換プログラムは、ソース・プログラムを読み取り、新しい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する言語で必要とされる形式の CALL ステートメントに変換されます。呼び出しによって、CICS 提供「EXEC」インターフェース・モジュールまたはスタブ が起動されます。スタブとは、CICS 高水準プ

プログラミング・インターフェースで使用するコードの機能依存セクションです。スタブは、SDFHLOAD ライブラリーで提供され、アプリケーション・プログラムとリンク・エディット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ミング言語のスタブが提供されています。

表 84. インターフェース・モジュール (スタブ)

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 DFHEAIO
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。したがって、これらのスタブ・モジュールについては以前および以降のバージョンとの互換性が確保されるので、以降のレベルのスタブを組み込むために 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。

DFHEAIO を除き、これら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。そのために、スタブは、変換された EXEC CICS コマンドから呼び出されるさまざまなエントリー・ポイントを提供してから、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。

それぞれのスタブは、DFHYxxxx という形式の 8 バイトの目印で始まります。x は、スタブがサポートしている言語を示します (例えば、A はアセンブラーを示し、I はスタブが言語非依存であることを示します)。nnn は、スタブの組み込み元の CICS のリリースを示します。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 という形式の目印が付いています。文字 E 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 5 リリース 5 における DFHELII の目印は DFHYI 720 です。

目印は、CICS アプリケーション・ロード・モジュールが最後にリンクされた CICS リリースを判別する際に役立ちます。

## 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'0e0800000600001000'
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、変換プログラムで単純な **EXEC CICS RETURN** コマンドを処理する場合に生成され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 ASSEMBLER) ENTRY(*,CHAR(4));
CALL DFHENTRY_B62D3C38_296F2687('xxxxxxxxxxxxxxxx' /* '0E 08 00 00 03
00 00 10 00 F0 F0 F0 F0 F0 F0 F1 F0 'X */ , NEXT);
END;
```

上記の例では、DFHENTRY\_B62D3C38\_296F2687 は、実際の入り口 DFHEI01 に関連付けられている入り口変数 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%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 マクロの呼び出しに変換されます。

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
DFHECALL =X'0E0800000800001000'
```

この DFHECALL マクロ呼び出しのアセンブリーは、レジスター 1 によってアドレスリングされるパラメーター・リストを作成するコードを生成し、項目 DFHEI1 のアドレスをレジスター 15 にロードし、BALR 命令を発行してスタブ・ルーチンを呼び出します。

```
DS 0H
LA 1,DFHEITPL
LA 14,=X'0E0800000800001000'
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 5 リリース 5 の DFHEAI の目印は、DFHYA720 と、このスタブが CICS Transaction Server for z/OS, バージョン 5 リリース 5 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。

リンク・エディットからの出力で、プログラムの先頭に DFHEAI スタブが含まれる必要があります。そのためには、JCL のリンク・エディット・ステップ内に DFHEAI に関する ORDER および INCLUDE ステートメントが含まれる必要があります。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・エディットするために、SDFHPROC ライブラリー内の CICS 提供アセンブラー・プロシージャー DFHEITAL を使用するとき、このプロ

シージャーの COPYLINK ステップは SDFHMAC(DFHEILIA) をコピーします。  
組み込まれる必要のある以下のステートメントが DFHEILIA に含まれます。

```
ORDER DFHEAI
INCLUDE SYSLIB(DFHEAI)
```

これらのステートメントは、プロシージャーの LKED ステップでアセンブルされる  
アプリケーション・プログラムの前に連結される一時ファイルの中に配置されま  
す。

## LEASM を使用するアセンブラー言語プログラムのサンプル

CICS アセンブラー・プログラム、およびそのプログラムの変換後の対応するコード  
のサンプルに、LEASM オプションを使用してアセンブラー・プログラムを変換し  
た場合の結果が示されています。この変換プログラム・オプションにより、  
Language Environment 準拠のアセンブラー MAIN プログラムのコードが生成さ  
れます。

図 146 に、単純な CICS アセンブラー言語プログラムを示します。

```
*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 146. 簡単な CICS アセンブラー言語プログラム。

下記のコードは、プログラムが変換されてアセンブルされた後の結果を示していま  
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 HDDHDM : 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 0CYDDD+ 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
EIBCPASN 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
EIBSRCE 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 5001D @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-CEEN
OI 8(14),X'80' 02-CEEN
BALR 3,0 @01A 02-CEEN
USING *,3 @01A 02-CEEN
L 3,CEE0EPV0007 @01A 02-CEEN
POP USING @01A 02-CEEN
USING TESTLE,3 @01A 02-CEEN
L 1,CEEDSANAB-CEEDSA(,13) Get the current NAB 02-CEEN
L 0,CEEY0007 02-CEEN
ALR 0,1 Compute new value. 02-CEEN
CL 0,CEECAAEOS-CEECAA(,12) Compare with EOS. 02-CEEN

BNH CEEY0007 02-CEEN
L 15,CEECAAGETS-CEECAA(,12) Get address overflow routine 02-CEEN
BALR 14,15 Get another stack segment. 02-CEEN
LR 1,15 02-CEEN
B CEEY0007 Branch around statics @01A 02-CEEN
CEEINPL0007 DC A(CEEINPL) @01A 02-CEEN
CEEINT0007 DC V(CEEINT) @01A 02-CEEN
CEE0EPV0007 DC A(TESTLE) @01A 02-CEEN
CEEY0007 EQU * 02-CEEN
ST 13,CEEDSABKC-CEEDSA(,1) Set back chain. 02-CEEN
ST 0,CEEDSANAB-CEEDSA(,1) Set new NAB value 02-CEEN
XC CEEDSAFLAGS-CEEDSA(,1),CEEDSAFLAGS-CEEDSA(1) . Clear 02-CEEN
ST 1,CEEDSAFWC-CEEDSA(,13) Set forward chain. 02-CEEN
LR 13,1 Set save area address 02-CEEN
USING CEEDSA,13 Addressability to SF V1R2M0 02-CEEN
MVC CEEDSALWS,CEECAALWS-CEECAA(12) Get LWS addr V1R2M0 02-CEEN
LR 1,2 02-CEEN
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'040200000800000014000040000000'

DS 0H 01-DFHEC
LA 1,DFHEIPL 01-DFHEC
LA 14,=X'040200000800000014000040000000' 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'040200000800000014000040000000'
=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) . Member 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-CEEP
DC AL1(0) Sub ID 02-CEEP
DC AL1(0) Member defined 02-CEEP
DC AL1(1) Level of PPAX control blocks 02-CEEP
PPA2S0018 DC A(CEESTART) A(CEESTART for this load module) 02-CEEP
DC A(0) A(Compile Debug Information (CDI)) 02-CEEP
DC A(CEETIMES-PPA20018) A(Offset to time stamp) 02-CEEP
PPA2M0018 DC A(TESTLE) . A(first entry point in comp. unit) 02-CEEP
*

* T I M E S T A M P *

*
* Time Stamp
*,Time Stamp = 2004/06/17 08:51:00 02-CEEP
*,Version 1 Release 1 Modification 0 02-CEEP
CEETIMES DS 0F 02-CEEP
DC CL4'2004' Year 02-CEEP
DC CL2'06' Month 02-CEEP
DC CL2'17' Day 02-CEEP
DC CL2'08' Hours 02-CEEP
DC CL2'51' Minutes 02-CEEP
DC CL2'00' Seconds 02-CEEP
DC CL2'1' Version 02-CEEP
DC CL2'1' Release 02-CEEP
DC CL2'0' Modification 02-CEEP

* 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

```

---

## AMODE(64) アプリケーションでの EXEC インターフェース・モジュールの使用

非 Language Environment AMODE(64) アセンブラ言語プログラムの場合、言語変換プログラムによって生成される CALL ステートメントで、ユーザー・コードと DFHEIG (CICS EXEC インターフェース・プログラム) との間の通信を実現する EXEC インターフェース・モジュールが呼び出されます。

言語変換プログラムは、ソース・プログラムを読み取り、新しい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する言語で必要とされる形式の CALL ステートメントに変換されます。呼び出しによって、CICS 提供「EXEC」インターフェース・モジュールまたはスタブ が起動されます。スタブとは、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コードの機能依存セクションです。これらのスタブは、SDFHLOAD ライブラリーで提供され、アプリケーション・プログラムとリンク・エディットする必要があります。これらのスタブは、EXEC CICS コマンドの実行時に呼び出されます。

制御を AMODE(64) アプリケーション・プログラムから CICS に移すため、以下のスタブ・プログラムが提供されています。これらのスタブは、CICS 内部制御ブロックを使用して、必要な CICS コードを見つけます。

- DFHEAG0 (prolog および epilog スタブ)。

このスタブは、DFHEIENT マクロによって呼び出された場合は、制御を AMODE(64) PROLOG プログラムに移します。

DFHEIRET マクロによって呼び出された場合は、制御を AMODE(64) EPILOG プログラムに移します。

- DFHEAG (command スタブ)。

このスタブは、DFHECALL マクロによって呼び出された場合は、制御を AMODE(64) 初期コマンド・プロセッサに移します。DFHEG1 は DFHEAG の別名であり、DFHECALL マクロは、実際には別名 DFHEG1 を呼び出します。

スタブ DFHEAG およびスタブ DFHEAG0 は、AMODE(64) アプリケーション・プログラムとリンク・エディットする必要があります。以下のバインダー・ステートメントを使用してください。

```
ORDER DFHEAG
INCLUDE SYSLIB(DFHEAG)
ENTRY
 program_name
NAME
 program_name
(R)
```

*program\_name* は、AMODE(64) アプリケーションの名前です。

これは、スタブ DFHEAI およびスタブ DFHEAI0 が AMODE(24) および AMODE(31) アセンブラー言語アプリケーション・プログラムとリンク・エディットされるのと同じような方法です。

---

## 大文字変換

端末で入力される小文字、大/小文字混合、および国別文字の変換をカスタマイズすることができます。一時記憶域データ共用によって生成されるオペレーター・メッセージを変換することもできます。

### 国別文字の大文字への変換

CICS では、端末のユーザー入力を大文字に変換する場合、PROFILE 定義と TYPETERM 定義 の UCTRAN オプションを使用するか、EXEC CICS SET TERMINAL(termid) UCTRANST コマンドを使用することができます。

ただし、一部の言語には、UCTRAN で変換される EBCDIC 文字のセットに含まれていない文字があります。そのため、リソース定義に指定されている内容に関係なく、これらの文字は大文字に変換されません。これらの国別文字を変換する場合は、以下の 2 つのオプションがあります。

- XZCIN 出口を使用する
- 現行の TCT (または、SIT に TCT = NO が指定されている場合はダミーの TCT である DFHTCTDY) に基づいて、新しい端末管理テーブル (TCT) を作成し、その中の変換テーブルを変更します。

どちらの方法を使用する場合でも、Character Data Representation Architecture は、コード・ページに関する情報を参照する際に役立ちます。

## XZCIN 出口の使用

XZCIN については、出口 XZCIN に説明があります。これを大文字変換に使用するには、端末の入力が発生した場合に呼び出される独自の変換ルーチンを提供する必要があります。

## DFHTCTxx の使用

UCTRAN で処理されない国別文字を変換する場合は、端末管理テーブル内の変換テーブルを変更できます。

### このタスクについて

すべての端末用の RDO を使用するとき、SIT またはそのオーバーライドに TCT=NO が指定されている場合、CICS ではダミーの TCT である DFHTCTDY を使用して、RDO 定義の自動インストールされた端末の制御ブロックを作成します。DFHTCTDY は、直接変更しないようにしてください。代わりに、DFHTCTDY ソース・ファイルのコピーを作成して、それを新しい名前で保存し、コピーを変更します。以下のステップを実行する必要があります。

注: DFHTCTDY ではなくカスタマイズされた TCT を既に使用している (つまり、SIT TCT パラメーターに「NO」や「DY」以外が指定されている) 場合は、使用している TCT に変換コードを追加する必要があります。

### 手順

1. DFHTCTDY アセンブラー・ソース・ファイルのコピーを作成します。CICS では、CICSTS55.CICS.SDFHSAMP ライブラリーにこれが用意されています。
2. 918 ページの図 147 に示すように、ソース・ファイルの変換テーブルを変更します。
3. ソース・ファイルを DFHTCTxx として保管します。ここで、「xx」は、「DY」以外の 2 文字の接尾部です。
4. CICS 提供の DFHAUPLE ジョブを使用してアセンブルし、SMP/E に定義して、新規テーブルにリンク・エディットします。
5. SIT TCT パラメーターに、新規 TCT の 2 文字の接尾部を指定します。
6. CICS を再始動すると、新規 TCT が有効になります。

### 例

918 ページの図 147 に、国別文字を変換するアセンブラーのソース・ステートメントをコーディングするための推奨される方法を示します。

```

MACRO
NATLANG
DFHUCTRT CSECT Resume UCTRAN table CSECT
.*
.* This example translates lowercase 'a' (EBCDIC X'81') to
.* uppercase 'A' (EBCDIC X'C1') for a US code page.
.*
ORG TCZUCTAB+X'81' Reset the counter to the
character to be translated.
DC X'C1' Declare the replacement
character as a constant.
.*
.* Repeat the above two statements for each extra character you want
.* to be translated.
.*
ORG , Reset the location counter
&SYSLOC LOCTR Resume previous location counter
MEND End of macro definition
DFHTCT TYPE=INITIAL,SUFFIX=xx, *
MIGRATE=COMPLETE, *
ACCMETH=(VTAM), *
DUMMY=DUMMY
NATLANG Execute NATLANG
DFHTCT TYPE=FINAL
END DFHTCTBA

```

図 147. 各国語文字変換の推奨されるコーディング

## データ共用メッセージの大文字への変換

CICS 一時ストレージ (TS) データ共用では、AXM サービスを使用してオペレータ・メッセージが作成されます。これらのメッセージは、大/小文字混合の英語です。印刷不能文字を削除する場合は、テーブル AXMMSTAB を使用します。必要に応じて、AXMMSTAB を変更し、メッセージを大文字の英語に変換できます。

AXM メッセージ・サービスを使用するモジュールは、AXMSI、AXMSC、および DFHXQMN です。AXMSI と AXMSC は、ともにリンク・リスト・ライブラリーにあります。DFHXQMN は CICS 許可ライブラリーにあります。TS データ共用メッセージを大文字に変換するには、(モジュールごとに) 以下の入力に SPZAP を使用し、これらの各モジュールで使用される AXMMSTAB のコピーを変更します。

```

NAME modulename AXMMSTAB
VER 0081 818283848586878889
VER 0091 919293949596979899
VER 00A2 A2A3A4A5A6A7A8A9
REP 0081 C1C2C3C4C5C6C7C8C9
REP 0091 D1D2D3D4D5D6D7D8D9
REP 00A2 E2E3E4E5E6E7E8E9

```

---

## 第 14 章 アプリケーションのセットアップ

アプリケーションは、クラウドを使用可能な CICS Transaction Server for z/OS の主要な機能 1 つです。CICS<sup>®</sup>内でビジネス・アプリケーションを構成するリソースの大規模なセットは、単一のエンティティーとして論理的に定義し、単一のリソースとしてプラットフォームにデプロイできます。

### 始める前に

アプリケーションの概要については、仕組み: アプリケーションを参照してください。アプリケーション・リソースの保護については、プラットフォームとアプリケーションのセキュリティを参照してください。

### このタスクについて

アプリケーションのセットアップの大部分の作業は、CICS Explorer で行います。CICS クラウド・パースペクティブでは、アプリケーションのライフサイクルを管理するためのビューを提供します。

以下のステップに従って、アプリケーションをセットアップします。各ステップには、より詳細な説明へのリンクが含まれています。

### 手順

1. アプリケーションの一部としてデプロイするアプリケーションのエントリー・ポイント、ポリシー、およびリソースを考慮して、アプリケーションを設計します。詳しくは、クラウド対応のアプリケーションの設計を参照してください。
2. CICS Explorer で CICS バンドルを作成し、アプリケーション・リソース、アプリケーション・エントリー・ポイント、依存関係、およびアプリケーションに関連するすべての CICS ポリシーを含めます。詳しくは、Defining CICS bundlesを参照してください。
3. CICS Explorer でアプリケーション・プロジェクトを作成し、アプリケーション・バンドルを定義します。アプリケーション・バンドルでは、必要な CICS バンドルを参照します。詳しくは、クラウド環境でのデプロイメントのための CICS アプリケーションのパッケージ化を参照してください。
4. CICS Explorer でアプリケーション・バインディング・プロジェクトを作成し、プラットフォーム内の領域タイプに対してアプリケーションのデプロイメント規則を記述します。
5. CICS Explorer から、アプリケーション・プロジェクト、アプリケーション・バインディング・プロジェクト、および CICS バンドルを zFS にエクスポートします。エクスポート・プロセスにより、zFS 内のプラットフォーム・ホーム・ディレクトリーのアプリケーション、バインディング、およびバンドルの各サブディレクトリーにファイルがエクスポートされます。詳細については、プラットフォームへのアプリケーションのデプロイを参照してください。
6. CICS Explorer で、アプリケーション定義を作成します。アプリケーション定義は、アプリケーションが実行されるプラットフォームのプラットフォーム・ホーム・ディレクトリー内のアプリケーション・バンドルおよびアプリケーション

ン・バインディングを指す CICSplex SM APPLDEF リソース定義です。詳細については、プラットフォームへのアプリケーションのデプロイを参照してください。

7. CICS Explorer で、アプリケーションをプラットフォームにインストールします。CICSplex SM では、アプリケーション・バンドルとアプリケーション・バインディングの情報を使用して、アプリケーションを構成する CICS バンドルをプラットフォーム内のすべてのターゲット CICS 領域にインストールします。各領域には、バンドルで指定されたリソースが動的に作成され、CICS により、依存関係として指定されたすべてのリソースが存在することが検査されます。詳細については、プラットフォームへのアプリケーションのデプロイを参照してください。
8. CICS Explorer で、ユーザーが使用可能なアプリケーション・エン트리・ポイントから開始できるようにアプリケーションを使用できるようにします。詳細については、アプリケーションの管理を参照してください。

## タスクの結果

プラットフォームで稼働するアプリケーションを作成しました。

## 次のタスク

環境を制御するためのポリシーをデプロイすることにより、サービス品質をさらに加えることができます。詳しくは、CICS ポリシーを参照してください。

アプリケーションの管理方法について詳しくは、プラットフォームおよびアプリケーションの管理を参照してください。

---

## プラットフォームへのデプロイメントのための CICS アプリケーションの設計

CICS クラウド・ソリューションの一部としてプラットフォームにデプロイするためのアプリケーションを作成する前に、アプリケーションをどのように構造化する予定なのかを識別します。例えば、アプリケーションにバンドルするリソースまたはその依存関係として宣言するリソース、アプリケーションにアクセスするために提供するアプリケーション・エン트리・ポイント、アプリケーションをデプロイする場所、および複数バージョンのアプリケーションを同時に実行する予定かどうかを検討します。

### アプリケーション・エン트리・ポイントの識別

エン트리・ポイントを識別する際には、考慮する必要がある要因がいくつかあります。

- トランザクション・フローでは、アプリケーションに含まれるタスクを早期に識別すればするほど、ポリシーを迅速に適用して、アプリケーションのコストのモニタリングを開始できます。これは、アプリケーションが複数の CICS TS 領域にわたる場合、特に重要となる可能性があります。アプリケーションが関与するすべての CICS TS 領域にわたってアプリケーションをモニターするには、最初にインバウンド要求を受け入れる CICS TS 領域でエン트리・ポイントを定義する必要があります。

- アプリケーションには、実行するさまざまな機能を含めることができます。これらは、アプリケーションの操作（顧客の照会など）と呼ばれます。これらの操作を識別したり、個々の操作をモニターしたり、個々の操作にポリシーを適用したりすることは有効です。

## CICS バンドルで定義する必要があるリソース

CICS バンドルの中のリソースを定義することは、重要なステップです。それらのリソースの管理とライフサイクルは、それらのリソースをインストールした CICS バンドルおよび管理バンドルに委任されているからです。もはや個別にリソースを変更したりリソースの状態を変更することはありません。なぜなら、CICS バンドルおよびアプリケーションで操作しているときに、リソースは自動的に追加、更新、または削除されるためです。したがって、アプリケーション設計者は、アプリケーションのどのリソースのライフサイクルを、CICS バンドルのライフサイクルに結び付けるかについて、注意深く考慮する必要があります。

バンドル・リソースの特性の情報を確認して、CICS バンドルで定義するリソースをどう選択し、それらをどう調整したらよいかについて調べてください。CICS バンドル内で定義できないリソース、あるいは CICS バンドル内で定義することが望ましくないリソースがアプリケーションで使用されている場合は、それらを引き続きインポートとして指定することができます。

サポートされるリソース・タイプにおいては、CICS リソースが、アプリケーションの一部としてパッケージされてインストールされる CICS バンドルで定義される場合、専用リソースになります。これらのリソースは、プラットフォームにインストールされている他のアプリケーションやバージョン、あるいは CICS 領域内の他のアプリケーションでは使用できないため、リソース名はインストール環境内で固有でなくても構いません。アプリケーションの専用リソースにある情報を確認し、専用リソースの動作と管理の変更について確認してください。サポートされているリソース・タイプのリソースを専用にしなない場合は、引き続きそれをインポートとして指定してください。

## インストールするバンドルおよびそのインストール場所

バンドルをアプリケーションにデプロイしたり、プラットフォームに追加したりする代わりに、どのバンドルをアプリケーションに含め、どのバンドルをアプリケーション・バイndingに追加しますか。これらのバンドルには、ターゲット・プラットフォームに合わせてアプリケーションの動作を制御したりカスタマイズしたりするポリシーまたはリソース定義を含めることができます。

---

## クラウド環境でのデプロイメントのための CICS アプリケーションのパッケージ化

クラウドを使用可能にするために、アプリケーションを作成します。これらのアプリケーションを使用して、プラットフォームにデプロイするためのコード、依存関係、およびリソースをパッケージ化します。CICS Explorer で、アプリケーションを作成します。各アプリケーション・コンポーネントの CICS バンドル、およびそれらをグループ化するアプリケーション・バンドルを作成します。これらのバンドルでアプリケーション・エントリー・ポイントを宣言し、アプリケーション・バイndingを作成します。

## 始める前に

アプリケーションの概要については、仕組み: アプリケーションを参照してください。このタスクでは、以下の作業を完了していることが前提となります。

1. アプリケーションを構造化する方法の決定。完了していない場合は、クラウド対応のアプリケーションの設計を参照してください。
2. アプリケーションのデプロイ先となるプラットフォームのセットアップ。完了していない場合は、**Setting up a platform**を参照してください。ターゲット・プラットフォームのプラットフォーム・プロジェクトが **CICS Explorer** のローカル・ワークスペースに存在していることを確認してください。この段階ではアプリケーションをデプロイしませんが、**CICS Explorer** には、アプリケーション・プロジェクトおよびアプリケーション・バインディング・プロジェクトを検証するためにプラットフォーム・プロジェクトが必要になります。

## このタスクについて

**CICS Explorer** で、アプリケーションを作成します。以下のステップで手順の概要を示します。詳細なステップについては、**CICS Explorer** 製品資料内の『**Working with applications**』を参照してください。

## 手順

1. 各アプリケーション・コンポーネント用に **CICS** バンドル・プロジェクトを作成します。それぞれについて、**CICS** バンドル内に適切なリソースを定義し、コンポーネントで必要となるその他のリソースに対するすべての依存関係を宣言します。この **CICS** バンドルはさまざまなリソースと一緒にグループ化し、1つのエンティティとしてバージョン管理および管理されます。このため、別々に更新および管理する必要のある複数のリソースは同じ **CICS** バンドルに入れないでください。**CICS** バンドルでのリソースについて詳しくは、**CICS** バンドルの定義を参照してください。
2. **CICS** バンドル・プロジェクトで、アプリケーションへのアクセス・ポイントとなるリソースを識別するためのアプリケーション・エン트리・ポイントを追加します。アプリケーション・エン트리・ポイントについて詳しくは、アプリケーションのエン트리・ポイントを参照してください。
3. **CICS** アプリケーションの名前とバージョンを指定する **CICS** アプリケーション・プロジェクトを作成します。**CICS** アプリケーション・プロジェクトでは、アプリケーション・バンドルを定義します。**CICS** バンドルの参照を追加することで、それらをアプリケーション・バンドルに含めます。アプリケーション・バンドルを作成後に編集して、**CICS** バンドルを追加または削除できます。
4. **CICS** アプリケーション・バインディング・プロジェクトを作成して、アプリケーションの各 **CICS** バンドルをターゲット・プラットフォームの **CICS** 領域タイプにデプロイする方法を定義します。アプリケーション・バインディングを作成後に編集して、**CICS** バンドルのデプロイ方法を変更できます。アプリケーション・バインディングについては、アプリケーション・バインディングを参照してください。

## 次のタスク

アプリケーションをプラットフォームにデプロイする準備ができたなら、CICS アプリケーション・プロジェクトおよび CICS アプリケーション・バインディング・プロジェクトを zFS のプラットフォーム・ホーム・ディレクトリーにエクスポートします。これにより、アプリケーション・バンドル、アプリケーション・バインディング、および関連付けられた CICS バンドルがターゲット・プラットフォームのホーム・ディレクトリーで使用できるようになり、プラットフォームの CICS 領域にインストールできるようになります。また、プラットフォーム・ホーム・ディレクトリー内のアプリケーション・バンドルを指す CICSplex SM APPLDEF リソース定義となるアプリケーション定義を作成します。詳しくは、プラットフォームへのアプリケーションのデプロイを参照してください。

---

## 複数バージョンに対応するアプリケーションの呼び出し

同じアプリケーションの 2 つ以上のバージョンを、1 つのプラットフォームに同時にインストールし、有効にし、使用可能にすることができます。複数のバージョンを選択できる場合、呼び出し元では、選択可能な最新のアプリケーション・バージョンにアクセスすることも、**EXEC CICS INVOKE APPLICATION** コマンドを使用して特定のバージョンまたは最小バージョンを呼び出すこともできます。

### 始める前に

プログラムをコーディングする前に、以下を知っておく必要があります。

- アプリケーションの名前。
- アプリケーションがインストールされているプラットフォームの名前。さもなくば、アプリケーションが現行プラットフォーム上にインストールされていることを確認します。
- 呼び出されるアプリケーションのいずれかのプログラム入り口点に対応する操作の名前。
- 呼び出すアプリケーションの正確なメジャー・バージョン。
- 呼び出すアプリケーションの正確なマイナー・バージョンまたは最小マイナー・バージョン。

### このタスクについて

同じアプリケーションの 2 つ以上のバージョンを、1 つのプラットフォームに同時にインストールし、有効にし、使用可能にすることができますが、**EXEC CICS LINK** コマンドに対して表示されるバージョンはこれらのうち 1 つのみです。このバージョンは、アプリケーションの最大のメジャー・バージョンとマイナー・バージョンであり、エントリー・ポイントはパブリックです。よって、エントリー・ポイント・プログラムへの **EXEC CICS LINK** は、常にアプリケーションの最上位バージョンを呼び出します。アプリケーションの下位レベルのエントリー・ポイントは専用です。よって、**EXEC CICS LINK** では表示されません。

注: マイクロ・バージョンは、内部変更 (例えば、バグ修正) を反映するので常に非表示となります。呼び出し元は常に最新のマイクロ・バージョンを取得します。

**EXEC CICS INVOKE APPLICATION** を使用すると、エントリー・ポイント・プログラムの名前を知らなくても、あるいはそのエントリー・ポイントがパブリックかどうか

に関係なく、そのプログラム・エン트리・ポイントの 1 つを介してアプリケーションが呼び出されます。バージョンを指定しない場合、最大のメジャー・バージョンおよびマイナー・バージョン (公開レベル) が呼び出されます。これは、アプリケーション・エン트리・ポイントに対して **EXEC CICS LINK** コマンドを使用した場合と同じ振る舞いです。ただし、**EXEC CICS INVOKE APPLICATION** を使用し、適切なメジャー・バージョンとマイナー・バージョンを指定することにより、有効で使用可能な下位バージョンを呼び出すことができます。アプリケーションのメジャー・バージョン番号とマイナー・バージョン番号の完全一致が必要であること、あるいは、マイナー・バージョン番号が必要な最小バージョンであることを指定できますが、より大きいマイナー・バージョンが使用可能な場合はそれが使用されます。より大きいマイナー・バージョンが複数使用可能である場合、最大のバージョンが使用されます。メジャー・バージョン番号を超えることはできません。完全に一致している必要があります。

**EXEC CICS INVOKE APPLICATION** コマンドの完全な構文については、**INVOKE APPLICATION**を参照してください。

## 手順

1. プログラムで、**EXEC CICS INVOKE APPLICATION** コマンドを使用して、呼び出すアプリケーションの名前を指定します。
2. コマンドに、**OPERATION** オプションを追加し、必要に応じて **PLATFORM** オプションを追加します。
  - **OPERATION** オプションは、アプリケーション・エン트리・ポイント・プログラムが実装するアプリケーション操作の名前を指定します。
  - **PLATFORM** オプションは、アプリケーションがインストールされているプラットフォームの名前を指定します。プラットフォーム名を指定しない場合、現在のプラットフォーム名が使用されます。
3. オプション: コマンドに、**MAJORVERSION** オプションと **MINORVERSION** オプションを追加し、**EXACTMATCH** キーワードまたは **MINIMUM** キーワードのいずれかを追加します。
  - **MAJORVERSION** オプションは、アプリケーションのメジャー・バージョン番号をフルワード・バイナリー値として指定します。**MAJORVERSION** を指定した場合には、**MINORVERSION** も指定する必要があります。バージョンを指定しない場合、最大のメジャーおよびマイナー・バージョンのアプリケーションが呼び出されます。
  - **MINORVERSION** オプションは、アプリケーションのマイナー・バージョン番号をフルワード・バイナリー値として指定します。
  - **EXACTMATCH** キーワードは、アプリケーションのメジャー・バージョン番号とマイナー・バージョン番号の完全一致が必須であることを指定します。
  - **MINIMUM** キーワードは、指定したマイナー・バージョン番号が、必要な最小バージョンであることを指定します。ただし、使用可能である場合はより大きいバージョンを使用します。より大きいマイナー・バージョンが複数使用可能である場合、最大のバージョンが使用されます。これは、マイナー・バージョン番号のみに適用されます。メジャー・バージョン番号を超えることはできません。完全に一致している必要があります。

EXACTMATCH キーワードまたは MINIMUM キーワードのいずれかを使用する場合、マイクロ・バージョンの一致基準は存在しません。最大のマイクロ・バージョンが常に使用されます。

4. オプション: コマンドに、COMMAREA オプションと LENGTH オプションを追加します。
  - COMMAREA オプションは、呼び出されたプログラムが使用できる連絡域を指定します。このオプションで、データ域が渡されます。受け取る側のプログラムは、このデータ域に DFHCOMMAREA という名前を指定しなければなりません。
  - LENGTH オプションは、COMMAREA のバイト単位の長さをハーフワード・バイナリー値で指定します。COMMAREA を任意の 2 つの CICS サーバー間で受け渡す場合、この値は 24 KB を超えてはなりません。
5. オプション: コマンドに CHANNEL オプションを追加します。CHANNEL オプションは、呼び出されたエントリー・ポイント・プログラムで使用可能にするチャンネルの名前 (1 から 16 文字) を指定します。チャンネルが存在しない場合は、作成されます。

## タスクの結果

プログラムがこのコマンドを発行した場合、CICS は、適切なエントリー・ポイントで指定されたアプリケーションを呼び出します。

---

## EXEC CICS INVOKE APPLICATION の例

- 例 1 この例は、アプリケーションのメジャー・バージョン番号とマイナー・バージョン番号が完全一致するものを引き渡して、現行プラットフォーム上で実行するアプリケーションを呼び出す方法を示しています。

```
EXEC CICS INVOKE APPLICATION(PAYROLL)
 OPERATION(APPLY_TAX_CHANGES)
 MAJORVERSION(2)
 MINORVERSION(3)
 EXACTMATCH
```

ここで、

### **PAYROLL**

呼び出されているアプリケーションの名前です。

### **APPLY\_TAX\_CHANGES**

呼び出されるアプリケーション操作の名前です。

- 2 呼び出されるアプリケーション **PAYROLL** のメジャー・バージョンです。
- 3 呼び出されるアプリケーション **PAYROLL** のマイナー・バージョンです。これは、正確に一致する必要があります。

このコードを実行すると、アプリケーション **PAYROLL** のバージョン 2.3.x の **APPLY\_TAX\_CHANGES** 操作が呼び出されます。x は最上位のマイクロレベルです。

- 例 2 この例は、アプリケーションの最小マイナー・バージョンを指定して、指定されたプラットフォーム上で実行するアプリケーションを呼び出す方法を示しています。

```
EXEC CICS INVOKE APPLICATION(PENSIONS)
 OPERATION(UPDATE_PENSIONS)
 PLATFORM(TEST_PENSIONS_PLATFORM)
 MAJORVERSION(4)
 MINORVERSION(2)
 MINIMUM
```

ここで、

**PENSIONS**

呼び出されているアプリケーションの名前です。

**UPDATE\_PENSIONS**

呼び出されるアプリケーション操作の名前です。

**TEST\_PENSIONS\_PLATFORM**

アプリケーションがインストールされているプラットフォームの名前です。

- 4 呼び出されるアプリケーション **PENSIONS** のメジャー・バージョンです。
- 2 呼び出されるアプリケーション **PENSIONS** の最小マイナー・バージョンです。上位のマイナー・バージョンが存在する場合は、最上位のマイナー・バージョンが呼び出されます。

このコードを実行すると、**PENSIONS** アプリケーションのバージョン 4.2 以上の **UPDATE\_PENSIONS** 操作が呼び出されます。例えば、**PENSIONS** のバージョン 4.2.4、4.3.3、および 4.4.1 が有効で使用可能であった場合、このコマンドの結果、バージョン 4.4.1 が呼び出されることになります。

---

## 第 15 章 アプリケーションのデバッグ

CICS 提供ユーティリティまたは IBM から入手できるその他のツールを使用して、CICS アプリケーションをデバッグすることができます。CICS 提供デバッグ・ツールには、ワークステーション・ベースのデバッグ・ツールおよび実行診断機能 (EDF) が含まれています。

これらのツールを使用して、コードをステップスルーし、アプリケーション・プログラミング・コマンドとシステム・プログラミング・コマンドの構文をチェックします。デバッグ用のその他の IBM ツールとしては、IBM Developer for Z や IBM Debug Tool などがあります。

また、トランザクション・ダンプを収集し、デバッグを目的としてトレースを実行することもできます。詳細については、『CICS トランザクション・ダンプ』および『トレース』を参照してください。

---

### 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。
- ・ タスク正常終了時点。
- ・ **ABEND** が発生し、異常なタスク終了の後。

EDF の実行例については、「*Designing and Programming CICS Applications (ISBN 1565926765)*」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ます。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・タスクを直接ペ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** タスクを強制的にページし、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 度目の強制ページを行います。**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』
- 931 ページの『EDF 表示画面での実行内容』
- 939 ページの『EDF を使用したプログラムのテスト』
- 946 ページの『EDF による情報の変更』
- 948 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。

### オープン TCB および EDF

ユーザー・プログラムが、通常 OPEN TCB (L8、L9、X8、または X9) を使用して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。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ります。

### セキュリティに関する考慮事項

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・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属性を定義します。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。

## アプリケーションの前提条件

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。

### 単一画面モードにおける制約事項

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 トランザクションのプロファイル定義に PROTECT=YES を指定しないでください。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をテストするには、二重画面モードで実行します。
- ・ 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 の下で 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はな

く、EDF 端末に対して発行されます。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。

- 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てください。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ることがあります。タスクが異常終了した場合には、EDF はモニター端末で再活動化されます。

## 両方の画面モードにおける制約事項

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せんが、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 はパスワード値またはパスワード・フレーズ値の表示を抑制し、それらが偶然見られてしまうリスクを減らします。
- EDF を接続に対して使用する場合、接続を介して送られるファイル制御コマンドの機能は、CEDF の実装環境の理由で表示されません。

## 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表示され、その代行受信点に関連する情報も表示されます。

932 ページの図 148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
端末
ERASE
NOFLUSH
NOHANDLE

OFFSET:X'002522' LINE:00673 EIBFN=X'1804'
RESPONSE: NORMAL EIBRESP=0
3
ENTER: CONTINUE
4
F1 : UNDEFINED F2 : SWITCH HEX/CHAR F3 : END EDF SESSION
F4 : SUPPRESS DISPLAYS F5 : WORKING STORAGE F6 : USER DISPLAY
F7 : SCROLL BACK F8 : SCROLL FORWARD F9 : STOP CONDITIONS
F10: PREVIOUS DISPLAY F11: EIB DISPLAY F12: ABEND USER TASK

```

図 148. 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、本文 (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。 本文が 1 画面に収まらない場合には、ファンクション・キー F7 と F8 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

画面のヘッダーには、実行中のトランザクションの ID、実行中のプログラムの名前、CICS によってトランザクションに割り当てられる内部タスク番号、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、および EDF による代行受信の理由である STATUS の情報が含まれます。

## 本文

本文、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。以下の表示画面は、本文の内容を、プログラムの開始時、CICS コマンドの実行の開始時と終了時、プログラムとタスクの終了時、および異常終了時に表示します。

プログラムの開始時:

この例は、プログラムの開始時の表示画面を示しています。 EDF は、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。

これらの EIB フィールドのプログラミング情報については、アテンション ID 定数、DFHAIDを参照してください。 COMMAREA が指定されていない場合、画面

上の行 4 はブランクのままになり、EIBCALEN の値はゼロになります。

```
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
EIBAIID = 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 149. 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:

この例は、CICS コマンドの実行開始時の表示画面を示しています。EDF は、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。

CICS コマンドの実行開始時の代表的な EDF 表示画面を、934 ページの図 150に示します。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
端末
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 150. CICS コマンドの実行開始時の代表的な EDF 表示画面

図 151 には、Db2 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 151. 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、図 150 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、901 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:

この例は、コマンドの実行終了時の表示画面を示しています。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。図 152 に、934 ページの図 150 の実行開始の画面に対応する完了画面を示します。

```
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
端末
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 152. 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 EIB フィールドに記載され、DL/I のコードについてはIMS 製品資料内の『DL/I 状況コード』に記載されています。

936 ページの図 153 および 936 ページの図 154 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 (' ')
SEGLLENGTH (+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 153. 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 ')
SEGLLENGTH (+00010)
LOCKED
INTO ('SMITH ')
WHERE (ACCOUNT = '12345')
FIELDLLENGTH (+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 154. 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 155. 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:

この例は、プログラムの終了時および通常のタスク終了時の表示画面を示しています。 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 156 と 938 ページの図 157 に、プログラムとタスクの終了の要約画面を示します。

```

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 156. 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 157. タスク終了時の代表的な EDF 表示画面

異常終了時:

この例は、異常終了または異常なタスク終了が発生したときの表示画面を示しています。

異常終了または異常なタスク終了が発生した場合、EDF には、図 158 と 939 ページの図 159 に示す画面が表示されます。

```
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 158. 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 159. 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 を使用したプログラムのテスト

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使用する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、CEDF トランザクションを使用します。

EDF を実行するには、通常いくつかの方法から 1 つ選択できますが、特定の方法が必要な状態がいくつかあります。例えば、リモート・トランザクションには単一画面モードを使用する必要があります。選択に影響するその他の条件については、929 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

EDF を使用すると、プログラムの代行受信点でさまざまな操作を実行することができます、デバッグに役立ちます。

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、コマンド自体を変更したり、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、任意のオプションに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、946 ページの『EDF による情報の変更』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、EXEC インターフェース・ブロック (EIB)、および DL/I インターフェース・ブロック (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。
- 一時記憶域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。
- 異常終了でタスクを停止する。

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。これを行う方法については、946 ページの『EDF による情報の変更』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。948 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 160. 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 は、次の 2 つの方法で開始できます。

- 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
- 適切なファンクション・キーを押す (EDF 用にファンクション・キーが定義されている場合)

次に、以下のステップを実行して、テストするトランザクションを開始します。

1. CLEAR キーを押して、画面をクリアしてください。
2. テストするトランザクションのトランザクション・コードを入力します。

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が必要とする場合
- 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・プログラムで使われている場合、また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 はそのアプリケーション・プログラムによって表示画面がいつ変更されたかを確認できません。このため、EDF は後で使用するためにその表示画面のコピーを保管することができません。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、復元できません。

同様に、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が必要とする場合、CEDF は現在の表示画面を復元できません。このため、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、直前の EDF 表示画面からの背景情報がランダムに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直前の EDF 画面に対して実行できます。または、それがトランザクションでの最初の入力コマンド受信である場合、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が必要とすることがあります。

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してください。

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します。

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

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デフォルトの Yes を受け入れるには、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 がこのトランザクション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可能性がある場合には、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

二重画面モードで 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、また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 の下で 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 を表示します。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らず、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します。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、次の条件を満たす必要があります。

- 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 コマンドで指定するトランザクションは、ローカルの CICS 領域で実行する必要があります。 CRTE ルーティング・トランザクションの後に CEDX トランザクションを使用することはできません。

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・クラスに戻します。

CEDX の読み取り専用の形式を使用する場合、つまり CEDY、同じ条件が適用されます。この場合、トランザクション・クラス DFHEDFTO が使用されます。

## EDF および DTP プログラム

リモート・リンク上のセッションをモニターするように実行診断機能 (EDF) を指定して、リモート・リンクを介して分散トランザクション処理 (DTP) を使用しているトランザクションをテストすることができます。

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 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 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 **INQUIRE TERMINAL** コマンドを使用して判別できますが、このことは、EDF を開始する前に、トランザクションが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に関連付けられている端末がある場合や、ある端末から（使用しなくても）トランザクションを呼び出せる場合は、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、そのシステム ID またはセッション ID で EDF をオフにします。以下に例を示します。

`CEDF sysid  
,OFF`

EDF をオフにできない場合、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発行された CICS コマンドは表示されませんが、リモート・プログラムが異常終了し、「a remote abend has occurred」というメッセージが、異常終了が発生したシステムのシステム ID と共に 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、プログラム状況ワード (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まだ実行中で、それを続行するのに、EDF を使用しない場合には、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。

## EDF による情報の変更

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。これらの変更を行うには、画面に表示される情報を、必要な情報で上書きします。

画面の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を移動でき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 コマンド画面では、任意の引数値を上書きできますが、引数のキーワードは上書きできません。オプションの引数を削除することはできず、オプションを追加または削除することはできません。
- (作業用ストレージ表示画面ではなく) コマンド表示画面で引数を変更する場合は、画面上に表示されている部分しか変更することができません。表示されている値の長さを超えて上書きしようとする、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します。アドレスを判別するには、引数位置のアドレスも表示されるように、引数を 16 進形式で表示します。
- フルワードの引数値を上書きする場合、入力できる最大値は 2147483639 です。
- 文字形式では、数値に必ず符号フィールドを付けます。符号付きフィールドは、マイナス文字 (-) またはブランクでしか上書きできません。
- 引数を文字形式で表示する場合、文字によっては表示可能でないことがあります (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。ピリオドを上書きするときには、ストレージに表示不能文字が含まれている場合があることに留意してください。

文字をピリオドで上書きして変更することはできません。そのような変更は無視され、診断メッセージは発行されません。文字をピリオドに変更するには、表示を 16 進形式に切り替え、F2 キーを使用して、値 X'4B' で上書きします。

- ストレージを文字形式と 16 進形式の両方で表示しているときに、両方の形式を変更し、それらの変更が競合する場合は、16 進数フィールドの値が優先されます。診断メッセージは発行されません。
- 一部のコマンド (例えば、HANDLE CONDITION など) 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・ラベルです。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、ヌル引数が表示され、それを変更することはできません (例: ERROR ()).
  - 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'))
  - AMODE(64) アセンブラー言語の場合、ラベルはサポートされていません。

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、現在の機能について発生する任意の例外条件の名前 (ERROR を含む) で、または単語 NORMAL で上書きすることができます。EDF が続行される場合の効果は、プログラムが、指定の応答で指示されている処置を行うことです。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 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ついて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してください。
- コマンドを処理する前にそのコマンドを NOOP または NOP で上書きすると、コマンドの処理を抑制することができます。ブランクで上書きするか、または ERASE EOF キーを使用した場合も、同様の効果があります。NOOP によって画面が再表示されたら、ERASE EOF キーを使用して verb 行全体を消去してから ENTER キーを押すと、元の verb 行を復元することができます。
- 引数が既に 64 ビット・アドレスである場合、アドレスの真ん中に下線を付けて 64 ビット・アドレスを入力することができます (例えば AAAAAAAAA\_BBBBBBBB または AAAAAAAAA\_BBBBBBBB)。

プログラム内のデータ域を表すフィールドを上書きした場合、変更は、アプリケーション・プログラム・ストレージで直接行われ、永続します。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、プログラム・ストレージは変更されません。なぜならば、そのような変更は、同じ定数を使用するプログラムの他の部分またはプログラムを使用する他のタスクに影響する可能性があるからです。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、次のコードを含むプログラムをテストするとします。

```
EXEC CICS SEND MAP('MENU') END-EXEC.
```

EDF を使用して名前を MENU から MENU2 に変更してからコマンドを実行した場合、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 の応答を確認するために、この規則のリストを使用できます。

規則は、以下に示す順序で適用されます。

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. ファンクション・キーを使用した場合には、EDF は変更を受け入れ、ファンクション・キーで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ファンクション・キー定義の下にカーソルを位置付けて ENTER キーを押すと、ファンクション・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 は状況表示画面を再表示します。

## EDF メニュー機能の使用

それぞれの時点で使用可能なファンクション・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 から PF24 を PF1 から 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。この処置を確定するため、「ENTER ABEND CODE AND REQUEST ABEND AGAIN」というメッセージが表示されます。カーソル位置にコードを入力してからこの機能を再び要求すると、指定された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が異常終了します。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 で使用するために予約済みです。CICS 異常終了コードを使用すると、予期しない結果になる場合があります。

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)

一時記憶域キュー 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。CEBR コマンドを使用し、一時記憶域キューの表示または変更、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)

変更を取り込むために、現在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)**

変更を取り込むために、現在の画面を再表示します。 変更を行わなかった場合は、EDF が最後の代行受信点の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 フィールドに関する詳細は、IMS 製品資料内の『IMS メッセージおよびコード』を参照してください。

#### **EIB DISPLAY (EIB 表示画面)**

EXEC インターフェース・ブロック (EIB) の内容を表示します。EIB 表示画面の例については、933 ページの図 149 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、EIB フィールド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)**

コマンド・レベルのインタープリター (CECI) にアクセスします。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。CECI が呼び出される画面の例については、941 ページの図 160を参照してください。そこで、964 ページの『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、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)**

直前の表示画面に戻った場合に、1 つ後の画面を表示して、表示番号を 1 だけ増やします。これは、PREVIOUS DISPLAY と逆のオプションです。

#### **PREVIOUS DISPLAY (前画面表示)**

他の表示画面が保管されていない限り、直前の表示画面を画面に送信します。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。それ以上古い画面がなくなると、PREVIOUS オプションがメニューで選択不可になり、対応するファンクション・キーが操作不能になります。

#### **REGISTERS AT ABEND (異常終了時のレジスター)**

ローカル ASRA 異常終了が発生した場合に、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、64 ビット・レジスターの後

に異常終了時の 16 バイト・プログラム状況ワード (PSW) が続きます。図 161 に、典型的な画面を示します。

場合によっては、EDF がレジスタ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。そのような状況が発生した場合には、もう一度テストを実行すると、さらに情報を得られる可能性があります。

```
TRANSACTION: UT PROGRAM: ASRA31 TASK: 0000487 APPLID: IYK2ZKE1
DISPLAY: 00
ADDRESS: 0006F010
0006F010 000000 00000000 00000090 00000000 0010003C
1
0006F020 000010 00000000 00041800 00000000 AE410028.....
1
0006F030 000020 00000000 2C2E35A8 00000000 00000000.....y.....
1
0006F040 000030 00000000 2C2D7800 00000000 7F2C9918.....".r.
1
0006F050 000040 00000000 2E410000 00000000 2C2D7818.....
1
0006F060 000050 00000000 00100008 00000000 00100100.....
1
0006F070 000060 00000000 009AF000 00000000 00100690.....0.....
1
0006F080 000070 00000000 AE410158 00000000 00100690.....
1
0006F090 000080 079D0000 80000000 00000000 2E410178.....
2
0006F0A0 000090 00040004 00000000 00000000 00000000.....
0006F0B0 0000A0 00000000 00000000 00000000 00000000
0006F0C0 0000B0 00000000 00000000 00000000 00000000.....
0006F0D0 0000C0 00000000 00000000 00000000 00000000
0006F0E0 0000D0 00000000 00000000 D4F0F0F0 F0F4F8F9.....M0000489
0006F0F0 0000E0 D4F0F0F0 F0F4F8F9 00000000 00000000 M0000489.....
0006F100 0000F0 00000000 000000E4 E3404000 00F00000UT..0..

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 161. REGISTERS AT ABEND の標準的な EDF 画面

注:

1. レジスター値
2. PSW

#### 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)

直前の画面を表示画面に表示します。この機能は、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。表示されている画面が表示画面の最初の画面ではなく、最初のオプションまたはフィールドの前に正符号 (+) が付いている場合には、この機能を選択して、前の画面をディスプレイに表示することができます。例については、933 ページの図 149 を参照してください。

#### **SCROLL FORWARD (順スクロール)**

次の画面を表示画面に表示します。この機能は、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。表示画面が収まらない場合、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。この機能を選択すると、次の画面が表示されます。

#### **SCROLL BACK FULL (一画面逆スクロール)**

直前の画面を作業用ストレージ表示画面に表示します。この機能は、作業用ストレージの表示画面に適用され、EIB 表示画面および DIB 表示画面の SCROLL BACK オプションと同じように機能します。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)**

次の画面を作業用ストレージ表示画面に表示します。この機能は、作業用ストレージの表示画面に適用され、EIB 表示画面および DIB 表示画面の SCROLL FORWARD オプションと同じように機能します。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)**

作業用ストレージの表示画面の半分を逆スクロールします。この機能は、作業用ストレージの表示画面を半画面分だけ逆スクロールするという点以外は、SCROLL BACK FULL と同じです。

#### **SCROLL FORWARD HALF (半画面順スクロール)**

作業用ストレージの表示画面の半分を順スクロールします。この機能は、作業用ストレージの表示画面を半画面分だけ順スクロールするという点以外は、SCROLL FORWARD FULL と同じです。

#### **STOP CONDITIONS (停止条件)**

SUPPRESS DISPLAYS 機能を使用した後に EDF が表示を再開するための条件を指定します。952 ページの図 162 のようなメニュー画面が表示されます。STOP CONDITIONS 機能と SUPPRESS DISPLAYS 機能を一緒に使用して、部分的に動作していることがわかっているプログラムを検査するときに対話を減らすことができます。

```

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 162. STOP CONDITIONS の代表的な EDF 表示画面

デフォルトでは、以下のいずれかの条件が発生した場合、EDF は、図 162 のように、表示を再開します。

- 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

STOP CONDITIONS メニューを使用して、適用しないデフォルトをオフにし、ユーザー・プログラム固有の条件を追加することができます。

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 エラー状況の発生、または特定の DL/I エラー状況の発生。
- 特定の例外条件の発生。CICS EXCEPTION CONDITION に ERROR (デフォルト) が指定されている場合、EDF はエラー条件 (例えば、NOTOPEN、EOF、または INVREQ) に応じて画面を再表示します。CICS EXCEPTION CONDITION に EOF などの特定の条件を指定する場合に、ANY CICS CONDITION に NO (デフォルト) が指定されていると、EDF は、その条件 (EOF) が発生したときにのみ画面を再表示します。

ANY CICS CONDITION に YES を指定すると、コマンドの結果がゼロ以外の EIBRESP 値 (NOTOPEN、EOF、QBUSY など) になるときは常に、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 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、EDF が表示を再開するように指定することができます。コマンド行で CICS 修飾子を DLI に書きし、表示抑制を停止する 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いし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 状況コードが発生した場合に表示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、「IMS 製品資料内の『IMS メッセージおよびコード』」で DL/I インターフェース・ブロック (DIB) 内の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影響ありません。

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)**

指定されているファンクション・キーが、現行の代行受信点における現行表示画面に対して定義されていません。

## USER DISPLAY (ユーザー表示)

トランザクションが EDF モードで実行されていなかった場合の画面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限りです)。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 24 ビットまたは 31 ビット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表示します。図 163 に、典型的な作業用ストレージ画面を示します。

この機能は 64 ビット・ストレージをサポートしていません。

```
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 163. 作業用ストレージの標準的な EDF 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 進形式と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ますか、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

現行プロシージャの動的ストレージ域 (DSA)。

#### アセンブラ言語

現行 DFHEISTG DSECT に定義されているストレージ。

アセンブラ言語プログラムでは、作業用ストレージが常に獲得されるわけではありません。例えば、プログラムで CICS コマンドを発行しない場合には必要ないこともあります。そのようなプログラムにリンクする場合に、「Register 13 does not address DFHEISTG」というメッセージが発行されることがあります。この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 プログラム以外では、作業用ストレージは 72 バイト標準形式の保管域で始まります。すなわち、レジスター 14 ~ 12 はオフセット 8 で始まり、レジスター 13 はオフセット 4 に格納されます。AMODE(64) プログラムの場合、作業用ストレージはフォーマット 4 保管域 (F4SA) で始まります。レジスター 14 ~ 12 はオフセット 8 で始まり、レジスター 13 はオフセット 128 に格納されます。

作業用ストレージは画面で変更することができます。16 進数セクションと文字セクションのいずれかを使用することができます。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上書きすることができます。その後 ENTER を押すと、そのアドレスで始まるストレージが表示されます。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。詳しくは、946 ページの『EDF による情報の変更』を参照してください。

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や、別のタスクに影響する可能性のあるストレージに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。この状態は、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 Language Environment プログラムを終了する場合、EDF がそのリターンをインターセプトしないので、作業用ストレージを表示することはできません。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。

Language Environment 対応のプログラムを使用している場合は、非 CICS リターンを使用してプログラムを終了すると、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。

---

## 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、一時記憶域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』
- 958 ページの『CEBR トランザクションの表示内容』
- 959 ページの『CEBR ファンクション・キーの使用』
- 960 ページの『CEBR コマンドの使用』
- 963 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、最大 16 文字を使用して入力することができます。例えば、AXBYQUEUENAME111 という名前の一時記憶域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 957 ページの図 164 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 (932 ページの図 148 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域キューを表示して、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 CEBR トランザクションを呼び出した場合の結果が 958 ページの図 165 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 を使用して、キュー名に 1 つ以上のブランクが埋め込まれた一時記憶域キューからデータを表示する場合、図 164に説明されているように、CEBR 画面から TS QUEUE フィールドにキュー名を入力する必要があります。CEBR に入った直後にこのようなキュー名を入力すると、予測不能な結果が生じます。

```
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 164. 一時記憶域キューの内容の典型的な CEBR 表示画面

```

CEBR TSQ AXBYQUEUEAME1AA SYSID CIJP REC 1 OF 0 COL 1 OF
1
ENTER COMMAND ==>
2
***** TOP OF QUEUE

***** BOTTOM OF QUEUE *****

3

TS QUEUE AXBYQUEUEAME1AA DOES NOT EXIST
4
PF1 : HELP PF2 : SWITCH HEX/CHAR PF3 : TERMINATE BROWSE
5
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

```

- 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 165. デフォルトの一時記憶域キューの典型的な CEBR 表示画面

## CEBR トランザクションの表示内容

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。
- ・ 一時記憶域キューの ID ( 957 ページの図 164 の AXBYQUEUEAME111 および 図 165 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、960 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、メッセージをユーザーに表示します。

例えば、958 ページの図 165 に示す「Does not exist」というメッセージです。

## 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、各 CEBR トランザクション画面の最下部に表示されており、すべての画面で同じ意味を持ちます。

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、元に戻したりします。

**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

CEBR は、一時記憶域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いくつかのコマンドを提供します。

**BOTTOM**

(省略形: B)

一時記憶域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。

以下に例を示します。

```
FIND /05-02-93
```

ストリング「05-02-93」の次のオカレンスを見つけます。 / 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、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。以下に例を示します。

```
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域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、963 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域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域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、963 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、一時記憶域キューの命名規則 (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、ブラウザ中の一時記憶域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、一時記憶域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・キューをブラウザした後で、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域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、一時記憶域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、常に一時記憶域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。

---

## 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 では、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、一時記憶域キューの作成または削除を行ったり、あるいは間違っただ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・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は、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』
- 970 ページの『CECI の使用』
- 973 ページの『コマンドの保管』
- 974 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。CECI 画面は、コマンド行、状況表示行、画面本文、メッセージ行、およびファンクション・キーの CECI オプションから構成されています。

- 『コマンド行』
- 965 ページの『状況表示行』
- 968 ページの『表示画面本体』
- 969 ページの『メッセージ行』
- 969 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。

- 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に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。

コマンド構文検査:

状況表示行が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たものの、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 (画面が分けられた後では、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:

この例は、コマンド実行開始の典型的な CECI 表示画面を示します。

この (図 166 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 166. コマンド実行開始の典型的な CECI 表示画面

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、まだ修正することができます。修正した場合には、CECI は前のコマンドを無視し、新しいコマンドを最初から処理します。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:

この例は、コマンド実行完了の典型的な CECI 表示画面を示します。

この画面 ( 968 ページの図 167 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 キーに応じて、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(+00000000361) >
+ < 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 167. 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、指定されていなくても、CICS 提供の値とともに強調表示されます。

## 表示画面本体

CECI 表示画面の本体には、3 つの画面すべてに共通の情報が含まれます。

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 ( 969 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、それを変更する場合は、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。 このアクションにより、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 967 ページの『CICS コマンドの完了』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。コマンドを実行した後で、メッセージ行は応答コードを表示します。

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りますを示しています。

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。PF9 を使用するとメッセージを表示できます。

968 ページの図 167 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 (968 ページの図 167 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、表示画面のファンクション・キーの効果を示すメニューが提供されています。

ファンクション・キーについて、以下に説明します。端末にファンクション・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。

### **F1 HELP**

コマンド・インタープリターの使い方およびファンクション・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。

### **F2 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。

### **F3 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。

### **F4 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。

### **F5 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。

#### **F6 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。

#### **F7 SBH**

(SCROLL BACK HALF) 本体を 1/2 画面分逆方向にスクロールします。

#### **F8 SFH**

(SCROLL FORWARD HALF) 本体を 1/2 画面分順方向にスクロールします。

#### **F9 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。

#### **F10 SB**

(SCROLL BACK) 本体を画面 1 つ分逆方向にスクロールします。

#### **F11 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。以下に例を示します。

```
CECS READ FILE('FILEA')
```

または

```
CECI READ FILE('FILEA')
```

CICS は、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 コマンド・サマリーやシステム・コマンドで説明されているコマンドは、どれでも使用できます。

トランザクション・コード CECS を使用した場合、インタ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。

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の場合があります。

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は `command syntax check` (コマンド構文検査) から処理が再開され、ない場合には `about to execute command` (コマンド実行開始) から処理が再開されます。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。

## 変数の定義

オプションの値のためコマンドがコマンド入力域の行の長さを超える場合や、オプション値を使用して 2 つのコマンドを接続する場合には、変数を定義することができます。

変数のリストを表示するには、PF5 を押します。現行のインタープリター・セッションに関連する変数ごとに、名前、長さ、および値が表示されます。CECI によって最初の 3 つの変数が作成され、それらの変数は、明示的に削除されない限り、常に表示されます。それらは、例を示すためのものであり、コマンド・リストを作成するために役立ちます。

これらの最初の 3 つの変数の後に、ユーザーが作成した変数が表示されます。例えば、次のようなコマンドを入力するとします。

```
READ FILE('FILEA') RID('009000') INTO(&REC)
```

&REC は変数として表示されます。

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、オプション値を使用して 2 つのコマンドを接続する場合などに

は、この値を表すのに変数を指定することができます。例えば、CECI を使用してレコードを変更するには、初めに次のコマンドを入力します。

```
EXEC CICS READ UPDATE INTO(&REC)
FILE('FILE') RID('009000')
```

次に、変数 &REC を変更することにより、レコードを変更します。それから、次のように入力します。

```
EXEC CICS REWRITE FROM(&REC) FILE('FILE')
```

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。

必要な値を使用して変数を作成し、コマンドでその変数名を指定すると、行の長さに関する制限を解消することができます。変数で利用できるデータ型は、文字、ダブルワード、フルワード、ハーフワード、またはパック 10 進数です。変数は、以下のいずれかの方法で作成することができます。

- 受信側で変数に名前を付けます。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。
- 既に定義されている変数のリストに新しい項目を追加します。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。ダブルワードの場合は、**FD** と入力します。フルワードの場合は、**F** と入力します。ハーフワードの場合は、**H** と入力します。パック変数の場合は、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。変数が作成されたら、「変数」画面でデータ・フィールドに変更を加えることにより、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作成してある場合は、状況表示行の NAME フィールドを使用します。そのためには、構文表示画面でオプションの下にカーソルを置いてから ENTER キーを押します。表示されているオプション値に関連付ける変数名を割り当てるため、その名前を NAME フィールドに入力し、再び ENTER キーを押します。
- 既存の変数をコピーします。そのためには、コピーする変数の拡張域表示画面を取得して、表示される名前を新しい変数の名前で上書きしてから 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します。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、973 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、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ます。セッションの終了前に変数を削除するには、カーソルを名前の先頭にあるアンパーサンドの下に置き、ERASE EOF キーを押してから ENTER キーを押します。

## 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があります。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域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` を入力してから 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。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「`CI`」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) 一時記憶域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 から 4 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

インタープリターは、CICS 提供のプログラムを使用して会話型トランザクションとして実行されます。セッションの開始と終了の間に行われることはすべて、単一のタスクの単一の作業論理単位です。

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、その実行状態が続きます。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。保護リソースに関連する一連の更新を行っている場合には、すべての更新を完了できることを確認してください。更新をすべて完了できない場合には、**SYNCPOINT ROLLBACK** コマンド、または **CANCEL** オプションを指定して **ABEND** コマンドを使用し、リカバリー可能リソースに対するこれより前のコマンドの影響を取り除きます。

### 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。この条件は、一部のコマンドで、明示的に指定しないオプションを CECI が使用するために返されます。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返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、代行受信されます。

**LINK** コマンドを使用した場合には、ターゲット・プログラムがインタープリターの環境で実行されます。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データを必要とする場合
- 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

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。

## 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、意図した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。

---

## 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

#### 手順

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 システム上で実行されているデバッガー・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、982 ページの『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、984 ページの『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い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 (Language Environment) 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、JVM プロファイルの検証およびプロパティー 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。

以下に例を示します。

- 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
- 名前が「setBankAccount」で始まるすべての 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 z/OS Communications Server 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、コンパイルの単位の名前 (プログラム名またはクラス名)
- Java オブジェクトでは、クラス名
- 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。

A3270 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

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 領域によって共用される 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 85 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。表 86 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 85. 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 86. 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

以下に例を示します。

\* 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わ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、982 ページの図 168 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 168. デバッグ・プロファイルのマッチング・アルゴリズム

---

## 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

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。

### 手順

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ります。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 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。

---

## 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 사용할 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。

## 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

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。

### 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。

### 手順

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 16 章 アプリケーションのデプロイ

アプリケーションを、CICS Explorer およびアプリケーション開発製品 (IBM Developer for Z など) から CICS にデプロイできます。デプロイメントには、コードが適切な場所 (例えば、zFS 内のデータ・セットやディレクトリーなど) にあることを確認する作業、およびターゲット CICS 領域でリソースを作成してアプリケーションを使用可能にする作業が含まれます。

ビジネス・イベントや、Web サービス、ユーザー出口プログラムなどのアプリケーションをサポートするサービスをデプロイすることもできます。何らかのアプリケーションを CICS にデプロイするには、z/OS システム内のディレクトリーおよびデータ・セットを更新するため、およびリソースを CICS にインストールするための適切なアクセス・レベルが必要です。

---

### プラットフォームへのアプリケーションのデプロイ

アプリケーションをプラットフォームにデプロイするには、CICS アプリケーション・プロジェクトおよび CICS アプリケーション・バインディング・プロジェクトを CICS Explorer から zFS のプラットフォーム・ホーム・ディレクトリーにエクスポートします。次に、CICSplex SM でアプリケーション定義 (APPLDEF) を作成し、インストールします。最後に、アプリケーションを有効にし、使用できるようにします。

#### 始める前に

アプリケーションの概要については、仕組み: アプリケーションを参照してください。このタスクでは、以下の作業を完了していることが前提となります。

- アプリケーションを構造化する方法の決定。完了していない場合は、クラウド対応のアプリケーションの設計を参照してください。
- アプリケーションのデプロイ先となるプラットフォームのセットアップ。完了していない場合は、Setting up a platformを参照してください。
- CICS Explorer での CICS アプリケーション・プロジェクトおよび CICS アプリケーション・バインディング・プロジェクトの作成。完了していない場合は、クラウド環境でのデプロイメントのための CICS アプリケーションのパッケージ化を参照してください。

#### このタスクについて

アプリケーション成果物を CICS Explorer から zFS のプラットフォーム・ホーム・ディレクトリーにエクスポートし、アプリケーション定義を作成します。アプリケーション定義 (CICSplex のデータ・リポジトリーに存在する APPLDEF リソース定義) は、アプリケーション・バンドルの場所と、アプリケーションが実行されるターゲット・プラットフォームを識別します。アプリケーション定義は、エクスポート処理の直後に作成することも、しばらく経ってから作成することもできます。

以下のステップで手順の概要を示します。詳細なステップについては、CICS Explorer 製品資料内の『Working with applications』を参照してください。

CICS Explorer を使用せずに、アプリケーション・デプロイメントを自動化する方法については、DFHDPLOY ユーティリティによる CICS アプリケーションのデプロイメントおよびアンデプロイメントの自動化を参照してください。

プラットフォームにその他のバージョンのアプリケーションがインストールされていない場合は、プラットフォームに特定のアプリケーションの初回インストールを実行するための以下の手順を使用してください。アプリケーションの既存のバージョンを新しいバージョンで置き換える場合、アプリケーションの管理の手順を実行してください。

## 手順

1. CICS Explorer の CICS クラウド・パースペクティブを使用し、CICS Explorer から、アプリケーションを実行するプラットフォームの zFS のプラットフォーム・ホーム・ディレクトリーに CICS アプリケーション・プロジェクトをエクスポートします。CICS アプリケーション・プロジェクトをエクスポートすると、CICS Explorer<sup>®</sup> によって、アプリケーション・バインディング・プロジェクトおよびアプリケーション・バンドルとアプリケーション・バインディングに関連付けられた CICS バンドルが zFS 上のプラットフォームのホーム・ディレクトリーにエクスポートされます。プラットフォーム・ホーム・ディレクトリーに既にデプロイされ、適切なバージョンで CICSplex にインストールされている CICS バンドルは、エクスポートに含まれません。プラットフォーム・ディレクトリーの構造について詳しくは、z/OS UNIX のプラットフォーム・ディレクトリーの構造を参照してください。
2. CICS Explorer を使用して、アプリケーション定義 (APPLDEF) を作成します。この定義は、zFS 上のプラットフォーム・ホーム・ディレクトリー内のアプリケーション・バンドルの場所を指し、アプリケーションのターゲット・プラットフォームを識別します。アプリケーション・エクスポート・ウィザード内のボックスをチェックすると、プラットフォーム・プロジェクトをエクスポートした後すぐにアプリケーション定義を作成する選択が可能です。別の時点でアプリケーション定義を作成するには、CICS Explorer の「新規アプリケーション定義 (New Application Definition)」ウィザードを使用します。CICSplex SM は、CICSplex にアプリケーションを表す APPLCTN リソースを作成します。また、アプリケーション・バンドルのリカバリー処理で使用する、データ・リポジトリのアプリケーションに関するレコードを作成します。CICSplex SM は、アプリケーション・バンドルとアプリケーション・バインディングの情報を使用して、プラットフォーム内の CICS 領域に CICS バンドルをインストールします。アプリケーションは、最初、使用不可の状態でインストールされます。

アクティブ・プラットフォームにアプリケーションをインストールすると、プラットフォームの一部として定義された、アプリケーション定義のインストール時に稼働しているすべての CICS 領域において、CICS バンドルが CICSplex SM によって直ちにインストールされます。また、アプリケーション定義のインストール後にプラットフォーム内の CICS 領域を開始または再始動した場合、それらの領域でも CICS バンドルが CICSplex SM によってインストールされます。これらの CICS バンドルのリソースは領域始動時にインストールされますが、制御が CICS に戻されるまで、完全には有効にならない場合があります。

ます。アプリケーション定義をインストールした時点よりも後で、CICS 領域をプラットフォームにさらに追加した場合、CICSplex SM はそれらの領域にも CICS バンドルをインストールします。

アプリケーションと、インストールされた各 CICS バンドルとの関係は、管理パートで保管されます。管理パートは、アプリケーション・インストール・プロセスで各 CICS バンドル用に自動的に作成される MGMTPART レコードです。これはバンドルのインストール場所である CICS 領域について記録し、CICS 領域でのバンドルの状況を追跡します。

3. 「クラウド・エクスプローラー」ビューまたは CICS Explorer のオンライン・アプリケーション・エディターを使用して、インストールしたアプリケーションを有効状態にし、その状況を確認します。アプリケーションは、最初は使用不可の状態です。アプリケーションを使用可能にすると、CICSplex SM は、すべての CICS 領域内でそのアプリケーション用にインストールされた CICS バンドルを使用可能にしますが、呼び出し元からそのアプリケーション・エントリー・ポイントを介してアプリケーションを使用することはまだできません。アプリケーションを使用可能にした時より後にプラットフォーム内の CICS 領域を始動または再始動した場合、CICSplex SM はその領域にバンドルを使用可能な状態でインストールします。
4. プラットフォームのユーザーがアプリケーションを使用可能にする準備ができたなら、「クラウド・エクスプローラー」ビューまたは CICS Explorer のアプリケーション記述子エディターを使用して、アプリケーションを使用可能にします。アプリケーションを使用可能にすると、CICS は、そのアプリケーション・エントリー・ポイントとして宣言されている CICS リソースを介して呼び出し元がアプリケーションにアクセスできるようにします。呼び出し元は、使用可能な最大のアプリケーション・バージョンにアクセスするか、または EXEC CICS INVOKE APPLICATION コマンドを使用して使用可能な特定のアプリケーション・バージョンにアクセスすることができます。アプリケーションの可用性状況は、アプリケーションを使用可能にした後にプラットフォームの CICS 領域を始動または再始動した場合に復元されます。

## 次のタスク

アプリケーションを制御するためのポリシーをデプロイすることにより、サービス品質をさらに加えることもできます。詳しくは、CICS ポリシーを参照してください。

アプリケーションの管理方法について詳しくは、プラットフォームおよびアプリケーションの管理を参照してください。

---

## アプリケーション・プログラムのインストール

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。アプリケーション・プログラムをインストールして CICS の下で実行するには、ソース・ステートメントを変換およびコンパイルして、結果のオブジェクト・モジュールを CICS ライブラリーにリンク・エディットし、プログラムをリソースとして CICS に定義する必要があります。

## 手順

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いない)、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を作成し、それらをインストールする。
5. 以下のいずれかの方法を使用して、プログラムをリソースとして CICS に定義する。
  - 単一の CICS 領域の CSD にリソース定義を作成する。リソース定義の概要を参照してください。
  - プログラムの自動インストールを使用する。これにより、CICS は、プログラムがロードされると、単一の CICS 領域でリソース定義を動的に生成します。プログラム、マップ・セット、区画セットの自動インストールを参照してください。
  - CICSplex SM Business Application Services (BAS) を使用してリソース定義を作成する。リソース定義を複数の CICS 領域にインストールできます。BAS の管理 を参照してください。
  - IBM CICS Explorer または IBM Developer for Z を使用して、スタンドアロン CICS バンドル内にリソース定義を作成します。CICS バンドルは、複数の CICS 領域にインストールできます。Defining CICS bundles を参照してください。
  - リソース定義を CICS バンドルに作成し、CICS バンドルをプラットフォームにデプロイするアプリケーションの一部としてパッケージ化およびインストールする。アプリケーションを複数の CICS 領域にインストールできます。また、アプリケーションの複数のバージョンを同時にインストールすることもできます。プラットフォームへのアプリケーションのデプロイを参照してください。

## 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。

### 手順

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いない)、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 連結の 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 連結内のデータ・セットは、拡張アドレス・ボリューム (EAV) DASD ボリュームの拡張アドレッシング・スペース (EAS) に常駐できます。

#### 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。動的プログラム 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、以下のようないくつかの利点があります。

- 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ハサービスを、連続可用性に影響することなく、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ます。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 連結内のデータ・セットは、拡張アドレス・ボリューム (EAV) DASD ボリュームの拡張アドレッシング・スペース (EAS) に常駐できます。

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。

**有効** 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戻され、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戻され、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。

CICS Explorer または IBM Developer for Z を使用して、複数の CICS 領域でのリソース管理およびデプロイメントのために、CICS バンドル内の動的 LIBRARY 連結を定義できます。

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が選択します。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:

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。

このタスクについて

- 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。

#### 手順

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. 修正が入っているデータ・セット (複数可) を含む LIBRARY リソースを定義する。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 を置くことになります。以下のいずれかの方法を使用して、LIBRARY リソースを定義します。
  - CICSplex SM の Business Application Services (BAS) コンポーネント。
  - CICS リソース管理トランザクションの CEDA。
  - CICS のオフライン CSD ユーティリティ・プログラムの DFHCSDUP。

- EXEC CICS CREATE LIBRARY コマンド。
- CICS Explorer。

CICS Explorer を使用して、単一の CICS 領域、CICSplex のデータ・リポジトリ、または CICS バンドルに LIBRARY リソースを定義できます。CICS バンドルに定義されたリソースの管理については、Characteristics of bundled resourcesを参照してください。

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、CICSplex SM WUI、または CICS Explorer を使用してインストールする。
4. 影響を受けるプログラム (複数可) に対して、EXEC CICS SET PROGRAM NEWCOPY または EXEC CICS SET PROGRAM PHASEIN コマンドを発行するか、あるいは CICSplex SM または CEMT における同等の手順を実行する。

#### タスクの結果

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:

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、稼働中の CICS システムに導入すること。

#### このタスクについて

- 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。

#### 手順

1. 新規アプリケーションが入っているデータ・セット (複数可) 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。以下のいずれかの方法を使用して、LIBRARY リソースを定義します。
  - CICSplex SM の Business Application Services (BAS) コンポーネント。
  - CICS リソース管理トランザクションの CEDA。
  - CICS のオフライン CSD ユーティリティ・プログラムの DFHCSDUP。
  - EXEC CICS CREATE LIBRARY コマンド。
  - CICS Explorer。

CICS Explorer を使用して、単一の CICS 領域、CICSplex のデータ・リポジトリ、または CICS バンドルに LIBRARY リソースを定義できます。CICS バンドルに定義されたリソースの管理については、Characteristics of bundled resourcesを参照してください。

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、CICSplex SM WUI、または CICS Explorer 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。

タスクの結果

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:**

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。

このタスクについて

- 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。

手順

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。

タスクの結果

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 内の CICS アプリケーションの再編成:

LIBRARY 内のアプリケーションは、追跡および管理しやすいように再編成することができます。LIBRARY に保管されているアプリケーションは、DD カード・データ・セットとして DFHRPL 連結内で定義されます。

始める前に

必要なアクセス許可があることを確認してください。

このタスクについて

この作業の目的は、アプリケーションの構成を再編成して LIBRARY リソースとしてまとめ、データ・セット名が、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連付けられるようにすることです。

手順

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方法を決定し、アプリケーションごとに 1 つの LIBRARY を使用するか、1 つの LIBRARY に複数のアプリケーションを入れるかを決定します。LIBRARY ごとに 1 つのアプリケーションを入れると、システム構成の保守が簡単かつ容易になります。また、どのようなアプリケーションが LIBRARY 内で複数のデータ・セットを連結させる必要があり、どのようなアプリケーションが単位のデータ・セットを必要とするかということも見極める必要があります。
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します。
3. 動的 LIBRARY 内で定義されるアプリケーションのうち、CICS の始動に関して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します。
4. CICSplex SM、CEDA、DFHCSDUP、または EXEC CICS CREATE の Business Application Services (BAS) コンポーネントを使用して、各アプリケーション (または、アプリケーションをグループ化する場合はアプリケーション・セット) について、動的リソースとなる LIBRARY リソースを定義します。
  - a. 各 LIBRARY について、CICS 領域内の他の LIBRARY との相対的な優先順位を表すランキングを指定します。通常は、アプリケーションが他のライブラリーと交差する部分がないため、デフォルトのランキング値を使用することができます。
  - b. CICS の実行に関してクリティカルな状況を LIBRARY ごとに指定します。CICS の実行に関してクリティカルではないライブラリーについては、デフォルトの状況 (NONCRITICAL) のままにしてください。
  - c. LIBRARY 内のデータ・セットの名前を指定する。
5. CEDA INSTALL LIBRARY コマンドと CICSplex SM WUI のいずれかを使用して、新規 LIBRARY リソースをインストールします。
6. 動的 LIBRARY 連結内にあるアプリケーションを含んでいるデータ・セットを、次の CICS 再始動時に DFHRPL 連結から削除します。

7. (オプション) テストのために DFHRPL よりも LIBRARY が優先されるようにランキングの値を設定した場合は、それぞれの LIBRARY のランキングを、意図する永続値にリセットします。
8. GRPLIST による CICS の再始動時、BAS のインストール時、または CICS の再始動後に、新規 LIBRARY リソースをインストールします。新規 LIBRARY リソースは既に DFHRPL 連結内にはないので、そのようにすると、システムでプログラムが新規リソースからロードされます。

#### タスクの結果

CICS が正常に実行されます。これで、LIBRARY アプリケーションが整理されたので、トラッキングが容易になります。また、どの CICS システムにアプリケーションがインストールされているかも、以前より簡単に判別できます。

**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：**

例えば、PDS の圧縮や、稼働中の 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。

#### このタスクについて

- 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。

#### 手順

1. EXEC CICS SET LIBRARY コマンド、CEMT、CICSplex SM WUI、または CICS Explorer を使用して LIBRARY を無効にします。
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。

#### タスクの結果

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：

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。

このタスクについて

- 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。

手順

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 のランキングを古い LIBRARY のランキングに戻す。その際、SET LIBRARY コマンド、WUI、または CICSplex SM API を使用できます。

タスクの結果

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:

LIBRARY リソースについての情報を検出します。

このタスクについて

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。

プラットフォームにデプロイされたアプリケーションの一部として定義およびインストールされる動的プログラム LIBRARY 連結は、そのバージョンのアプリケーションに対してプライベートになります。EXEC CICS INQUIRE LIBRARY システム・プログラミング・コマンドを使用して、プライベート・リソースを照会またはブラウズすることができます。デフォルトでは、CICS は、EXEC CICS INQUIRE LIBRARY コマンドが発行されるプログラムで使用可能なリソースを検索します。指定したアプリケーションのプライベート・リソースをブラウズすることも選択できます。プライベート・リソースのブラウズについて詳しくは、Browsing resource definitionsを参照してください。ここには、異なるアプリケーション・コンテキストにおけるブラウズの例も含まれます。

## 手順

- 実行中の CICS システムで **EXEC CICS INQUIRE LIBRARY** コマンドを発行します。パブリック LIBRARY リソースの場合のみ、CEMT トランザクションで **INQUIRE LIBRARY** コマンドを使用します。特定の LIBRARY またはプロパティを指定しない場合、この照会では、EXEC CICS INQUIRE コマンドが発行されるプログラムで使用可能なインストール済みのすべての LIBRARY リソースが表示されます。LIBRARY リソースの一部のプロパティを指定する場合は、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。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 のメインメニューから、「**CICS 操作ビュー**」 > 「プログラム操作ビュー」 > 「**LIBRARY、DFHRPL を含む**」をクリックします。「**DSNAME 数**」フィールド名をクリックし、LIBRARY のデータ・セット名レコードを表示します。
- CICS Explorer で、メニュー・バーの「操作」をクリックし、「**LIBRARY**」を選択するか、「**LIBRARY DS 名 (LIBRARY DS Name)**」を選択して、LIBRARY データ・セット名レコードを表示します。CICS システムに接続している場合には、ビューに表示されるリソースは、そのシステムのリソースです。CICSplex に接続している場合には、ビューに表示されるリソースは、設定したブラウズ・スコープによって異なります。
- プラットフォームにデプロイされたアプリケーションの場合、CICS Explorer の「クラウド・エクスプローラー」ビューで、インストールされているアプリケーションをダブルクリックして、アプリケーション記述子エディターを開きます。「リソース」タブを選択してから、「ライブラリー」タブを選択し、アプリケーションのプライベート LIBRARY リソースを表示します。リソースがインストールされている CICS システム、またはリソースが定義されている CICS バンドルで、リソースをフィルタリングできます。「**LIBRARY DS 名 (LIBRARY DS Name)**」タブを選択し、プライベート LIBRARY リソースのデータ・セットの LIBRARY 連結に対して z/OS が生成した DD 名を表示します。

## タスクの結果

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。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:

(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。

始める前に

- 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。

手順

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。

タスクの結果

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** プロパティの修正:

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。

始める前に

- 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。

手順

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。

タスクの結果

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:**

監査ログを使用して、CICS システムの LIBRARY 構成に対する変更を判別します。

このタスクについて

メッセージ DFHLD0555I および DFHLD0556I は CSLB 一時データ・キューに書き込まれ、LIBRARY 構成に対する変更の監査ログを提供します。これらのメッセージは、検索順序に影響する可能性のある変更が生じるたびに、現在の LIBRARY 検索順序を示すために出されます。メッセージ DFHLD0555I が発行され、それに続いて、アクティブな LIBRARY ごとに、検索順にメッセージ DFHLD0556I の 1 つのインスタンスが示されます。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。

### **手順**

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。監査ログには、プラットフォームにインストールされたアプリケーションの一部として定義される LIBRARY 連結の LIBRARY 構成への変更点も示されます。これらの LIBRARY 構成は (アプリケーションに対してプライベートになります)、アプリケーションに対する検索順序に表示されますが、CICS システムに対する検索順序の一部にはなりません。
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。

### **タスクの結果**

DFHLD0555I と DFHLD0556I は、検索順序に影響する可能性がある変更が発生した後の、新しい LIBRARY 検索順序を表示します。アクティブな LIBRARY ごとに、検索順にメッセージ DFHLD0556I のインスタンスが 1 つあります。検索順序が変更されていなくてもこれらのメッセージが表示されるシナリオがいくつかあります。例えば、無効な LIBRARY のランキング値を変更しても、無効な LIBRARY は検索順序に含まれていないため検索順序には影響しませんが、この変更によってこれらのメッセージが出されます。同様に、有効な LIBRARY のランキングを、検索順序でその前後にある LIBRARY のランキングの間の数に変更すると、これらのメッセージが出されますが、順序は変更されません。

## **LIBRARY 構成のタイディアップ:**

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。

始める前に

- CICS が稼働中である。

手順

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。

タスクの結果

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

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、モードを変更する方法、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

コマンド・レベル・プログラムは、16 MB より上だが 2 GB より下もあるアドレス域に常駐することが可能です。

### 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。

このタスクについて

AMODE 属性は、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。プログラムのモードは切り替えが可能であり、ロード・モジュール内の異なる入り口点には異なる AMODE 属性が設定されていますが、一般にはユーザーのプログラムは、このアドレッシング・モードで実行されるように設計されています。

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。

表 87. AMODE および RMODE の有効な指定値

属性	意味
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。
AMODE(64)	64 ビット・アドレッシング・モードを指定します。

表 87. AMODE および RMODE の有効な指定値 (続き)

属性	意味
RMODE(24)	モジュールが 16 MB より下の仮想記憶域に常駐している必要があります。24 ビット依存性を持つ 31 ビット・プログラムまたは 64 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)	モジュールは、16 MB より下の仮想記憶域、および 16 MB より上で 2 GB より下の仮想記憶域にのみ存在できます。

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。

CICS では、64 ビット常駐モード (RMODE(64)) はサポートされておらず、すべての RMODE(64) プログラムが RMODE(31) として処理されます。つまり、RMODE(64) プログラムは、64 ビット (2 GB 境界より上) ストレージではなく、31 ビット (16 MB 境界より上) ストレージにロードされます。

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。

#### 手順

- 次のようにして、リンク・エディット MODE 制御ステートメントで AMODE および RMODE を指定します。  
MODE AMODE(31),RMODE(ANY)
- 以下のいずれかの方法で、AMODE および RMODE を指定します。
  - 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tring。  
//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。
- アセンブラー・プログラムのソース・コードで AMODE ステートメントまたは RMODE ステートメントを指定することができます。また、コンパイラー・オプションを使用して、これらのモードを COBOL で設定することもできます。ご使用の COBOL コンパイラーに関連するアプリケーション・プログラミング情報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

AMODE 属性と RMODE 属性の有効な組み合わせとその効果は、次のとおりです。

表 88. 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

表 88. AMODE および RMODE の有効な指定とその効果 (続き)

AMODE	RMODE	常駐	アドレッシング
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
64	24	16 MB より低位	64 ビット・モード
64	任意	16 MB より高位	64 ビット・モード

以下は、31 ビット標準に従って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の例です。*anyname* はロード・モジュールの名前です。

```
//LKED.SYSIN DD *
MODE AMODE(31),RMODE(ANY)
NAME anyname(R)
/*
//
```

## 永続的に常駐するプログラムの作成

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。

## このタスクについて

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 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 ライブラリー・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 は入出力アクティビティー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。

## 読み取り専用 DSA にあるアプリケーション・プログラムの実行

16 MB 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。16 MB 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、16 MB 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 での常駐に適格なものとなるためには、以下の条件が適用されます。

- プログラムが読み取り専用標準に従って正しく記述されていること。
- プログラムが 31 ビット・アドレッシング標準に従って記述されていること。
- プログラムが 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こと。

プログラムが RDSA での常駐に適格なものとなるためには、以下の条件が適用されます。

- プログラムが読み取り専用標準に従って正しく記述されていること。
- プログラムが RENT 属性および RMODE(24) 常駐属性を指定してリンク・エディットされていること。

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 を実行する場合、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。

個々の言語の ERDSA 要件については、以下のトピックで説明しています。

## アセンブラー

CICS でアセンブラー・プログラムを ERDSA にロードする場合は、RENT アセンブラー・オプション、リンク・エディット RENT 属性、および RMODE(ANY) 常駐モード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。

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であること（例えば、静的ストレージに書き込まれないことなど）を確認してください。読み取り専用でない場合、ストレージ例外が発生します。さらにプログラムは、31 ビットまたは 64 ビットの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に関して考えられる原因については、Causes of protection exceptionsを参照してください。

## AMODE(31) プログラム

AMODE (31) アプリケーション・プログラム用の CICS 提供プロシージャである DFHEITAL には、XREF オプションおよび 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。

以下に例を示します。

```
//ASMPROG JOB
1,user_name,MSGCLASS=A,CLASS=A,NOTIFY=userid
//EITAL EXEC DFHEITAL,
.
. (other parameters as necessary)
.
// LNKPARM='LIST,XREF,RMODE(ANY),RENT'
```

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・エディットする場合は、アセンブラー・プログラムで該当のステートメントを使用します。以下に例を示します。

```
MYPROG CSECT
MYPROG AMODE 31
MYPROG RMODE ANY
```

AMODE および RMODE は、以下の方法で設定することができます。

- 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。以下に例を示します。

```
//EITAL EXEC DFHEITAL,
LNKPARM='LIST,XREF,RENT,AMODE(31),RMODE(ANY)'
```

- バインダーを使用した場合、AMODE と RMODE の指定の競合に関する予期されない警告メッセージが表示されることがあります。

AMODE (64) アプリケーション・プログラム用の CICS 提供プロシージャである DFHEGTAL には、XREF オプションおよび 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。

```
//ASMPROG JOB
1,user_name,MSGCLASS=A,CLASS=A,NOTIFY=userid
//EGTAL EXEC DFHEGTAL,
.
(other parameters as necessary)
.
// LNKPARM='LIST,XREF,RMODE(ANY),RENT'
```

アプリケーション・プログラムを AMODE(64) および RMODE(ANY) としてリンク・エディットする場合は、アセンブラー・プログラムで該当のステートメントを使用します。以下に例を示します。

AMODE および RMODE は、以下の方法で設定することができます。

- ```
//EITAL EXEC DFHEGTAL,
LNKPARM='LIST,XREF,RENT,AMODE(64),RMODE(ANY)'
```

- バインダーを使用した場合、AMODE と RMODE の指定の競合に関する予期されない警告メッセージが表示されることがあります。

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、RENT コンパイラ・オプションを指定してプログラムをコンパイルおよびリンク・エディットします。

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 (903 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'
```

シーケンス番号を含む C または C++ コード、例えば、CICS とともに配布される C または C++ のサンプル・プログラムをコンパイルする場合は、CPARM パラメーターを指定変更して SEQ を指定する必要があります。以下に例を示します。

```
//EXAMPLE EXEC DFHYITEL,
// CPARM='/CXX OPT(1) SEQ NOMAR SOURCE'
//TRN.SYSIN DD *
... source code goes here ...
/*
//LKED.SYSIN DD *
NAME EXAMPLE(R)
/*
```

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、以下の理由により、自動的に ERDSA に対して適格になります。

- CBLCARD 変換プログラム・オプション (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。NOCBLCARD 変換プログラム・オプションを使用する場合、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・スタブを組み込むと、自動的

に、プログラムが AMODE(31) および RMODE(ANY) としてリンク・エディットされます。903 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。

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 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。以下に例を示します。

```
//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。

- PL/I コンパイル・プロシージャは、RENT コンパイラ・オプションを指定する必要があります。PL/I をコンパイルする CICS 提供のプロシージャ (例えば DFHYITPL) は、このオプションを自動的に組み込みます。
- PL/I コンパイラは、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・スタブを組み込むと、自動的に、プログラムが AMODE(31) および RMODE(ANY) としてリンク・エディットされます。903 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。

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 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。以下に例を示します。

```
//PLIPROG JOB
1,user_name,MSGCLASS=A,CLASS=A,NOTIFY=userid
//YITPL EXEC DFHYITPL,
.
  (other parameters as necessary)
.
// LNKPARM='LIST,XREF,RENT'
```

プログラムが間違いなく読み取り専用であること (例えば、静的ストレージに書き込まれないことなど) を確認していない場合は、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場合、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に関して考えられる原因については、Causes of protection exceptions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

アプリケーション・プログラムで BMS マップ・セットを使用する方法について概説します。

このタスクについて

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前に、以下の手順を実行してください。

手順

- 1029 ページの『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 CICSTS55.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むには、以下の手順を実行します。

1. 物理マップ・セットおよびシンボリック・マップ・セットと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。
2. 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 CICSTS55.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、CICSTS55.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、CICSTS55.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。

3. 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。

次のタスク

マップ・セットのインストールの詳細については、1029 ページの『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、459 ページの『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atalog式プロシージャで提供します。

CICS をインストールした後に、CICSTS55.CICS.SDFHPROC ライブラリーにインストールされているプロシージャを、プロシージャ・ライブラリーにコピーします。各プロシージャ名は、DFHwxTyL という形式をとります。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)、コンパイラのタイプ、およびプログラム言語に応じて決まります。以下の表に、プロシージャ名を示します。

表 89. 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: 非 *Language Environment* 準拠のコンパイラー

| 言語 | スタンドアロンの変換プログラム | EXCI |
|--------|--|----------|
| アセンブラー | DFHEITAL (AMODE(24) および AMODE(31) アプリケーション)

DFHEGTAL (AMODE(64) アプリケーション) | DFHEXTAL |

表 90. 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: *Language Environment* 準拠のコンパイラー

| 言語 | スタンドアロンの変換プログラム | 統合変換プログラム | スタンドアロン変換プログラムを使用した EXCI | 統合変換プログラムを使用した EXCI |
|----------------------------|-------------------------------|-------------------------------|--------------------------|---------------------|
| C | DFHYITDL (注 1 (1012 ページ) を参照) | DFHZITDL (注 1 (1012 ページ) を参照) | DFHYXTDL | DFHZXTDL |
| XPLINK コンパイラー・オプションを使用した C | DFHYITFL (注 2 (1012 ページ) を参照) | DFHZITFL (注 1 (1012 ページ) を参照) | - | - |
| C++ | DFHYITEL (注 1 (1012 ページ) を参照) | DFHZITEL (注 1 (1012 ページ) を参照) | DFHYXTEL | DFHZXTEL |

表 90. 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: *Language Environment* 準拠のコンパイラ (続き)

| 言語 | スタンドアロンの変換プログラム | 統合変換プログラム | スタンドアロン変換プログラムを使用した EXCI | 統合変換プログラムを使用した EXCI |
|-----------------------------|--------------------|--------------------|--------------------------|---------------------|
| XPLINK コンパイラ・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 3 を参照) | DFHYITVL | DFHZITCL (注 2 を参照) | DFHYXTVL | DFHZXTCL |
| PL/I (注 4 を参照) | DFHYITPL (注 2 を参照) | DFHZITPL (注 2 を参照) | DFHYXTPL | DFHZXTPL |

注:

1. DFHYITEL は C でも使用できます。その場合は、C コンパイラの正しい名前を **COMPILER** パラメーターで指定する必要があります。
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. 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です。
4. 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です。
5. 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。

注: このプロシージャにはマクロ DFHLI000 が必要です。

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 ロードーは、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。

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、1025 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・プログラムの変換、アセンブル、およびリンク・エディット

アセンブラー言語で書かれた AMODE(24) および AMODE(31) 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 プロシージャまたは DFHEXTAL プロシージャを使用することができます。アセンブラー言語で書かれた AMODE(64) 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、DFHEGTAL プロシージャを使用することができます。

このタスクについて

図 169 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・プログラムであり、それらのプログラムの AMODE 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 提供プロシージャの名前については、1011 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 提供プロシージャの使用』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)
/*
//
```

図 169. DFHEXTAL プロシージャを呼び出すジョブ制御ステートメントの例

anyname は、ロード・モジュールの名前です。

注:

1. プログラムを読み取り専用 DSA にインストールするには、1005 ページの『読み取り専用 DSA にあるアプリケーション・プログラムの実行』で詳細を参照してください。

使用するプログラムを LPA からインストールするため、以下のオプションを追加します。

- DFHEXTAL プロシージャの 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
- DFHEXTAL プロシージャ呼び出しの 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

(1004 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み可能な変換プログラム・オプションの詳細については、901 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。

以下に、CICS 提供プロシージャ DFHEGTAL を使用して AMODE(64) 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・エディットするためのジョブ制御ステートメントの例を示します。

```
//APPLPROG EXEC DFHEGTAL
//TRN.SYSIN DD *
      . Application program
      .
/*
//LKED.SYSIN DD *
      ENTRY program_name
      NAME program_name(R)
/*
```

program_name は、AMODE(64) アプリケーションの名前です。

1016 ページの図 170 は、コマンド・レベル変換プログラムが CICS.SDFHLOAD への参照を使用してアセンブラ・ソース・プログラムを処理し、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。続いてアセンブラが CICS.SDFHMAC への参照を使用してこの出力ファイルを処理し、アセンブラ・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、リンケージ・エディターがこの出力ファイルを処理し、リンケージ・エディター・リストとアプリケーション・ライブラリーに保管されるロード・モジュールが生成されます。

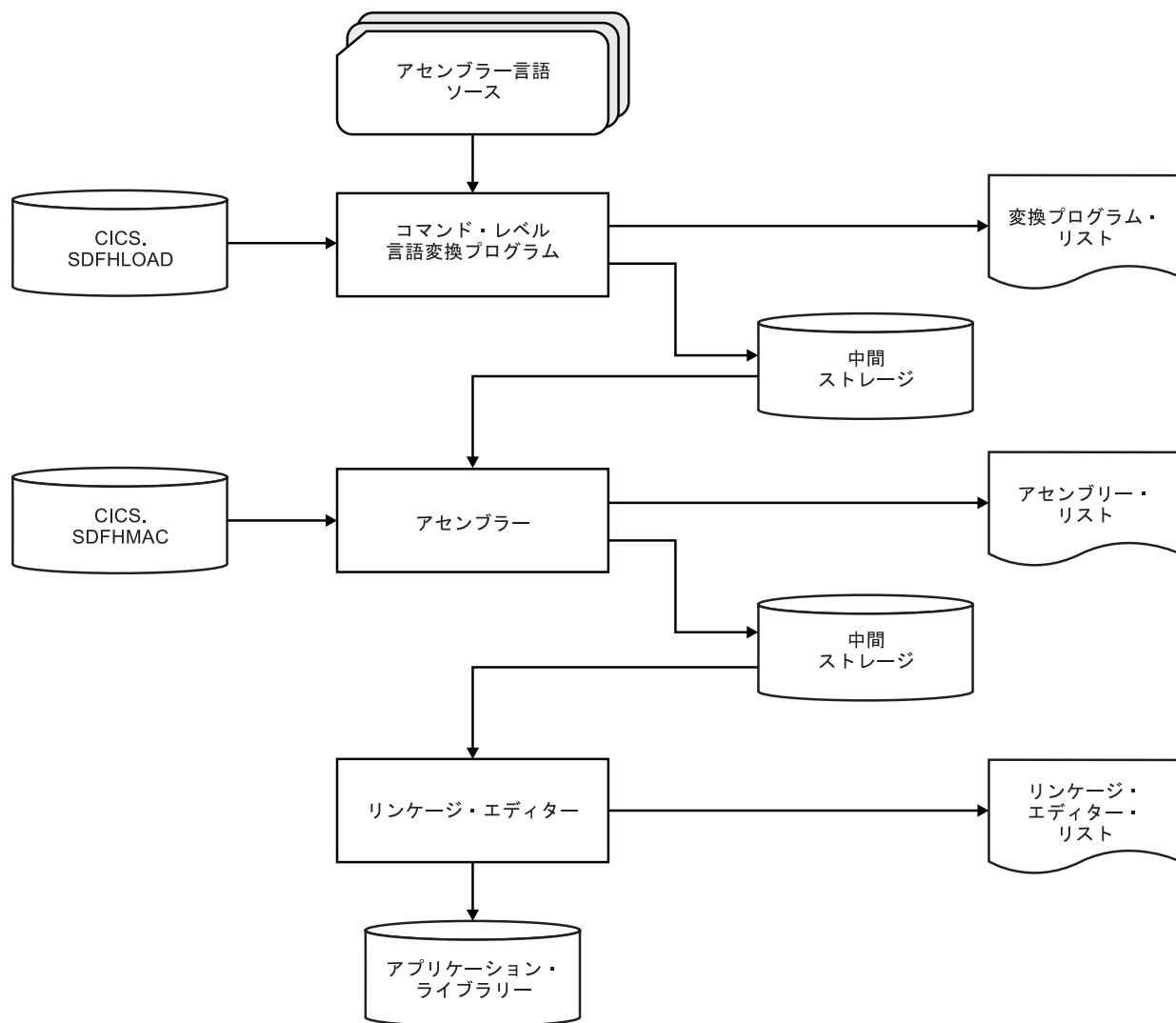
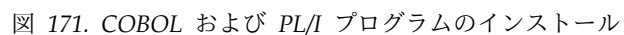


図 170. DFHEITAL または DFHEGTAL プロシーチャーを使用したアセンブラー言語プログラムのインストール

COBOL アプリケーション・プログラムのインストール

この図は、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チャーの制御のフローを示しています。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。



これらの例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または統合変換プログラムで 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。

第 16 章 アプリケーションのデプロイ 1017

```
//jobname JOB accounting info,name,MSGLEVEL=1
// EXEC PROC=procname1
//TRN.SYSIN DD *2
CBL XOPTS(Translator options . . .)3.
COBOL source statements.
/*
//LKED.SYSIN DD *4
NAME anyname(R)
/*
//
```

図 172. DFHYITVL または DFHYXTVL プロシーチャーを呼び出すジョブ制御ステートメントの例

ここで、*procname* はプロシーチャーの名前であり、*anyname* はユーザーのロード・モジュール名です。

プロシーチャー DFHZITCL または DFHZXTCL 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 173 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 PROC=procname,PROGLIB=dsname1
//COBOL.SYSIN DD *
.. COBOL source statements.
/*
//LKED.SYSIN DD *
NAME anyname(R)
/*
//
```

図 173. プロシーチャー DFHZITCL または DFHZXTCL を使用するジョブ制御ステートメントの例

ここで、*anyname* はユーザーのロード・モジュール名です。

1. 変換プログラム・オプション: コンパイル・ステップに必要な COBOL 機能のバージョンに従って、COBOL3 または COBOL2 の変換プログラム・オプションを指定します。
2. 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
| および NODYNAM が必要です。

| 注: コンパイラーの LIB オプションは、COBOL 5 以降では不要になり、除去
| されました。以前のバージョンの COBOL の場合は、このオプションを手動で
| 復元する必要があります。

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 で提供される COBOL プロシーチャーにある 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。以下に例を示します。

```
| //COB EXEC PGM=IGYCRCTL,REGION=&REG,  
| // PARM='NODYNAM,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,MAP,CICS(''COBOL3'')'
```

注: 統合変換プログラムに対して PARM スtring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 で提供される COBOL プロシージャでは、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。以下に例を示します。

```
EXEC PROC=procname  
,PARM.COB='SIZE=512K,BUF=16K,.,.,.'
```

以下の方法のいずれかを使用して、コンパイラー・オプションを変更します。

- COBOL プロシージャの 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。

- COBOL プロシージャ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。
- COBOL のインストール済み環境のデフォルトのマクロ IGYCOPT を使用する。このマクロは、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 NOCBLCARD を指定した場合) に必要とされます。
- ユーザーの COBOL プログラム用のコンパイラー・オプションを含むデータ・セットを定義する。このデータ・セットには、CICS コンパイラー・オプションおよびサブパラメーターを含める必要があります。

次のいずれかの方法で、SYSOPTF DD ステートメントをコーディングします。

```
// SYSOPTF DD DSNAME=dsname,UNIT=SYSDA,VOLUME=(subparms),DISP=SHR
```

このコード・フラグメントでは、コンパイラー・オプションはデータ・セット *dsname* に保管されます。

```

//COBOL EXEC PGM=IGYCRCTL,REGION=4M,PARM=(OPTFILE)
//SYSOPTF DD *
APOST
TRUNC(OPT)
CICS('COBOL3,SP')
NODYNAM
RENT
LIST
MAP
XREF
OPT
TEST(ALL,SEPARATE)
//STEPLIB DD DSN=PP.COBOL390.V410.SIGYCOMP,DISP=SHR

```

このコード・フラグメントでは、コンパイラー・オプションは、**OPTFILE** パラメーターの後にコードに直接配置されます。

SYSOPTF ステートメン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。

変換プログラム・オプション CBLCARD|NOCBLCARD について詳しくは、901 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。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資料を参照してください。

3. 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。
4. 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、COBOL プロシーチ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、901 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。

5. 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、1005 ページの『読み取り専用 DSA 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、COBOL プロシージャ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。詳しくは、1004 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

DFHYxTPL プロシージャを使用して、独立した変換プログラムにより PL/I アプリケーションを処理できます。x の値は、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応じて決まります。別の方法として、DFHZxTPL プロシージャを使用して、統合変換プログラムを呼び出すこともできます。

1017 ページの図 171 は、PL/I プログラム用の、カタログ式プロシージャの制御のフローについて説明しています。

PL/I プログラムの準備について詳しくは、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。

C アプリケーション・プログラムのインストール

この図は、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の制御のフローを示しています。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。

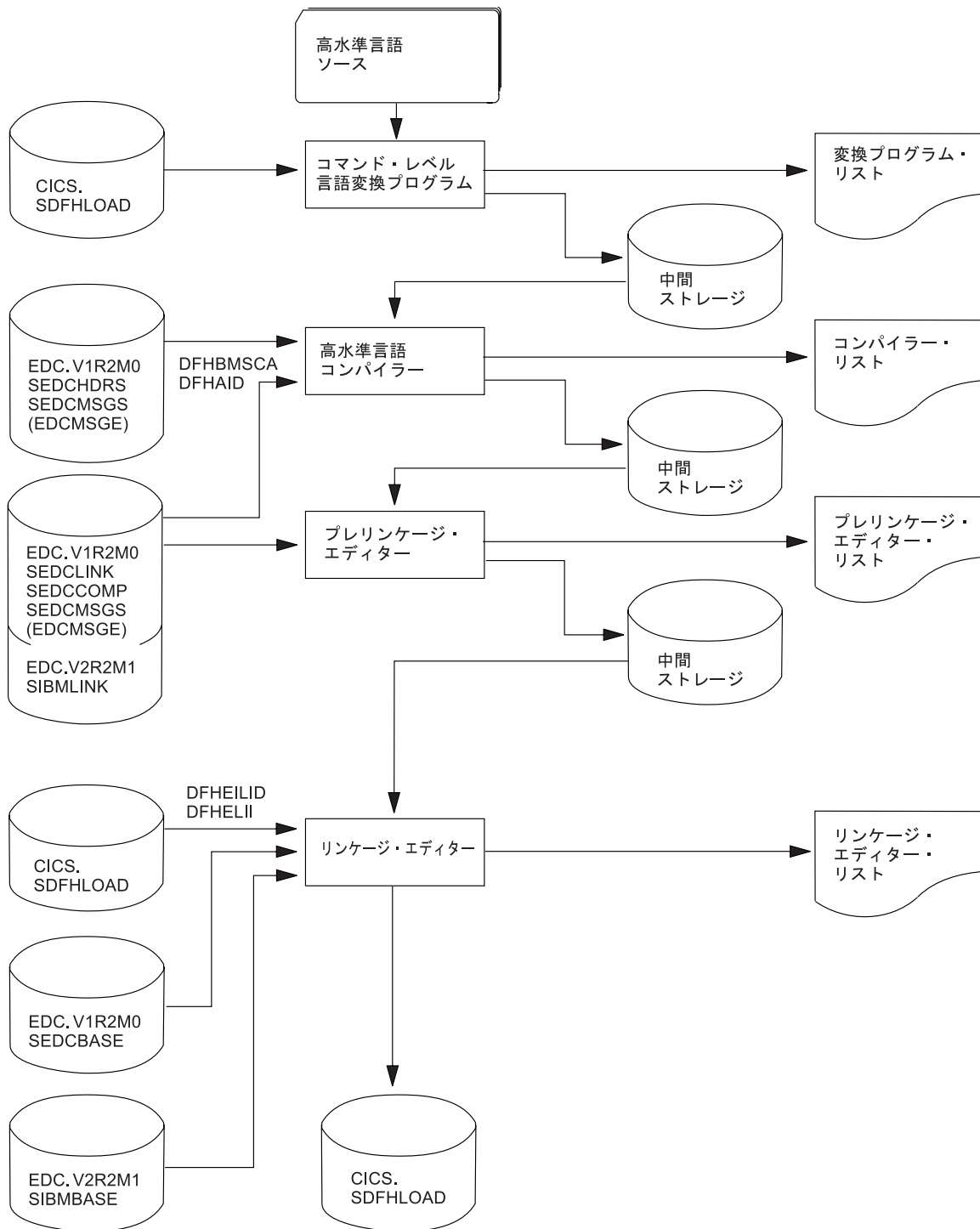


図 174. DFHYxTzL プロシーチャーを使用した C プログラムのインストール

変換プログラム、コンパイラ、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じます。C ライブラリーは、コンパイラ、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・オプションを選択する場合は、この図のプリリンクのステップがありません。

C プログラムをインストールする前に、C ライブラリーおよびコンパイラーをインストールして、C の CICS サポートを生成しておく必要があります。(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ために、これらの例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 提供プロシージャの名前については、1011 ページの表 90を参照してください。

```
//jobname JOB accounting info,name,MSGLEVEL=1
// EXEC PROC=DFHYxTzL1
//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 175. DFHYxTzL プロシージャを呼び出す 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: プロシージャを呼び出す EXEC ステートメントのパラメーター指定変更 **CPARM**、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。

C または C++ のソースを JCL プロシージャ DFHYITEL、DFHZITEL、または DFHYXTEL で利用できるようにする場合、ソースの入力マージンを制限する必要があります。ソースの入力マージンは、次のように制限できます。

- 次の **pragma** の宣言を指定することによって、C または C++ のソースを直接変更します。

```
#pragma margins(m,n)
```

ここで、*m* および *n* は、C または C++ のソースが配置されている列です。例えば、**#pragma margins(1,72)** です。

- JCL プロシージャで **CPARM** 指定変更オプションを変更します。ソースがシーケンス番号を持っているかどうかによって、SEQ または NOSEQ を指定します。
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、901 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、1005 ページの『読み取り専用 DSA 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 プロシージャ呼び出しの LNKPARM パラメーターに追加します (詳細については、1004 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 プロシージャが AMODE(31) を指定するようになります。

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 176 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=dsname1
//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 176. 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を使用して統合変換プログラムを呼び出すには、以下の 1025 ページの図 177 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=dsname1
//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 177. 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:

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。このエラーを防ぐには、`dfhexec` を含む古いコードを再コンパイルする方法、またはジョブでプリリンカー `RENAME` 制御ステートメントを使用する方法の 2 つあります。

このタスクについて

次のサンプル JCL は、`RENAME` 制御ステートメントを使用する方法を示しています。

```
//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 178. `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

提供されるカタログ式プロシージャは、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 を作成するためのモデルとして使用することができます。

プロシージャーは、CICSTS55.CICS.SDFHPROC ライブラリーにインストールされています。

ここでは、変換プログラムに関する重要点、およびプログラムの主要カテゴリーそれぞれについて概説します。わかりやすくするため、プログラムが CICSTS55.CICS.SDFHLOAD または IMS.PGMLIB にロードされるものとして説明します。プログラムは任意のライブラリーにロードすることができますが、そのようなことが可能なのは、ライブラリーが CICS ジョブ・ストリームの DFHRPL 連結または動的 LIBRARY 連結に組み込まれている場合、または (独立型 IMS バッチ・プログラムの) バッチ・ジョブ・ストリームの STEPLIB ライブラリー連結に組み込まれている場合のみです。

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。独自の命名規則を IMS ライブラリーに使用する場合は、IMS ライブラリーを適宜、名前変更する必要があります。

変換プログラムの要件

CICS 変換プログラムは、256 KB 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) には、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、CICSTS55.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。

図 179 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 DFHYITVL を基にした JCL 例とインライン・プロシージャを示しています。このプロシージャは、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(DFHELIB)
```

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マークされます。

図 179. COBOL プログラムをインストールするユーザー定義 JCL の例

```

/* The following JCL could be used to execute this procedure
/*
//APPLPROG EXEC MYITVL,
// INDEX='CICSTS55.CICS
// PROGLIB='CICSTS55.CICS.SDFHLOAD',
// DSCTLIB='CICSTS55.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 data sets

//TRN.SYSIN DD *
/* .
/* . アプリケーション・プログラム
/* .
/*
//LKED.SYSIN DD *
INCLUDE SYSLIB(DFHELIB)
NAME anyname(R)
/*
//MYITVL PROC SUFFIX=1$, Suffix for translator module
// INDEX='CICSTS55.CICS
', Qualifier(s) for CICS libraries
// PROGLIB='CICSTS55.CICS.SDFHLOAD', Name of o/p library
// DSCTLIB='CICSTS55.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 data sets
/*

/* 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 data set &PROGLIB

//TRN EXEC PGM=DFHECP &SUFFIX,,
// PARM='COBOL3',
// REGION=&REG

//STEPLIB DD DSN=&INDEX..SDFHLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=&OUTC
//SYSPPUNCH DD DSN=&&SYSCIN,
// DISP=(,PASS),UNIT=&WORK,

```

```

// DCB=BLKSIZE=400,
// SPACE=(400,(400,100))
//*
//COB EXEC PGM=IGYCRCTL,REGION=&REG,
// PARM='NODYNAM,OBJECT,RENT,APOST,MAP,XREF'
//*      Note:
//*      The compiler LIB option is no longer required for
//*      COBOL 5 and later, and the option has been removed.
//*      For earlier versions of COBOL, this option must be
//*      manually reinstated.
//*
//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))
//SYSPRINT DD SYSOUT=&OUTC
//SYSLIN DD DSN=&&COPYLINK,DISP=(OLD,DELETE)
// DD DSN=&&LOADSET,DISP=(OLD,DELETE)
// DD DDNAME=SYSIN
//PEND
//*
```

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呼び出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、CICSTS55.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。CICSTS55.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 プログラムを準備する場合は、CICS 提供プロシージャ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。

マップ・セットおよび区分セットのインストール

CICS の基本マッピング・サポート (BMS) 機能を使用して、マップ・セットおよび区分セットをアセンブルおよびリンク・エディットします。BMS マクロを使用して、BMS マップから生成された HTML テンプレートをインストールできます。

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。

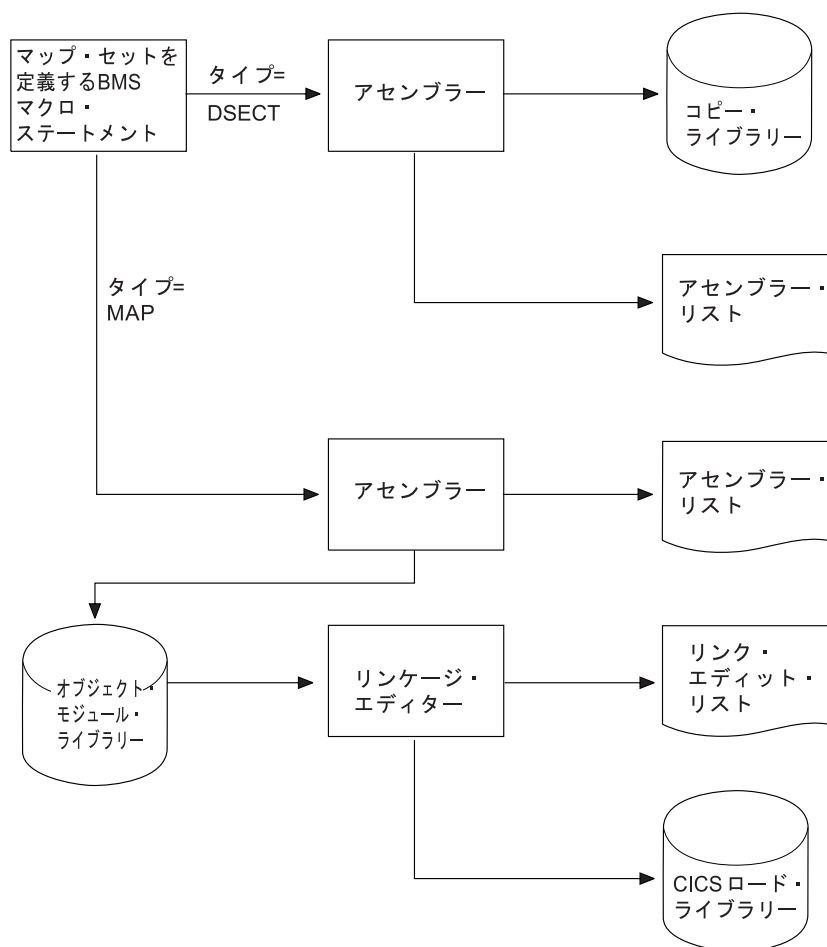


図 180. マップの準備

どのような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) 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 の詳細については、Screen Definition Facility II Library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 16 MB 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、16 MB 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』
- 1040 ページの『区分セットのインストール』
- 1041 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』

マップ・セットのインストール

これらの例を使用して、物理マップ・セットおよびシンボリック記述マップ・セットを別々および一緒にインストールする方法を確認します。

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』
- 1033 ページの『物理マップ・セットのインストール』
- 1035 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』
- 1037 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、物理マップ・セットとシンボリック記述マップ・セットの 2 つのタイプのマップ・セットを準備する必要があります。

- 物理マップ・セットは、アプリケーション・プログラムで使用される標準の装置独立形式のデータを、端末で必要な装置依存形式のデータに変換するために BMS で使用します。
- シンボリック記述マップ・セットは、ユーザー・データの標準の装置独立形式を定義するため、アプリケーション・プログラムで使用します。これは、アセンブラ言語における 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、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 ステートメントで、DFHMSD マクロの TYPE オペランド、または SYSPARM オペランドのいずれかを使用して、マップ・セットの 2 つのタイプを区別できます。

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 91を参照してください。

表 91. 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 |
|---------------------------------|-------------------------------------|-------------------------|
| 位置合わせされるシンボリック記述マップ・セット (DSECT) | A | 指定されていません |
| | A | DSECT |
| | ADSECT | 任意 (SYSPARM が使用される) |
| 位置合わせされる物理マップ・セット | A | 任意のマップ (SYSPARM が使用される) |
| | AMAP | |

表 91. 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 |
|----------------------------------|-------------------------------------|---|
| 位置合わせされないシンボリック記述マップ・セット (DSECT) | 指定されていません
指定されていません
DSECT | 指定されていません
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 | 指定されていません
MAP | 任意のマップ (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:

物理マップ・セットを再アセンブルすることによって、固定された拡張データ・ストリーム属性を使用でき、動的属性の処理に関しては、物理記述マップ・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することによって使用できます。

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

これらの例は、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と、サンプルのジョブ・ストリームを示しています。

1034 ページの図 181 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。

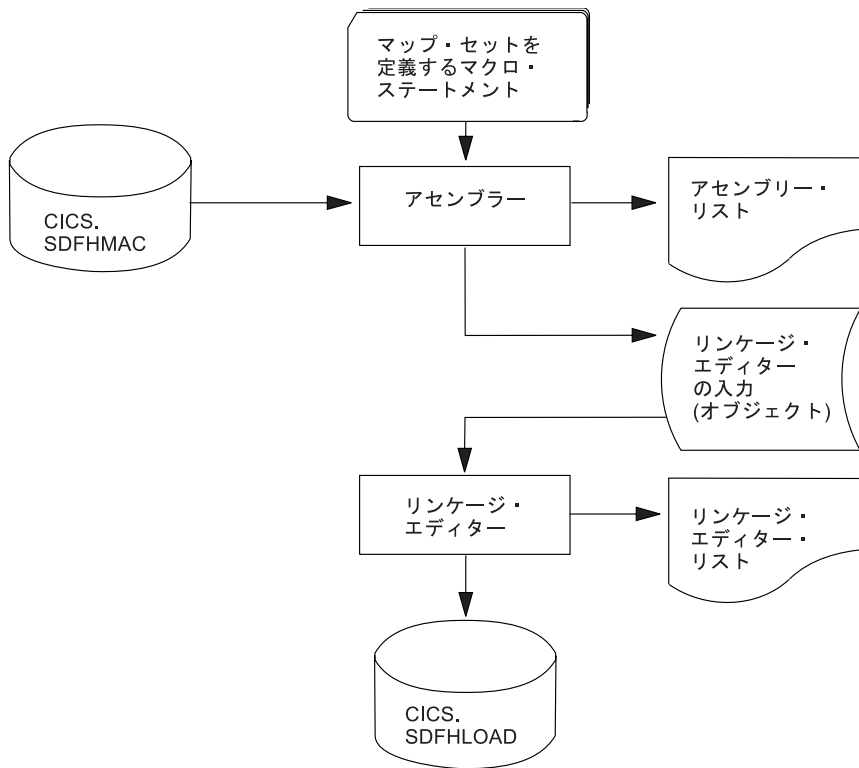


図 181. 物理マップ・セットのインストール

図 182 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)3
NAME mapsetname(R)4
/*
//
```

図 182. 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

注

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、RENTPGM 初期設定パラメーターで RENTPGM=PROTECT が指定される

と、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。原則として、送信先が 3270 装置または LU1 装置に限られているマップ・セットの場合は、RENT オプションまたは REFR オプションを使用してください。CICS で使用できるストレージ保護機能の詳細については、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 16 MB 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 16 MB 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、1041 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

これらの例は、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のステップを示しています。

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 183 に、BMS 用のシンボリック記述マップ・セットの準備を示します。

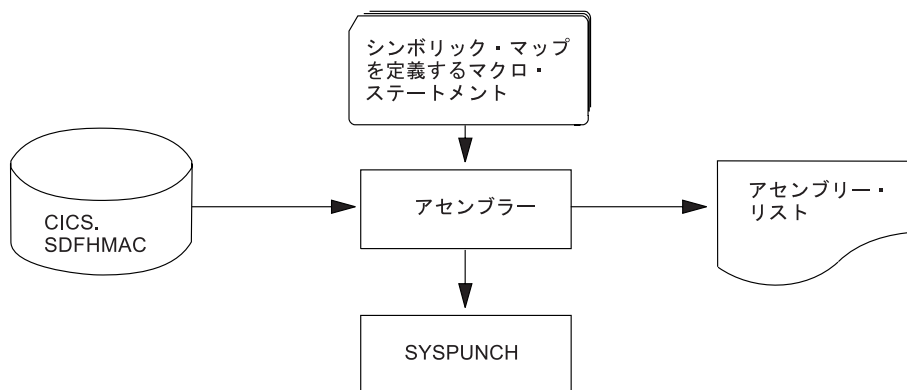


図 183. DFHASMVS プロシージャーを使用したシンボリック記述マップ・セットのインストール

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。これを最初に行うと、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 184 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 184. 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 184 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・コピーを取得するには、前述の例の //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。以下に例を示します。

```
//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

これらの例は、DFHMAPS プロシーチャーを使用した物理マップ・セットおよびシンボリック記述マップ・セットのインストールのステップを示しています。

1038 ページの図 185 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 プロシーチャーを示しています。DFHMAPS プロシーチャーは、1038 ページの図 185 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 プロシーチャーでのこの宛先は CICSTS55.CICS.SDFHMAC ライブラリーです。

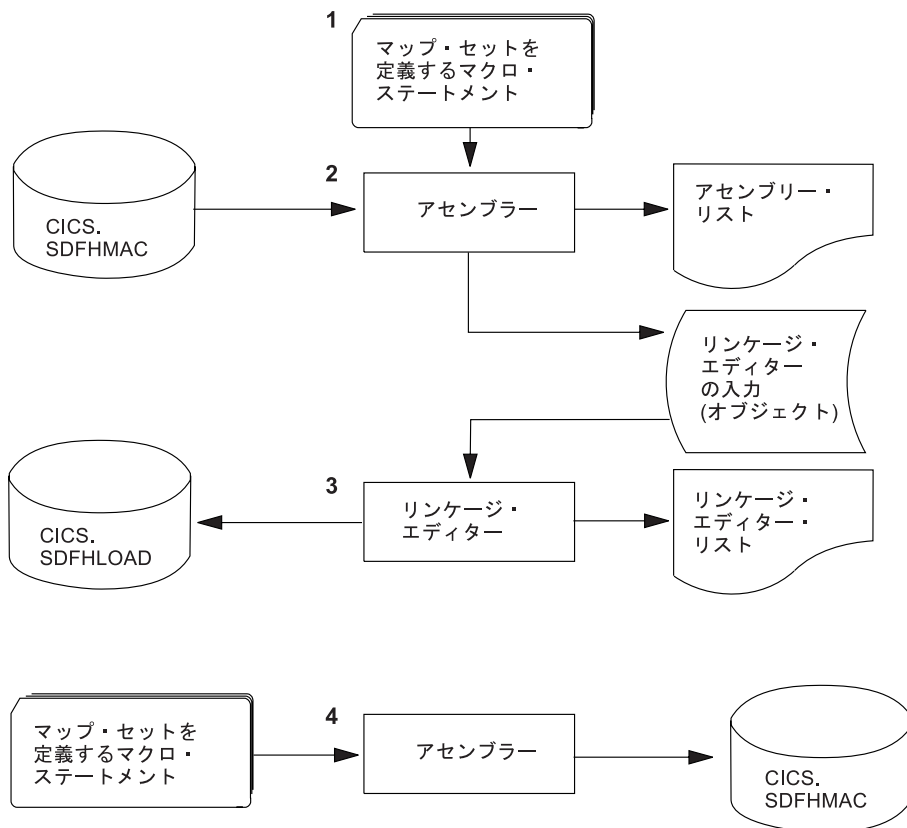


図 185. 物理マップ・セットおよびシンボリック記述マップ・セットの同時インストール

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:

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。

このタスクについて

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 **CICSTS55.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**:

この例は、物理マップ・セットとシンボリック記述マップ・セットを合わせてインストールするのに必要な **JCL** ジョブ・ストリームを示します。

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 186 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 186. 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ています。

DFHMAPS プロシージャは、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 186 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 プロシージャでは、シンボリック記述マップ・セットの出力 (SYSPUNCH) を、CICSTS55.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:

この例は、CSECT 名および AMODE と RMODE ステートメントの両方をマップ・アセンブリーに追加する方法を示しています。

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 16 MB 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,HIGHLIGHT), C
DSATTS=(COLOR,HIGHLIGHT)
SPACE
DFH0STM DFHMDI SIZE=(24,80)...
SPACE
DFHMSD TYPE=FINAL
END.
/*
//

```

図 187. 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できます。記号記述区分セットの概念について説明しません。

図 188 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 188. 区分セットのアセンブルおよびリンク・エディット

注

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、**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、キー 0 保護ストレージにロードされます。

CICS で使用できるストレージ保護機能の詳細については、ストレージ保護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 16 MB 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完全なリストを提供するプログラミング情報については、CICS コマンドの LENGTH オプション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、CEDA DEFINE コマンドおよび CEDA INSTALL コマンドで説明するように、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

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。

このタスクについて

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は、PROGRAM リソースを参照してください。

アプリケーションのテスト

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。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。

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、プログラム論理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ができます。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いえです。

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 (330 ページの『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。

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. 1044 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

このリストは、アプリケーションおよびシステム・テーブル項目を準備する際に考慮する必要がある事項を示しています。

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. 使用するトランザクションごとに PROFILE リソース定義を作成して、次に定義をインストールするようにします。
5. アプリケーション内のトランザクションごとに TRANSACTION リソース定義を作成して、定義がインストール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プログラムごとに PROGRAM リソース定義を作成し、定義がインストール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マップ・セットごとに MAPSET リソース定義を作成し、各定義がインストールされていることを確認します。
8. 使用するファイルごとに FILE リソース定義を作成して、各定義がインストールされていることを確認します。
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・キューをアプリケーションで使用するよう定義します。
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

このリストは、デバッグのためにシステムを準備する際に考慮する必要がある事項を示しています。

1. GRPLIST システム初期設定パラメーター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なります。
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・オプションのセットアップについて詳しくは、CICS トレースの使用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・オプションのセットアップについては、Using dumps in problem determination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。DFHDU720 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、それら进行处理するための 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。統計の使用について詳しくは、CICS 統計の概要を参照してください。

JVM サーバーへのアプリケーションのデプロイ

Java アプリケーションを JVM サーバーにデプロイするには、そのアプリケーションを正常にインストールして実行するために適切にパッケージ化する必要があります。アプリケーションのパッケージ化とデプロイには、IBM CICS SDK for Java を使用できます。

Java アプリケーションをデプロイするには、以下のようないくつかのオプションがあります。

- OSGi フレームワークを実行する JVM サーバーの中に、アプリケーションの OSGi バンドルを含む 1 つ以上の CICS バンドルをデプロイする。
- Liberty JVM サーバーの中に、1 つ以上の WAR ファイルを含む 1 つ以上の CICS バンドルをデプロイする。
- Liberty JVM サーバーの中に、Enterprise Bundle Archive (EBA) ファイルを含む 1 つ以上の CICS バンドルをデプロイする。

- EAR ファイルを含む 1 つ以上の CICS バンドルを Liberty JVM サーバーにデプロイする。
- プラットフォームに、CICS バンドルと OSGi バンドルで構成されるアプリケーション・バンドルをデプロイする。

JVM サーバーにおける OSGi バンドルのデプロイ

JVM サーバーに Java アプリケーションを配置するには、ターゲット JVM サーバーの OSGi フレームワークにそのアプリケーションの OSGi バンドルをインストールする必要があります。

始める前に

アプリケーションの OSGi バンドルを含む CICS バンドルは、zFS にデプロイされる必要があります。ターゲット JVM サーバーが CICS 領域で有効になっている必要があります。

このタスクについて

CICS バンドルには、1 つ以上の OSGi バンドルを含むことができます。CICS バンドルは配置の単位であるため、すべての OSGi バンドルは、BUNDLE リソースの一部として一緒に管理されます。また、OSGi フレームワークは、依存関係とバージョン管理方式の管理を含めて、OSGi バンドルのライフサイクルを管理します。

1 つの Java アプリケーション・コンポーネントを構成するすべての OSGi バンドルを、必ず同じ CICS バンドル内にデプロイしてください。OSGi バンドル相互間に依存関係がある場合は、それらを同じ CICS バンドルに配置してください。CICS BUNDLE リソースをインストールする際に、CICS によって、OSGi バンドル間のすべての依存関係が解決されていることが確認されます。

共通コードのライブラリーを含む OSGi バンドルへの依存関係がある場合、そのライブラリー用に 1 つの別個の CICS バンドルを作成してください。この場合、そのライブラリーを含む CICS BUNDLE リソースを最初にインストールすることが重要です。Java アプリケーションをインストールしてから、その Java アプリケーションが依存する CICS バンドルをインストールすると、OSGi フレームワークは、その Java アプリケーションの依存関係を解決できません。

OSGi バンドルを含む CICS バンドルを Liberty JVM サーバーの中にインストールしようと試みないでください。このような構成はサポートされていません。その代わりに、OSGi バンドルを Web アプリケーションと一緒にエンタープライズ・バンドル・アーカイブ (EBA) にパッケージ化できます。または WebSphere Liberty プロファイル・バンドル・リポジトリを使用して、Liberty JVM サーバー内のすべての Web アプリケーションに対して OSGi バンドルを使用可能にすることもできます。

手順

1. zFS 内のバンドルのディレクトリーを指定する BUNDLE リソースを作成します。

- a. CICS SM パースペクティブで、CICS Explorer メニュー・バーの「定義」 > 「バンドル定義」をクリックして、「バンドル定義」ビューを開きます。
 - b. そのビュー内の任意の場所を右クリックし、「New」をクリックして「New Bundle Definition」ウィザードを開きます。そのウィザードのフィールドに、BUNDLE リソースの詳細を入力してください。
 - c. BUNDLE リソースをインストールします。リソースを Enabled 状態または Disabled 状態のどちらかでインストールできます。
 - DISABLED 状態でリソースをインストールすると、CICS は OSGi バンドルをフレームワークにインストールし、依存関係を解決しますが、バンドルを開始しようとしません。
 - ENABLED 状態でリソースをインストールすると、CICS は OSGi バンドルをインストールし、依存関係を解決し、OSGi バンドルを開始します。遅延的なバンドル・アクティベーターが OSGi バンドルに含まれる場合、別の OSGi バンドルによって呼び出されるまでは、OSGi フレームワークはバンドルを開始しようとしません。
2. オプション: BUNDLE リソースがまだ ENABLED 状態でない場合、そのリソースを使用可能にして、フレームワークで OSGi バンドルを開始します。
3. CICS Explorer メニュー・バーの「Operations」 > 「Bundles」をクリックして、「Bundles」ビューを開きます。BUNDLE リソースの状態を確認します。
- BUNDLE リソースが ENABLED 状態である場合、CICS はバンドル内のすべてのリソースを正常にインストールできました。
 - BUNDLE リソースが DISABLED 状態である場合、CICS はバンドル内の 1 つ以上のリソースをインストールできませんでした。

BUNDLE リソースが ENABLED 状態でインストールできなかった場合、BUNDLE リソースのバンドル・パーツを確認してください。いずれかのバンドル・パーツが UNUSABLE 状態である場合、CICS は OSGi バンドルを作成できませんでした。通常、この状態は、zFS で CICS バンドルに問題があることを示します。その BUNDLE リソースを破棄し、問題を修正してから、BUNDLE リソースを再度インストールする必要があります。

4. CICS Explorer メニュー・バーで「Operations (操作)」 > 「Java」 > 「OSGi Bundles (OSGi バンドル)」をクリックして、「OSGi Bundles (OSGi バンドル)」ビューを開きます。OSGi フレームワークにインストールされた OSGi バンドルおよびサービスの状態を確認します。
 - OSGi バンドルが STARTING 状態である場合、バンドル・アクティベーターが呼び出されましたが、まだ戻っていません。OSGi バンドルに遅延活動化ポリシーがある場合、OSGi フレームワークで呼び出されるまで、そのバンドルはこの状態のままです。
 - OSGi バンドルと OSGi サービスがアクティブである場合、Java アプリケーションは作動可能です。
 - OSGi サービスが非アクティブである場合、CICS は、その名前を持つ OSGi サービスが OSGi フレームワークに既に存在することを検出した可能性があります。

- BUNDLE リソースを使用不可にすると、OSGi バンドルは RESOLVED 状態に移ります。
- OSGi バンドルが INSTALLED 状態である場合、OSGi バンドル内の依存関係を解決できなかったために、このバンドルは開始されなかったか開始に失敗しました。

5. 1052 ページの『JVM サーバー内の Java アプリケーションの呼び出し』

タスクの結果

BUNDLE が使用可能になり、OSGi バンドルが OSGi フレームワークに正常にインストールされ、すべての OSGi サービスがアクティブです。OSGi バンドルは、フレームワーク内の他のバンドルから使用可能です。

次のタスク

OSGi フレームワークの外部にある他の CICS アプリケーションから Java アプリケーションを使用可能にすることができます。それには、1052 ページの『JVM サーバー内の Java アプリケーションの呼び出し』の説明に従ってください。

CICS バンドル内の Java EE アプリケーションの Liberty JVM サーバーへのデプロイ

Liberty JVM サーバーに、CICS バンドルとしてパッケージされている Java EE アプリケーションをデプロイすることができます。

始める前に

WAR ファイル、EAR ファイルまたは EBA ファイルのいずれの形式の Java EE アプリケーションも、zFS に CICS バンドルとしてデプロイする必要があります。ターゲット JVM サーバーが CICS 領域で有効になっている必要があります。

Java アプリケーションの作成および再パッケージ化に関する一般情報については、704 ページの『IBM CICS SDK for Java を使用したアプリケーションの開発』を参照してください。

共通コードのライブラリーを含んでいる OSGi バンドルに依存している場合は、そのバンドルを Liberty バンドル・リポジトリにインストールします。1045 ページの『JVM サーバーにおける OSGi バンドルのデプロイ』を参照してください。

このタスクについて

CICS アプリケーション・モデルでは、Java アプリケーション・コンポーネントが CICS バンドルにパッケージされて zFS にデプロイされます。CICS バンドルをインストールすることによって、アプリケーション・コンポーネントのライフサイクルを管理できます。

Java EE アプリケーションには次のものを含めることができます。

- アプリケーションのプレゼンテーション層およびビジネス・ロジックを提供する 1 つ以上の WAR ファイル

- EBA ファイルにエクスポートされた 1 つの OSGi アプリケーション・プロジェクト (その中にはプレゼンテーション層を提供する 1 つの Web 対応 OSGi バンドル・プロジェクトと、ビジネス・ロジックを提供する追加の OSGi バンドル・セットが含まれる)
- プレゼンテーション層およびビジネス・ロジックを提供する 1 つ以上の WAR ファイルが含まれたエンタープライズ・アプリケーション・アーカイブ (EAR) ファイル

手順

1. zFS 内のバンドルのディレクトリーを指定する BUNDLE リソースを作成します。
 - a. CICS Explorer の CICS SM パースペクティブで、CICS Explorer メニュー・バーの「定義」 > 「バンドル定義」をクリックして、「バンドル定義」ビューを開きます。
 - b. そのビュー内の任意の場所を右クリックし、「**New**」をクリックして「New Bundle Definition」ウィザードを開きます。そのウィザードのフィールドに、BUNDLE リソースの詳細を入力してください。
 - c. BUNDLE リソースをインストールします。以下に示すように、使用可能または使用不可の状態ではリソースをインストールできます。
 - DISABLED 状態でリソースをインストールすると、CICS は Java EE アプリケーションを Liberty サーバーにインストールしようとしません。
 - ENABLED 状態でリソースをインストールすると、CICS は、Java EE アプリケーション (WAR ファイル、EAR ファイル、EBA ファイル) を `${server.output.dir}/installedApps` ディレクトリーにインストールし、`<application>` 項目を `${server.output.dir}/installedApps.xml` に追加します。
2. オプション: BUNDLE リソースがまだ ENABLED 状態でない場合は、そのリソースを有効にして、Liberty サーバーで Java EE アプリケーションを開始します。
3. CICS Explorer メニュー・バーの「**Operations**」 > 「**Bundles**」をクリックして、「Bundles」ビューを開きます。BUNDLE リソースの状態を確認します。
 - BUNDLE リソースが ENABLED 状態である場合、CICS はバンドル内のすべてのリソースを正常にインストールし、バンドルに含まれるすべての Liberty アプリケーションが開始されました。
 - BUNDLE リソースが ENABLING 状態である場合、バンドル内のすべてのリソースを CICS が現在インストール中であるか、バンドルに含まれる 1 つ以上の Liberty アプリケーションがまだインストール中または開始中です。
 - BUNDLE リソースが DISABLED 状態である場合、CICS はバンドル内の 1 つ以上のリソースをインストールできませんでした。この状況は、バンドルに含まれる Liberty アプリケーションが開始に失敗したか、タイムアウト前にアプリケーションが Liberty をインストールしなかった場合に発生する可能性があります。タイムアウトは `JVM システム・プロパティ` `com.ibm.cics.jvmserver.wlp.bundlepart.timeout` によって構成されます。

BUNDLE リソースが ENABLED 状態でインストールできなかった場合、BUNDLE リソースのバンドル・パーツを確認してください。いずれかのバンドル・パーツが UNUSABLE 状態である場合、問題の原因を説明するメッセージが発行されます。例えば、この状態は、zFS で CICS バンドルに問題がある、あるいは関連付けられている JVMSERVER リソースが利用不可であることを示します。その BUNDLE リソースを破棄し、報告された問題を解決してから、BUNDLE リソースを再度インストールする必要があります。

4. オプション: アプリケーション・トランザクションで Java EE アプリケーション要求を実行するために、URIMAP および TRANSACTION リソースを作成できます。アプリケーションに対するセキュリティーを制御する場合は、URI マップの定義が役に立ちます。URI を特定のトランザクションにマップして、トランザクション・セキュリティーを使用できるからです。通常、これらのリソースは CICS バンドルの一部として作成され、アプリケーションによって管理されます。しかし、それらのリソースを別に定義することが望ましい場合は、そうすることもできます。
 - a. PROGRAM 属性を DFHSJTHP に設定するアプリケーション用に TRANSACTION リソースを作成します。この CICS プログラムは、Liberty JVM サーバーに対するインバウンド Java EE 要求のセキュリティー検査を処理します。何らかのリモート属性を設定しても、それらは CICS によって無視されます。トランザクションは常にローカル CICS 領域と接続されている必要があるからです。
 - b. JVMSERVER のタイプが USAGE である URIMAP リソースを作成します。TRANSACTION 属性をアプリケーション・トランザクションの名前に設定し、SCHEME 属性を HTTP または HTTPS に設定します。USERID 属性を使用してユーザー ID を設定することもできます。アプリケーションのセキュリティー認証メカニズムを使用する場合、この値は無視されます。認証が行われず、URI マップにユーザー ID が設定されていない場合、デフォルトの CICS のユーザー ID で処理は実行されます。

タスクの結果

CICS リソースが有効になり、Java EE アプリケーションが Liberty JVM サーバーに正常にインストールされます。

次のタスク

Web クライアントを介して Java アプリケーションの使用可能性をテストすることができます。アプリケーションを更新または削除する場合は、Java アプリケーションの管理を参照してください。

Liberty JVM サーバーへの Java EE アプリケーションの直接デプロイ

Java EE アプリケーションをデプロイするには、server.xml 内の application エレメントを定義するか、アプリケーションを以前に定義した dropins ディレクトリにコピーします。

始める前に

JVM サーバーは、Liberty テクノロジーを使用するように構成する必要があります。

このタスクについて

Java EE アプリケーションは、Web アーカイブ (WAR)、エンタープライズ・バンドル・アーカイブ (EBA)、またはエンタープライズ・アプリケーション・アーカイブ (EAR) としてパッケージ化できます。

Liberty では、以下の 2 つの方法で Java EE アプリケーションをインストールできます。

- `application` エlementを `server.xml` に追加できます。
- あるいは、アプリケーションを Liberty JVM サーバーの `dropins` ディレクトリにコピーすることもできます。 `dropins` を使用すると、CICS は常にトランザクション CJSA の下で実行され、CICS セキュリティーなどの追加のサービス品質による利点は得られません。

注:

- 両方の手法を使用して、同じアプリケーションを同じ JVM サーバーにデプロイしないようにしてください。
- CICS 自動構成によって提供されたデフォルトを受け入れた場合、`dropins` ディレクトリは自動的に作成されません。

手順

- サーバー構成ファイルに追加してアプリケーションをデプロイする場合は、次のようにします。

`server.xml` にアプリケーション・Elementの以下の属性を構成する必要があります。

- `id` - 固有でなければなりません。サーバーによって内部で使用されます。
- `name` - 固有でなければなりません。
- `type` - アプリケーションのタイプを指定します。サポートされるタイプは、WAR、EBA、および EAR です。
- `location` - アプリケーションの場所を指定します。この場所は、絶対パスまたは URL になります。

以下に例を示します。

```
<application
  id="com.ibm.cics.server.examples.wlp.tsq.app"
  name="com.ibm.cics.server.examples.wlp.tsq.app"
  type="eba"
  location="${server.output.dir}/path_to_app"/>
```

- **dropins** ディレクトリを作成して、そこにアプリケーションをデプロイする場合は、次のようにします。
 1. `dropins` を使用可能にするには、以下の例と同様な構成を `server.xml` に追加する必要があります。

```
<applicationMonitor dropins="dropins" dropinsEnabled="true" pollingRate="5s"
updateTrigger="disabled"/>
```

詳しくは、Controlling dynamic updatesを参照してください。

2. FTP を使用して、エクスポートしたファイルをバイナリー・モードで dropins ディレクトリーに転送します。ディレクトリー・パスは WLP_USER_DIR/servers/server_name/dropins です。ここで、server_name は com.ibm.cics.jvmserver.wlp.server.name プロパティーの値です。プロパティーが設定されていない場合、このプロパティーは defaultServer になります。

タスクの結果

Liberty JVM サーバーによってアプリケーションがインストールされます。

次のタスク

Web ブラウザーから Java EE アプリケーションにアクセスして、アプリケーションが正常に実行されていることを確認します。アプリケーション・ファイルを削除するには、dropins ディレクトリーから WAR、EBA または EAR ファイルを削除します。アプリケーション・エレメントと一緒にデプロイされている場合は、そのエレメントを server.xml から削除します。

Liberty JVM サーバーへの共通ライブラリーのデプロイ

DLL ファイル、JAR ファイル、または OSGi バンドルのどちらとして提供されているかに応じて、共通ライブラリーをデプロイします。

手順

- DLL ファイルとして提供される共通ライブラリーの場合は、JVM プロファイルの LIBPATH_SUFFIX オプションによって参照されているディレクトリーにファイルをコピーします。

LIBPATH_PREFIX および LIBPATH_SUFFIX について詳しくは、JVM プロファイルに使用するシンボルを参照してください。

- OSGi バンドルの JAR ファイルとして提供される共通ライブラリーの場合は、server.xml ファイルの bundleRepository 定義で参照されているディレクトリーに JAR ファイルをコピーします。

詳しくは、server.xml の手動調整の『バンドル・リポジトリー』を参照してください。

- JAR ファイルとして提供されるが、OSGi バンドルではない共通ライブラリーの場合は、server.xml ファイルのグローバル・ライブラリー定義で参照されているディレクトリーに JAR ファイルをコピーします。

詳しくは、server.xml の手動調整の『グローバル/共用ライブラリー』を参照してください。

JVM サーバー内の Java アプリケーションの呼び出し

JVM サーバーで稼働している Java アプリケーションを呼び出す方法は多数あります。使用される方式は JVM サーバーの特性によって異なります。

このタスクについて

特定の URL を指定した HTTP 要求を使用することにより、Liberty JVM サーバーで実行される Web アプリケーションを呼び出すことができます。 **EXEC CICS LINK** または **EXEC CICS START** から直接 Web アプリケーションを起動することはできません。Plain Old Java Object (POJO) として実装され、WAR や EAR にパッケージ化された Java EE アプリケーションがある場合は、**EXEC CICS LINK** または **EXEC CICS START** を使用してそれらのアプリケーションのビジネス・ロジック・コンポーネントを呼び出すことができます。

OSGi JVM サーバーで実行されている Java アプリケーションを呼び出す場合は、Java で定義されている PROGRAM に対して **EXEC CICS LINK** を実行するか、ターゲット PROGRAM が Java で定義されている TRANSACTION に対して **EXEC CICS START** を実行できます。PROGRAM 定義では、JVMSEVER と、呼び出す CICS 生成 OSGi サービスの名前を指定します。このようなリンク可能な OSGi サービスは、マニフェストに CICS-MainClass ヘッダーが含まれている OSGi バンドルのインストール時に、CICS によって作成されます。CICS-MainClass ヘッダーには、アプリケーションへのエントリー・ポイントとして機能させる、OSGi バンドル内の Java クラスの main メソッドが指定されています。

OSGi サービスは、OSGi フレームワークに登録されている明確に定義されたインターフェースです。OSGi バンドルやリモート・アプリケーションは OSGi サービスを使用して、OSGi バンドルにパッケージされているアプリケーション・コードを呼び出します。OSGi バンドルは複数の OSGi サービスをエクスポートできます。詳しくは、OSGi JVM サーバーの OSGi バンドルの更新を参照してください。

クラスパス・ベースの JVM サーバーでの Java 関数の呼び出しは、通常、バッチ、Axis2 および SAML などの JVM サーバー固有の機能の一部として実行されます。これらの機能用に、DFHSJJI ベンダー・インターフェースが提供されています。

手順

- Web アーカイブ (WAR) ファイル、エンタープライズ・アーカイブ (EAR) ファイル、または、Web アプリケーション・バンドル (WAB) を含み Liberty JVM サーバーで実行されるエンタープライズ・バンドル・アーカイブ (EBA) ファイルとして開発された Web アプリケーションの場合は、URL を使用してクライアントのブラウザからアプリケーションを呼び出します。Java EE アプリケーションのビジネス・ロジック・コンポーネントの呼び出しについて詳しくは、761 ページの『CICS プログラムで呼び出すための Java EE アプリケーションの準備』を参照してください。
- OSGi JVM サーバーにデプロイされている OSGi バンドルの場合は、以下のステップに従ってください。

1. OSGi フレームワークで使用したい、アクティブな OSGi サービスのシンボル名を判別します。CICS Explorer で「**Operations**」 > 「**Java**」 > 「**OSGi Services**」をクリックして、アクティブな OSGi サービスをリストします。
 2. 他の CICS アプリケーションに対して OSGi サービスを表す PROGRAM リソースを作成します。
 - JVM 属性で、YES を指定して、プログラムが Java プログラムであることを示します。
 - JVMCLASS 属性で、OSGi サービスのシンボル名を指定します。この値は大/小文字の区別があります。
 - JVMSERVER 属性で、OSGi サービスが実行される JVMSERVER リソースの名前を指定します。
 3. 以下の 2 つの方法で Java アプリケーションを呼び出すことができます。
 - トランザクション ID を指定する 3270 または **EXEC CICS START** 要求を使用します。OSGi サービスの PROGRAM リソースを定義する TRANSACTION リソースを作成します。
 - **EXEC CICS LINK** 要求、ECI 呼び出し、または EXCI 呼び出しを使用します。要求をコーディングする際に、OSGi サービスの PROGRAM リソースの名前を指定します。
- Axis2 または SAML 機能については、Axis2 用の JVM サーバーの構成および SAML 用の CICS の構成を参照してください。

タスクの結果

他のコンポーネントが Java アプリケーションを使用できるようにするための定義を作成しました。CICS は、ターゲット JVM サーバーで要求を受け取ると、指定された Java クラスまたは Web アプリケーションを新しい CICS Java スレッドで呼び出します。関連付けられた OSGi サービスまたは Web アプリケーションが登録されていないか、非アクティブである場合、呼び出し側プログラムにエラーが戻されます。

CICS の非 OSGi Java アプリケーションの配置

Java アプリケーションは、CICS バンドルに組み込むことができ、z/OS UNIX System Services (z/OS UNIX) ファイル・システムに CICS Explorer から直接配置できます。エクスポートされたバンドルには、CICS で使われるアプリケーションの JAR ファイルが含まれます。

このタスクについて

このタスクでは、非 OSGi Java アプリケーションを配置する手順の概要を示します。OSGi アプリケーションの場合と同じ手順ですが、唯一の違いは、CICS がバンドルの代わりにアプリケーション JAR ファイルを使用するという点です。z/OS ファイル・システムに直接バンドルを配置する権限がない場合、バンドルを圧縮ファイルとしてエクスポートできます。詳しくは、CICS Explorer 製品資料内の『ローカル・ファイル・システムへの CICS バンドル・プロジェクトのエクスポート』を参照してください。

手順

1. Java アプリケーションをプラグイン・プロジェクトに変換します。 839 ページの『既存の Java プロジェクトのプラグイン・プロジェクトへの変換』の説明に従ってください。
2. プラグイン・プロジェクトを CICS バンドルに追加します。 711 ページの『CICS バンドル・プロジェクトへのプロジェクトの追加』の説明に従ってください。
3. バンドル・プロジェクトを z/OS UNIX ファイル・システムにデプロイします。 CICS Explorer 製品資料内の『CICS バンドルのデプロイ』の説明に従ってください。

タスクの結果

Java アプリケーションが z/OS UNIX にエクスポートされます。エクスポートされたバンドルには、アプリケーションの JAR ファイルが含まれます。

CICS Db2 プログラムの実行および実動の準備

このセクションでは、CICS Db2 環境でのプログラムの準備について説明します。

CICS Db2 環境での Java プログラムのサポートについては、Java プログラムから Db2 データにアクセスするための JDBC および SQLJ の使用を参照してください。

以下のトピックには、診断、変更または調整に関する情報が含まれています。

CICS Db2 テスト環境

CICS Db2 テスト環境をセットアップするときには、いくつかの CICS システムを Db2 システムに接続させる必要があるかを考慮してください。

複数の CICS システムを同じ Db2 システムに接続することができます。ただし CICS Db2 接続機能では、1 つの CICS システムを複数の Db2 システムに同時に接続することはできません。

次のように、実動環境およびテスト環境をセットアップすることが可能です。

- 1 つの CICS システムを 1 つの Db2 システムに接続する
- 実動用とテスト用に複数の CICS システムを同じ Db2 システムに接続する
- 複数の CICS システムを、複数の異なる Db2 システムに接続する

最初の選択肢 (実動とテストに 1 つの CICS システムを使用すること) は推奨されません。テストのアプリケーションが実動システムのパフォーマンスに影響を与える可能性があるためです。

2 番目の選択肢 (ただ 1 つの Db2 システム) をテスト用および実動用に使用することも可能です。これが適しているかどうかは、実際の開発環境と実動環境に応じて異なります。テスト CICS システムと実動 CICS システムを別個に実行することで、テストにおける障害が実動に影響するのを防ぐことができます。

3 番目の選択肢 (例えばテスト用と実動用にそれぞれ 1 つの Db2 システムを使用すること) は最も柔軟です。2 つの CICS サブシステムを 1 つ以上の Db2 システムと共に実行することができます。複数の CICS システムが異なる Db2 システムに接続される場合、

- ユーザー・データと Db2 カタログは共用されません。テスト・データを実動データから分離する必要がある場合には、これが利点となります。
- テスト対象アプリケーションにおける間違った設計やプログラム・エラーは、実動システムのパフォーマンスに影響を与えません。
- 実動データが使用不可であるため、テスト・システムでの権限付与をそれほど厳しくする必要がないかもしれません。2 つの CICS システムが同じ Db2 システムに接続される場合、プログラマーにとって使用可能な機能とデータに関連する権限を厳しく制御する必要があります。

CICS Db2 プログラムの準備

Db2 Interactive (DB2I) インターフェースを使用して CICS Db2 プログラムを準備することも、独自の JCL をバッチ処理のために実行依頼することもできます。

このタスクについて

1056 ページの図 189に示すステップは、アプリケーション・プログラム設計およびコーディングの完了後に、プログラムを実行するための準備方法を要約しています。

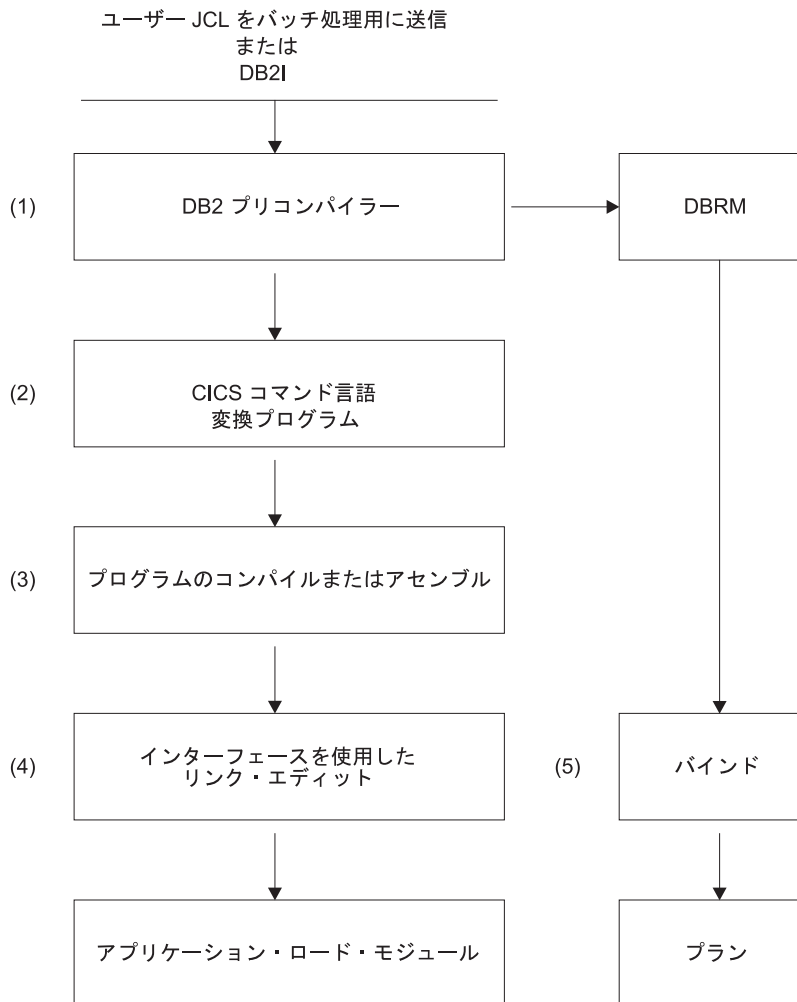


図 189. Db2 にアクセスする CICS アプリケーション・プログラムを準備するステップ

このプロセスの各段階の概要については、Db2 にアクセスする CICS アプリケーション・プログラムの準備を参照してください。

Db2 にアクセスする CICS アプリケーション・プログラムを準備するには、次のようにします。

- Db2 プリコンパイラー (ステップ 1) では、プログラムの各 SQL ステートメントに関する情報を含む DBRM を作成します。また、プログラム内の SQL ステートメントを妥当性検査します。Db2 プリコンパイラーの使用については、「Db2 for z/OS 製品資料内の『Db2 for z/OS のプログラミング』」を参照してください。
- ソース・プログラムが PL/I で作成される場合、ステップ 1 の Db2 プリコンパイラーへの入力、PL/I マクロ局面 (使用された場合) からの出力となります。
- ステップ 1 の Db2 プリコンパイラー、およびステップ 2 の CICS コマンド言語変換プログラムを、どちらの順序でも実行できます。示した順序は、推奨される方式であり、これは、DB2I プログラム準備パネルでサポートされている方式

です。CICS コマンド言語変換プログラムを最初に実行した場合、EXEC SQL ステートメントが検出されるたびに、警告メッセージが生成されますが、これらのメッセージは結果に影響しません。

- CICS 変換プログラムを統合した、言語環境プログラム (Language Environment) に準拠するコンパイラ (COBOL および PL/I) の 1 つを使用する場合、EXEC CICS コマンド (ステップ 2) の変換は、プログラムのコンパイル (ステップ 3) 中に行われます。統合 CICS 変換プログラム、およびそれをサポートするコンパイラの詳細については、変換およびコンパイルを参照してください。
- DB2 バージョン 7 以上を実行しており、言語環境プログラム準拠の COBOL コンパイラまたは PL/I コンパイラのいずれかを使用して、COBOL プログラムまたは PL/I プログラムを準備する場合、コンパイラが SQL ステートメント・コプロセッサ (DBRM を生成する) も提供するので、独立した Db2 プリコンパイラ (ステップ 1) を使用する必要はありません。SQL ステートメント・コプロセッサの使用方法について詳しくは、「Db2 for z/OS 製品資料内の『Db2 for z/OS のプログラミング』」を参照してください。
- DB2 バージョン 6 以前を実行しており、COBOL プログラムまたは PL/I プログラムを準備している場合は、独立した Db2 プリコンパイラを使用します。COBOL プログラムの場合、Db2 プリコンパイラおよび統合 CICS 変換プログラムの場合と同様に、ストリング区切り文字を指定するようにしてください。デフォルトの区切り文字は、互換性がありません。
- プログラムのリンク・エディット (ステップ 4) では、コーディングしている言語に適した CICS EXEC インターフェース・モジュールまたはスタブと、CICS Db2 言語インターフェース・モジュール DSNCLI の両方を組み込みます。CICS EXEC インターフェース・モジュールを最初にロード・モジュールに組み込む必要があります。24 ビット・アドレッシング・モードまたは 31 ビット・アドレッシング・モード (AMODE=31) のいずれかで、DSNCLI をプログラムにリンクできます。アプリケーション・プログラムが 31 ビット・アドレッシング・モードで稼働する場合は、アプリケーションが 16MB より上で稼働できるように、属性 AMODE=31 および RMODE=ANY を指定して DSNCLI スタブをアプリケーションにリンク・エディットする必要があります。
- バインド・プロセス (ステップ 5) では、Db2 が必要です。バインド・プロセスでは、DBRM を使用して、プログラムによる Db2 データへのアクセスを可能にするアプリケーション・プラン (多くの場合、単にプランと呼ばれる) を作成します。バインド・プロセスの詳細については、バインド・プロセスを参照してください。同一のエントリ・スレッド (つまり、同一の DB2ENTRY で指定されている) を使用するトランザクションのグループは、同一のアプリケーション・プランを使用しなければなりません。それらの DBRM は、同一のアプリケーション・プランにバインドするか、または後で同一のアプリケーション・プラン内にリストされるパッケージにバインドする必要があります。

1058 ページの表 92 では、プログラムの言語および Db2 のバージョンに応じて、CICS Db2 プログラムを準備するために必要となるタスクを示しています。

表 92. Db2 にアクセスする CICS プログラムを準備するタスク

| Db2 バージョン
およびプログラ
ム言語 | ステップ 1
(SQL ステ
ートメント
処理) | ステップ 2
(CICS コマン
ド変換) | ステップ 3
(プログラム・
コンパイル) | ステップ 4 (リ
ンク・エディッ
ト) | ステッ
プ 5 (バ
インド) |
|--|--|--|-----------------------------|---|-----------------------|
| DB2 バージョン
6 およびアセン
ブラー | Db2 プリコ
ンパイラー | CICS 提供の
独立変換プロ
グラム | 言語コンパイ
ラー | EXEC インター
フェースおよび
DSNCLI を使用
したリンク・エ
ディット | バイン
ド・プ
ロセス |
| DB2 バージョン
6 および PL/I | Db2 プリコ
ンパイラー | 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 | | EXEC インター
フェースおよび
DSNCLI を使用
したリンク・エ
ディット | バイン
ド・プ
ロセス |
| DB2 バージョン
6 および
COBOL | Db2 プリコ
ンパイラー | 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 | | EXEC インター
フェースおよび
DSNCLI を使用
したリンク・エ
ディット | バイン
ド・プ
ロセス |
| DB2 バージョン
6 およびその他
の言語 | Db2 プリコ
ンパイラー | CICS 提供の
独立変換プロ
グラム | 言語コンパイ
ラー | EXEC インター
フェースおよび
DSNCLI を使用
したリンク・エ
ディット | バイン
ド・プ
ロセス |
| DB2 バージョン
7 (またはそれ以
降) およびアセン
ブラー | Db2 プリコ
ンパイラー | CICS 提供の
独立変換プロ
グラム | 言語コンパイ
ラー | EXEC インター
フェースおよび
DSNCLI を使用
したリンク・エ
ディット | バイン
ド・プ
ロセス |
| DB2 バージョン
7 (またはそれ以
降) および PL/I | 統合 CICS 変換プログラムおよび SQL ス
テートメント・コプロセッサをサポート
する言語コンパイラー | | | EXEC インター
フェースおよび
DSNCLI を使用
したリンク・エ
ディット | バイン
ド・プ
ロセス |
| DB2 バージョン
7 (またはそれ以
降) および
COBOL | 統合 CICS 変換プログラムおよび SQL ス
テートメント・コプロセッサをサポート
する言語コンパイラー | | | EXEC インター
フェースおよび
DSNCLI を使用
したリンク・エ
ディット | バイン
ド・プ
ロセス |
| DB2 バージョン
7 (またはそれ以
降) および他の言
語 | Db2 プリコ
ンパイラー | CICS 提供の
独立変換プロ
グラム | 言語コンパイ
ラー | EXEC インター
フェースおよび
DSNCLI を使用
したリンク・エ
ディット | バイン
ド・プ
ロセス |

CICS Db2 プログラムの準備は、Db2 Interactive (DB2I) インターフェースを使っ
て行うか、または独自の JCL をバッチ実行のために処理依頼して行います。

- Db2 Interactive (DB2I) インターフェース: DB2I は、プリコンパイル、コンパイル、またはアセンブルするためのパネル、アプリケーション・プログラムをリンク・エディットするためのパネル、およびプランをバインドするためのパネルを提供します。アプリケーション・プログラムの準備の詳細については、「Db2 for z/OS 製品資料内の『Db2 for z/OS のプログラミング』」を参照してください。
- バッチ実行のために処理依頼されるユーザー JCL: Db2 ライブラリー SDSNSAMP 内のメンバー DSNTJE5C および DSNTJE5P には、CICS 用の COBOL プログラムおよび PL/I プログラムの準備に必要な JCL のサンプルが含まれています。

CICS の稼働中にプログラムを実行のために準備する場合、プログラムの新しいバージョンを CICS に認識させるために、CEMT NEWCOPY コマンドの実行が必要になることがあります。

CICS SQLCA フォーマット設定ルーチン

IBM 提供の SQLCODE メッセージ・フォーマット設定プロシージャである DSNTIAR を使用すると、アプリケーションに「SQL メッセージをオンラインで」送信できます。

DB2 バージョン 3.1 では、DSNTIAR は 2 つのフロントエンド・モジュール (DSNTIAC と DSNTIAR) および 1 つのランタイム・モジュール (DSNTIA1) に分割されました。DSNTIAC は CICS アプリケーションに使用され、DSNTIAR はその他の Db2 インターフェースに使用されます。DB2 バージョン 3.1 より前では、(リリース変更またはメンテナンス適用によって) DSNTIAR が変更されるたびにアプリケーション・モジュールを再リンク・エディットする必要がありましたが、上記のような変更点により、その必要がなくなりました。以前に DSNTIAR でリンク・エディットされたアプリケーションが存在する場合、代わりに DSNTIAC を使ってそれらを再びリンク・エディットすることを考慮してください。こうするとパフォーマンスが改善され、DSNTIAR の変更からそれらを分離することができます。

CICS フロントエンド部分である DSNTIAC は、Db2 ライブラリー SDSNSAMP でソース・メンバーとして提供されます。

DSNTIAC および DSNTIA1 に必要なプログラム定義は、CSD における IBM 提供グループ DFHDB2 で提供されます。DSNTIA1 をロード可能にするには、SDSNLOAD ライブラリーを CICS DFHRPL 連結に追加する必要があります (CICS ライブラリーの後)。

プログラム変更後に何をバインドするか

プログラムを変更した場合、そのプログラムを使用する前に、それを準備して再バインドする必要があります。

このタスクについて

バインド・プロセスの概要については、バインド・プロセスを参照してください。プランとパッケージの概要については、計画、パッケージ、および動的計画出口を参照してください。

4 つのプログラム・モジュールから成る CICS トランザクションがあるとしします。モジュール 1 がメイン・モジュールです。モジュール 1 がモジュール 2 を呼び出します。モジュール 1 はモジュール 3 も呼び出し、モジュール 3 はモジュール 4 を呼び出します。実際のトランザクションでモジュールの数が多いのは珍しくありません。モジュールの 1 つで少なくとも 1 つの SQL ステートメントが変更されたとすると、次の手順を実行してプログラムを準備し、トランザクションを再度実行可能にする必要があります。

手順

1. Db2 でプログラムをプリコンパイルします。
2. CICS 変換プログラムを使ってプログラムを変換します。
3. ホスト言語ソース・ステートメントをコンパイルします。
4. リンク・エディットします。
5. プログラム C 用の **DBRM** がパッケージの中にバインドされていた場合、新しい **DBRM** を使ってそのパッケージをバインドすると、プログラム C を使用するすべてのアプリケーション・プランは新しいパッケージを自動的に見つめます。
6. プログラム C 用の **DBRM** がいずれかのアプリケーション・プランの中に直接バインドされていた場合、プログラム C 用 **DBRM** を含むすべてのアプリケーション・プランを見つけます。直接バインドされた全プログラム用の **DBRM** を使ってすべてのアプリケーション・プランを再びバインドして、新しいアプリケーション・プランを獲得します。変更されなかったプログラムに関しては、古い **DBRM** を使用します。REBIND サブコマンドを使用できないことに注意してください。REBIND への入力 は **DBRM** ではなく、プランであるためです。

注: 以前にパッケージを使用しなかった場合は、パッケージを使用すると再バインド・プロセスが簡単になります。それぞれ別個の **DBRM** を 1 つのパッケージとしてバインドし、パッケージ・リストにそれらを含めることができます。パッケージ・リストを **PLAN** の中に含めることができます。その後、(**BIND PLAN** コマンドを使ってアプリケーション・プラン全体をバインドする代わりに) **BIND PACKAGE** コマンドを使用して、変更済みプログラム用の **DBRM** をバインドできます。これによりトランザクションの可用性が高まり、パフォーマンスが改善されます。パッケージの使用方法について、詳しくは Db2 パッケージの使用を参照してください。

プログラムのバインド・オプションと考慮事項

複数のプログラムを 1 つのアプリケーション・プランにバインドするとき、Db2 でタイム・スタンプが使用される方法に注意してください。

各プログラムに関して、Db2 プリコンパイラーは以下のものを作成します。

- Db2 プリコンパイラーは、Tdx タイム・スタンプを持つ **DBRM** を作成します。例えば、最初のプログラムには Td1、2 番目のプログラムには Td2 となります。
- Db2 プリコンパイラーは、SQL パラメーター・リスト内で Tsx タイム・スタンプを持つ変更されたソース・プログラムを作成します。例えば 2 つのプログラムが処理に含まれる場合は Ts1 および Ts2 です。

バインド時に、各プログラム用の DBRM は、指定したパッケージまたはプランの中にバインドされます。さらに、Db2 のカタログ・テーブル SYSIBM.SYSDBRM が更新され、DBRM ごとに 1 行およびそれぞれのタイム・スタンプが含まれるようになります。実行時に Db2 は各 SQL ステートメントのタイム・スタンプを検査して、DBRM のタイム・スタンプとソース・プログラムで設定されたタイム・スタンプが異なる場合には -818 SQL コードを返します (この例では Td1 と Ts1 が異なるか、Td2 と Ts2 が異なる場合)。-818 SQL コードを回避するには、以下のいずれかの方法を使用してください。

- すべてのプログラムをパッケージにバインドして、アプリケーション・プランでこれらのパッケージをリストします。1 つのプログラムが変更された場合、そのプログラムをプリコンパイル、コンパイル、およびリンク・エディットして、それを 1 つのパッケージに再びバインドします。
- いずれかのプログラムをアプリケーション・プランに直接バインドする場合は、それぞれの新しいプログラムまたは変更済みプログラムを必ずプリコンパイル、コンパイル、およびリンク・エディットした後、そのプログラムを含んでいるすべてのアプリケーション・プランをバインドします。その際、これらのプランに直接バインドされるすべてのプログラムからの DBRM を使用します。これを行うには REBIND コマンドではなく、BIND コマンドを使用してください。

プランをバインドするときには多数のオプションを使用できます。ほとんどすべてのバインド・オプションはアプリケーションに依存しているため、アプリケーション設計時にそれを考慮してください。異なるプランに対して異なる BIND オプションを扱うようプロシージャーを開発する必要があります。さらに、プロシージャーは、時間の経過に伴う同じプランの BIND オプションの変化に対処できる必要があります。

以下のセクションは、CICS での BIND オプションに関する具体的な推奨事項をいくつか示しています。

RETAIN

RETAIN は、古いプランからの BIND および EXECUTE 権限が変更されないことを意味します。

RETAIN オプションが使用されない場合、それより前の GRANT による権限はすべて REVOKED (取り消し) になります。BIND コマンドを実行しているユーザーがプランの作成者になります。新しい GRANT コマンドによってすべての権限を再び確立する必要があります。

このような理由で、CICS 環境でプランをバインドするときには RETAIN オプションを使用することをお勧めします。

分離レベル

完成したプランに関して、分離レベルが指定されます。反復可能読み取り (RR) を使用する具体的な必要が生じない限り、カーソル固定 (CS) を使用することをお勧めします。CS を使用することで、ハイレベルな並行性が可能になり、デッドロックのリスクが軽減されます。

分離レベルは、完成したプランに関して指定されることに注意してください。つまり、CICS での特定のモジュール用に RR が必要な場合、プランに含まれるすべての DBRM も RR を使用する必要があります。

さらに、パフォーマンス上の理由で、同じ DB2ENTRY を使う目的で使用頻度の低い多数のトランザクションを一緒にグループ化し、共通のプランを使用させるようにした場合、ただ 1 つのトランザクションだけが RR を必要とする場合でも、この新しいプランもまた RR を使用する必要があります。

プラン検証時

プランは VALIDATE(RUN) または VALIDATE(BIND) でバインドされます。バインドできない SQL ステートメントの処理方法を決定するには VALIDATE(RUN) を使用します。

あるステートメントを実行時にバインドする必要がある場合、それぞれの実行ごとにそれが再バインドされます。つまり、新しい作業単位 (UOW) ごとにステートメントが再バインドされます。

実行時にステートメントをバインドすると、パフォーマンスが影響を受ける可能性があります。実行時にバインドされるステートメントは、実行ごとに再バインドされます。つまり、それぞれの同期点の後でステートメントを再バインドする必要があります。CICS でこのオプションを使用することは推奨されません。

動的 SQL を使用するときには VALIDATE(RUN) が必要でないことに注意してください。それでも動的 SQL は、ステートメントが実行時にバインドされることを暗黙に意味します。

CICS Db2 環境では VALIDATE(BIND) を使用すべきです。

ACQUIRE および RELEASE

ACQUIRE および RELEASE パラメーターはトランザクション比率に関連しているため、時間の経過と共にプランによって変化します。

これらのパラメーターに関する一般的な推奨事項については、最適なパフォーマンスのための BIND オプションの選択で説明されています。

CICS Db2 プログラムのテストおよびデバッグ

Db2 にアクセスする CICS アプリケーション・プログラムをテストおよびデバッグするには、CICS 環境で通常使われるツールを使用することができます。これには実行診断機能 (EDF)、CICS 補助トレース、トランザクション・ダンプが含まれます。

これらの問題判別プロセス、および他の問題判別プロセスについては、Db2 のトラブルシューティングを参照してください。

実動への移行: CICS Db2 アプリケーションのチェックリスト

このチェックリストは、設計、開発、テストが終わった後のアプリケーションを実動に移すために実行する必要があるタスクを示しています。

このタスクについて

これらのタスクは、テスト・システムで使われた標準にかなり依存します。例えば以下のような場合には、実行すべきタスクが異なります。

- テストと実動で別個の Db2 システムが存在する
- テストと実動で 1 つの Db2 だけが使用される。

以下の説明では、テストと実動で別個の Db2 および CICS サブシステムを使用することを想定します。

実動への移行は、以下のアクティビティーを実行することを意味します。

DDL を使って実動データベースを準備する

テスト・システムからの DDL ステートメントを基礎として使用し、実動 Db2 システムに対してすべての DDL 操作が実行される必要があります。いくつかの変更が必要になる可能性があります。例えば、1 次および 2 次割り振りを増やすこと、他のボリューム通し番号の定義、CREATE STOGROUP ステートメントでの新しい VCAT の定義などです。

DCLGEN の準備

COBOL および PL/I プログラムの場合、テスト Db2 システムからの DCLGEN 入力を使用して、実動 Db2 システムに対する DCLGEN 操作を実行する必要が生じることがあります。

コンパイルに関するオプションによっては (実動システムでコンパイルが実行されない場合)、代わりの方法として、テスト・ライブラリーから実動ライブラリーに DCLGEN 出力構造をコピーすることができます。これにより、テスト・システムと実動システムの間ですべての情報が分離されたままになります。

実動システムのためのプリコンパイル

テスト・システムで既にプログラムをパッケージにバインドした場合は、このステップを実行する必要はありません。パッケージを実動システムに直接移すことができます。その方法について、詳しくは『実動システムのためのアプリケーション・プランの生成』を参照してください。ただし、プログラムをアプリケーション・プランに直接バインドするか、実動システムでプログラムをパッケージにバインドするためには、実動システムにプログラム用の DBRM を配置する必要があります。以下のいずれかを行うことができます。

- 実動システムで、EXEC SQL ステートメントを含む CICS モジュールをプリコンパイルする。または
- DBRM をテスト・システムから実動システム・ライブラリーにコピーする。

実動システムのためのコンパイルおよびリンク・エディット

ロード・モジュールを生成するには、次のようにします。

- 実動システムでのプリコンパイルによって DBRM が生成された場合は、実動システムで CICS モジュールをコンパイルしてリンク・エディットします。または、

- DBRM がコピーされた場合、またはテスト・システムから実動システムにパッケージを移動しようとしている場合には、ロード・モジュールをテスト・システムから実動システム・ライブラリーにコピーします。

表 93 に示されている手順に従って、変更後のロード・モジュールをテスト・システムから実動システム・ライブラリーにコピーし、古いバージョンのロード・モジュールを置換することができます。

表 93. 変更されたプログラムをテスト環境から実稼働環境に移動する

| テスト・システム | 実動システム | 注 |
|-------------------------|-----------------------------|---------------------------------|
| | USER.PROD.LOADLIB(PGM3) | 元のロード・モジュール |
| USER.TEST.LOADLIB(PGM3) | | テスト・ロード・モジュール |
| | USER.OLD.PROD.LOADLIB(PGM3) | 古いバージョンのプログラムは別の実動ライブラリーに配置されます |
| | USER.PROD.LOADLIB(PGM3) | 新しいバージョンのプログラムが実動ライブラリーに配置されます |

適切な JCL を使って実稼働ライブラリーを選択することにより、古いバージョンまたは新しいバージョンのプログラムを実行できます。これにより、プログラム・ロード・モジュールに組み込まれた整合性トークンによって判別される正しいバージョンのパッケージが実行されます。

実動システムのためのアプリケーション・プランの生成

テスト・システムで既にプログラムをパッケージにバインドした場合は、パッケージを実動システムにコピーして、アプリケーション・プランにリストされる集合にそれらを含めることができます。パッケージを実動システムにコピーするとき、アプリケーション・プランに既にリストされている集合にそのパッケージが含まれている限り、実動システムでアプリケーション・プランを再びバインドする必要はありません。

表 94 に示されている手順に従って、変更後のパッケージをテスト・システムから実動システム・ライブラリーにコピーし、古いバージョンのパッケージを置換することができます。この例では、異なるバージョンのパッケージを識別するためにプリコンパイル時に VERSION キーワードを使用しています。VERSION キーワードに関する説明と使用方法について詳しくは、Db2 for z/OS 製品資料内の『Db2 for z/OS のプログラミング』を参照してください。

表 94. 変更されたパッケージをテスト環境から実稼働環境に移動する

| テスト・システム | 実動システム | 注 |
|------------------------------------|---------------------------------------|---|
| | location_name.
PROD_COLL.PRG3.VER1 | 古いバージョンのパッケージ |
| location_name. TEST_COLL.PRG3.VER2 | | 新しいバージョンのパッケージがテスト・システムでバインドされた後、実動システムにコピーされます |
| | location_name.
PROD_COLL.PRG3.VER1 | 古いバージョンが引き続き実動コレクションの中に残ります |
| | location_name.
PROD_COLL.PRG3.VER2 | 新しいバージョンが実動コレクションに配置されます |

プログラムをアプリケーション・プランに直接バインドする場合、または実動システムでプログラムをパッケージにバインドする場合には、実動システムに配置された DBRM に対してバインド・プロセスを実行する必要があります。プログラムをアプリケーション・プランに直接バインドする場合には、それらのプログラムを含んでいる (実動システム上の) すべてのアプリケーション・プランをバインドする必要があります。バインド・プロセスについて詳しくは、バインド・プロセスを参照してください。なお、テーブルや索引のサイズなどさまざまな要因のために、テスト・システムと実動システムの間で EXPLAIN 出力を比較しても無意味である可能性があります。それでも、Db2 最適化プログラムの決定を検査するために、実動システムで最初にプランをバインドする時点で EXPLAIN を実行することをお勧めします。

GRANT EXECUTE

実動システムで Db2 アプリケーション・プランに対する EXECUTE 権限をユーザーに付与する必要があります。

テスト

この時点で、これ以上のテストは必要ありませんが、リソース競合、デッドロック、およびタイムアウトの発生を最小限に抑え、期待されるトランザクション応答時間が得られることを確認するうえで、ストレス・テストが役立つので、それを行うことが推奨されます。

CICS 定義

新しいアプリケーション・プログラムの実行準備ができた状態にするには、CICS 実動システムで以下の RDO 定義を更新します。

- 新しいトランザクション・コード用の RDO トランザクション定義
- 新しいアプリケーション・プログラムおよびマップに関する RDO プログラム定義
- 特定の Db2 要件に関する SIT (それが実動に移される最初の Db2 向けアプリケーションである場合)
- アプリケーションに関する RDO DB2ENTRY および DB2TRAN 定義。RDO DB2CONN 定義 (それが実動に移される最初の Db2 向けアプリケーションである場合)。DB2ENTRY で新しいトランザクションおよびアプリケーション・プランを定義するとき、無保護スレッドを使用して、最初に詳細なアカウント情報とパフォーマンス情報を得ることができます。その後、必要に応じて、保護されたスレッドを使用できます。

加えて、RACF がインストールされている場合、新しいユーザーと Db2 オブジェクトを定義する必要があります。

Db2 にアクセスする CICS アプリケーションのチューニング

Db2 にアクセスする CICS アプリケーションを実動に移す前に調整する必要があります。実動に移した後も定期的に調整する必要があります。

このタスクについて

Db2 にアクセスする CICS アプリケーションを実動に移すときには、CICS に対して既に行われている検査のほかに、以下の検査を追加してください。

- Db2 要求を出すすべてのアプリケーション・プログラムがスレッド・セーフであることを確認します。そうである場合、オープン・トランザクション環境 (OTE) を活用して、アプリケーションのパフォーマンスが改善されます。オープン・トランザクション環境でアプリケーション・プログラムが作動する方法については、スレッド・セーフ・プログラミングにより CICS Db2 アプリケーションが OTE を使用できるようにするの説明を参照してください。
- 使用する SQL ステートメントの数とタイプが、プログラム仕様に合致していることを確認します (Db2 アカウンティング機能を使用)。
- バッファ・プール内で取得および更新されたページの数、予期した数よりも多いかどうかを確認します (Db2 アカウンティング機能を使用)。
- 計画された索引が使用されていることを確認し (EXPLAIN を使用)、非効率的な SQL ステートメントが使われていないことを確認します。
- DDL が使用されているかどうかを確認し、使用されている場合はその理由を確認します (Db2 アカウンティング機能を使用)。
- 会話型トランザクションが使用されているかどうかを確認します。

疑似会話型トランザクションを代わりに使用できるかどうかを判別します。会話型の設計が必要な場合は、複数の会話にわたってロックされる Db2 オブジェクトを調べます。また、この会話型設計のために必要とされる新しいスレッドの数が許容範囲内であることも確認します。

- 使用されるロックとそれらの継続期間を確認します。

例えば以下の項目の指定が間違っている (または最適なものではない) ことが原因で、表スペース・ロックが使用されていないかどうか確認します。

- LOCK TABLE ステートメント
- LOCKSIZE=TS の指定
- ISOLATION LEVEL(RR) の指定
- ロック・エスカレーション

この情報はカタログ表で入手可能です。ただしロック・エスカレーションは例外で、これはインストール・パラメーター (DSNZPARM) です。

- 使用されるプランとそれらのサイズを確認します。アプリケーション・プランがセグメント化されるとしても、プラン内でより多くの DBRM が使用されるほど、プランの BIND および REBIND (変更の場合) に要する時間が長くなります。可能なときには常に、パッケージの使用を試みてください。パッケージは以下のような問題を解決するために設計されました。
 - SQL アプリケーションを変更した後に、プラン全体を再びバインドする。
(これは動的プラン選択によって対処可能でしたが、パフォーマンスへの影響がありました。)
 - 変更された SQL アプリケーションが多数のアプリケーションによって使用される場合、各アプリケーション・プランをバインドする。

この調整が完了したら、想定されるトランザクション負荷を使用して、必要な DB2ENTRY 定義と、必要なスレッド数を決めてください。また、これらのトランザクションが Db2 および CICS サブシステムに与える影響も確認してください。

実動時に Db2 にアクセスする CICS アプリケーションを調整するには、

- バッファ・プールでの GET PAGES の数をモニターすることにより、計画された索引が CICS アプリケーションで使われることを確認します (Db2 アカウンティング機能を使用)。使用されていない索引がある場合、その理由はおそらく、索引が既にドロップされたか、プランのバインド後に索引が作成されたためです。
- アカウンティング機能からのロック・マネージャー・データを使用して、サスペンション、デッドロック、およびタイムアウトを検査します。

CICS Build Toolkit による CICS アプリケーション・ビルド自動化

CICS TS ビルド・ツールキット (CICS Build Toolkit) には、CICS バンドル、CICS アプリケーション、CICS アプリケーション・バインディング、および CICS プラットフォームを含む、CICS プロジェクトのビルド自動化のためのコマンド行インターフェースが用意されています。

CICS Build Toolkit は、OSGi アプリケーション・プロジェクト、OSGi バンドル・プロジェクト、エンタープライズ・アプリケーション・プロジェクト、動的 Web プロジェクト など、CICS バンドルによって参照されるプロジェクトもサポートし、事前ビルドされた OSGi バンドルおよび Liberty アプリケーションを入力データにすることができます。

ビルド自動化

継続的統合環境では、開発者がアプリケーションの更新をチェックインすると、ビルド・スクリプトが自動的に実行されます。スクリプトはソース管理から最新アプリケーションをチェックアウトし、CICS Build Toolkit を呼び出して、アプリケーションを形成するプロジェクトをビルドします。スクリプトはエラーが存在するかどうかビルドの結果を検査して、適切であれば、ビルドされたプロジェクトを成果物リポジトリや zFS 上のステージング・エリアなどの適切な場所にコピーします。

変数置換

同じアプリケーションを異なる複数の環境 (例えば、開発、品質保証、実動) にデプロイする操作を円滑に進めるには、CICS Build Toolkit を使用して、CICS バンドル内の変数を解決できます。スクリプトは通常、ビルドされたプロジェクトを、ターゲット環境にある変数の値を定義したプロパティ・ファイルとともに使用します。

サポートされるオペレーティング・システム

CICS Build Toolkit は、z/OS、Linux、および Microsoft Windows オペレーティング・システムでサポートされています。

CICS Build Toolkit の使用の準備

CICS Build Toolkit をインストール、アップグレード、アンインストールする方法。CICS Build Toolkit を実行するための Java 要件と Eclipse ワークスペースの考慮事項についても説明します。

CICS Build Toolkit のインストール

1. CICS Transaction Server for z/OS 製品情報 Web サイトから最新バージョンの *cicsbt-#####.zip* ファイルをダウンロードします。
2. CICS Build Toolkit を実行するシステムに、*cicsbt-#####.zip* ファイルをバイナリー形式で転送します。
3. *cicsbt-#####.zip* ファイルを解凍して、*cicsbt* ディレクトリを作成します。コマンド **unzip cicsbt-#####.zip** を発行するか、**unzip** が使用できない場合は、コマンド **jar -xf cicsbt-#####.zip** を発行します。
4. CICS Build Toolkit に対するアクセス権限を必要とするすべてのユーザーが、*cicsbt* パス内のすべてのディレクトリおよびサブディレクトリに対する読み取り権限を持っていることと、シェル・スクリプト *cicsbt*、*cicsbt.bat*、または *cicsbt_zos* に対する実行権限を持っていることを確認します。例えば、z/OS では以下のようにします。

```
chmod -R 755 cicsbt_install_dir
```

CICS Build Toolkit のアップグレード

CICS Build Toolkit を新しいバージョンにアップグレードするには、*cicsbt* ディレクトリを削除し、新しいバージョンの README ファイルに記載されたインストール指示に従います。

CICS Build Toolkit のアンインストール

CICS Build Toolkit をアンインストールするには、*cicsbt* ディレクトリを削除します。

CICS Build Toolkit の実行

CICS Build Toolkit には、32 ビット (z/OS では 31 ビット) または 64 ビットの Java 8 互換の Software Development Kit (SDK) が必要です。

SDK ディレクトリを指すように環境変数 *JAVA_HOME* を設定する必要があります。また、ユーザー・ホーム・ディレクトリを指すように環境変数 *HOME*(z/OS および UNIX の場合) または環境変数 *USERPROFILE* (Windows の場合) を設定する必要があります。

使用可能なコマンド行オプションの概要を表示する場合、Windows では *cicsbt.bat* を実行し、Linux では *cicsbt* を実行し、z/OS では *cicsbt_zos* を実行します。

Eclipse ワークスペースの管理

CICS Build Toolkit を実行すると、ユーザー・ホーム・ディレクトリに一時的な Eclipse ワークスペースが作成され、指定された CICS バンドルまたはアプリケーションがそこにビルドされます。CICS Build Toolkit によって作成されたワーク

スペースは、使用中は削除しないでください。そのような操作を行うと、ビルド・プロセスが失敗し、ビルドされた成果物が壊れてしまいます。

1 つのユーザー・アカウントで CICS Build Toolkit の複数のインスタンスを同時に実行する場合は、**--workspace** オプションを使用してワークスペース・ディレクトリーを指定できます。

注: 使用後にワークスペース・ディレクトリーを削除したこと、および各ワークスペースに別個のソース・ディレクトリーを使用していることを確認してください。CICS Build Toolkit はビルド・プロセス中にソース・ディレクトリーに書き込むために、このことは重要です。

CICS バンドル、アプリケーション、アプリケーション・バインディング、またはプラットフォームのビルド

CICS バンドル・プロジェクト、アプリケーション・プロジェクト、アプリケーション・バインディング・プロジェクト、およびプラットフォーム・プロジェクトなどの CICS プロジェクトのビルドは、CICS Build Toolkit を呼び出すビルド・スクリプトを作成することによって自動化できます。

始める前に

CICS Build Toolkit にはソース・プロジェクトに対する書き込み権限が必要です。ビルド・プロセスの一環としてソース・プロジェクトを変更する可能性があるからです。ビルド完了後に破棄される可能性のあるソース・プロジェクトについては、一時コピーを使用するようにしてください。準備について詳しくは、1068 ページの『CICS Build Toolkit の使用の準備』を参照してください。

Java プロジェクトをソースからビルドしている場合、ターゲットとなる CICS リリースを識別して、追加の Java ライブラリー依存関係があればそれらを利用できるようにする必要があります。別の方法として、それらの Java プロジェクトのビルドされたバージョン (例: *.jar*、*.wab*、*.eba*、*.ear* ファイルなど) を CICS バンドルのルート・ディレクトリーにコピーしてから、CICS Build Toolkit を実行することもできます。

このタスクについて

アプリケーションを作成した後、継続的デリバリー・モデルの一環としてアプリケーションをデプロイする前に、以下の手順を行います。ビルド自動化の一環として *cicsbt* を実行します。あるいは、シェルで *cicsbt* を実行するか (UNIX または Linux の場合)、コマンド・プロンプトで *cicsbt.bat* を実行します (Windows の場合)。

手順

1. **--input** オプションを指定して、CICS プロジェクトの場所、およびビルドする参照プロジェクトを指定します。
2. **--build** オプションを指定して、ソース・ロケーションからビルドする CICS バンドル・プロジェクト、アプリケーション・プロジェクト、アプリケーション・バインディング・プロジェクト、およびプラットフォーム・プロジェクトを指定します。

アプリケーション・バイnding・プロジェクトをビルドするとき、CICS Build Toolkit は、関連付けられたアプリケーションおよびプラットフォームを入力ディレクトリーで探します。関連付けられたアプリケーションが見つかった場合、それもビルドされます。関連付けられたアプリケーションまたはプラットフォームが見つからなかった場合、ビルドは警告が表示されて完了します。

3. **--output** オプションを指定して、出力ディレクトリーを示します。
4. オプション: Java プロジェクトを参照する CICS プロジェクトをビルドするには、**--target** オプションを使用して、ターゲット・プラットフォームを指定します。Java プロジェクトが既にビルドされており、CICS バンドルのルート・ディレクトリーに含まれている場合、Java プロジェクトをビルドする必要はありません。

ターゲット・プラットフォームを指定するには、次のようにいくつかの方法があります。

- a. アプリケーションが実行される CICS の最も低いバージョンに対応する、組み込みターゲットを指定します。引数を指定せずに **--target** を指定すると、使用可能な組み込みターゲットのリストを表示できます。
 - b. CICS Explorer または Eclipse によって作成された Eclipse **.target** ファイルのパスを指定します。CICS Build Toolkit を実行するユーザーが **.target** ファイルによって参照されるすべてのライブラリーとディレクトリーにアクセスできるようにします。
5. オプション: ビルド成果物に代替文字セットを使用するには、**--encoding** オプションを指定します。

いくつかの CICS バンドル成果物は、**LOCALCCSID** CICS SIT パラメーターで指定された文字セットになければなりません。CICS Build Toolkit では、デフォルトで文字セット **cp037** が使用されます。

タスクの結果

CICS プロジェクトおよび参照先プロジェクトは、出力ディレクトリーのサブディレクトリー内にビルドされます。そのディレクトリーは、シンボル名_バージョン という命名規則に従います (例えば、**com.ibm.cics.test.bundle_1.0.0** のようになります)。

例

この例は、アプリケーションとバンドルのみがビルドされ、バイndingが関連付けられていない場合の出力ディレクトリーを示しています。

```
/output/dir/  
  applications/  
    my.application_1.0.0  
    my.other.application_1.0.0  
  bundles/  
    my.application.bundle_1.0.0
```

この例は、プラットフォーム **testplatform** に関連付けられたバイndingがビルドされた場合の出力ディレクトリーを示しています。アプリケーション **my.application_1.0.0** とバンドル **my.application.bundle_1.0.0** もビルドされますが、以下のバイndingには関連付けられていません。

```

/output/dir/
  applications/
    my.application_1.0.0
  bundles/
    my.application.bundle_1.0.0
  testplatform/
    applications/
      my.other.application_1.0.0
    bindings/
      my.other.application.test.binding_1.0.0
    bundles/
      my.other.application.bundle_1.0.0
      my.other.application.another.bundle_1.0.0

```

CICS Build Toolkit によって作成される出力ディレクトリーの構造は、プラットフォーム・ホーム・ディレクトリー (PLATHOME) と同じです。プラットフォーム・ホーム・ディレクトリーの構造について詳しくは、z/OS UNIX のプラットフォーム・ディレクトリーの構造を参照してください。

この例は、CICS アプリケーション・バインディング `my.application.binding` (1.0.1) と入力ディレクトリーにある関連アプリケーションをビルドする CICS Build Toolkit 呼び出しを示しています。

```

cicsbt --input my/source/dir/*                                ¥
       --build my.application.binding(1.0.1)                 ¥
       --output /my/output/dir

```

このサンプルを Windows で使用するには、CICS Build Toolkit 呼び出しと同じ行にすべてのパラメーターを入力します。スペースを含んだパスは引用符で囲む必要があります。

次のタスク

ビルドされたプロジェクトに変数が含まれている場合は、CICS にインストールする前にその変数を解決する必要があります。変数置換について詳しくは、『CICS バンドル内の変数の解決』を参照してください。

CICS Build Toolkit を実行した後、ビルド・スクリプトを使用して、ビルドされた CICS プロジェクトを成果物リポジトリまたは zFS のステージング・ロケーションにコピーし、デプロイメント可能な状態にします。

注: 成果物を FTP で zFS に転送する場合は、内容が保持されるようにバイナリー・モードで転送する必要があります。

CICS バンドル内の変数の解決

ビルドされた成果物に未解決の変数が含まれている場合、そうした成果物をターゲット環境にデプロイする前に、CICS Build Toolkit を使用してそれらの変数を解決します。

始める前に

ビルドされた成果物に含まれる変数を解決しようとする前に、CICS Build Toolkit の以前の実行でターゲット・プロジェクトをビルドしたこと確認します。ビルド・プロセスについて詳しくは、1069 ページの『CICS バンドル、アプリケーション、

アプリケーション・バインディング、またはプラットフォームのビルド』を参照してください。

このタスクについて

アプリケーションをビルドした後、継続的デリバリー・モデルのデプロイ・フェーズの一環として、以下の手順を行います。自動デプロイメント・プロセスの一部として、`cicsbt` を実行します。あるいは、シェル (UNIX または Linux) で `cicsbt` を実行するか、コマンド・プロンプト (Windows) で `cicsbt.bat` を実行します。

手順

1. **--resolve** オプションを指定して、解決する CICS プロジェクトの場所を指定します。

注: **--resolve** オプションで指定するパスは、最初にプロジェクトをビルドしたときに **--output** オプションで指定した最上位ディレクトリーと一致している必要があります。

2. オプション: スタンドアロン・バンドル内の変数を解決する場合は、**--properties** オプションを指定して、スタンドアロンの `variables.properties` ファイルの場所を指定します。

プラットフォームまたはアプリケーション・バインディングに関連付けられたバンドル内の変数を解決する場合は、このステップは必要ありません。変数は、プラットフォームまたはアプリケーション・バインディングのルート・フォルダー内の `variables.properties` ファイルを使用して解決されます。

3. オプション: 成果物のビルド時に **--encoding** オプションを指定した場合、CICS Build Toolkit が成果物を正しく読み取れるように、解決時と同じ **--encoding** 値を指定してください。
4. **--output** オプションを指定して、出力ディレクトリーを示します。解決済み出力のディレクトリー構造は、**--resolve** オプションで指定された入力のディレクトリー構造と一致します。

タスクの結果

CICS プロジェクト内の変数が完全に解決され、解決されたプロジェクトは指定された出力ディレクトリーに置かれます。

同ースクリプトでのビルドと解決

ほとんどの場合、成果物のビルドと変数の解決は配信サイクルの異なる時点で行われます。比較的単純なアプリケーションの場合、同ースクリプトでビルドと解決を行うことができます。

次の例では、CICS Build Toolkit を呼び出して、2 つの異なる入力ディレクトリーにある、ID が `OSGiBundleProject` と `AnotherBundleProject` の最新バージョンの CICS バンドルをビルドします。この呼び出しは、CICS Transaction Server バージョン 5.1 ターゲットを使用して、参照先 Java プロジェクトをビルドします。次いで、2 番目の呼び出しでターゲット・バンドル内の変数を解決します。

このサンプルを Windows で使用するには、CICS Build Toolkit 呼び出しと同じ行にすべてのオプションを入力します。パスにスペースが含まれる場合は、必ずパスを引用符で囲んでください。

デプロイメント処理の次のステップに進みます。例えば、解決されたプロジェクトを CICS プラットフォームのホーム・ディレクトリに移動するスクリプトを作成し、DFHDPLOY ユーティリティを使用してバンドル・リソースやアプリケーション・リソースを CICS にデプロイします。

CICS Build Toolkit コマンド行オプションと、CICS バンドルのビルドの例、およびビルドされた成果物に含まれる変数の解決の例を示します。

```
>>-+build project-name-or-path--input path--+-----+--+>
|                                     '-target .target-file-or-target-ID-' |
+-resolve path--+-----+-----+-----+-----+
|               '-properties path-' |
|'-help-----|
|                                     '.encoding cp037-----.'
>>-output path--+-----+-----+-----+-----+-----+
|               '-collectDiagnostics-' '-encoding character-set-'
>>-+-----+-----+-----+-----+-----+-----+-----+
|               '-verbose-'           '-workspace path-'
```

--input オプションで指定したディレクトリーからビルドする CICS バンドル・プロジェクト、アプリケーション・プロジェクト、アプリケーション・バイnding・プロジェクト、およびプラットフォーム・プロジェクトを指定します。

プロジェクトは、シンボル名とバージョン (例: MyBundleProject(1.0.1))、またはプロジェクトのパス (例: /u/user/applications/testapplication) のいずれでも指定できます。シンボル名を指定してバージョンを指定しない場合、CICS Build Toolkit は使用できる最も高いバージョンのプロジェクトをビルドします。単一の呼び出しで複数のプロジェクトをビルドするには、**--build** に複数の引数をスペースで区切って指定します。あるいは、引数なしで **--build** を指定すると、**--input** で指定されたパスにあるすべてのプロジェクトがビルドされます。

--build オプションと --resolve オプションを同時に使用することはできません。

--collectDiagnostics、-c

ビルドが完了した後で、診断データの収集を要求します。

診断データは *.zip* ファイルに保管されます。このデータは、ビルドの開始時に使用されていた作業ディレクトリーに書き込まれます。作業ディレクトリーが読み取り専用である場合は、システムの一時ディレクトリーに書き込まれます。この診断データを IBM に送信して、問題の診断に役立てることもできます。

別の方法として、環境変数 *JAVA_TOOL_OPTIONS* を *-Dcollect.diagnostic.data* に設定して、診断データの収集を要求することもできます。

--encoding、-e character-set

バンドルをインストールする CICS 領域のデフォルト CCSID を表す IANA 文字セット名。このオプションを指定しない場合、IANA 文字セット *cp037* が使用されます。このエンコードは、EBCDIC での表示に必要な Atom 構成ファイル、JVM プロファイル、および Node.js プロファイルを変換するために使用されます。

例えば、CICS 領域の **LOCALCCSID** システム初期設定パラメーターが *00285* の場合、**--encoding ibm285** を指定します。CCSID および対応する文字セット名のリストについては、CICS がサポートされている変換を参照してください。

--help、-h

選択可能なオプションのリストを出力します。

--input、-i path

ビルドするプロジェクトのパスを指定します。

スペース区切り文字を使用して複数のパスを指定できます。パスの末尾にアスタリスクを使用してそのディレクトリー内のすべてのプロジェクトを指定することで (例: */path/to/top/level/directory/**)、単一の呼び出しで複数のプロジェクトをビルドすることもできます。

--output、-o path

ビルドまたは解決されたプロジェクトを配置するパスを指定します。

CICS Build Toolkit は、シンボル名およびバージョンに基づいてビルドされたアプリケーションおよびバンドルのためのサブディレクトリーを作成します。アプリケーションまたはバンドルのみがビルドされる場合、それらは */applications* および */bundles* サブディレクトリーに配置されます。1 つ以上のアプリケーション・バインドイングがビルドされる場合、関連付けられたすべての成果物は、プラットフォーム名を含んだディレクトリー構造に配置されます。バインドイングに関連付けられていないアプリケーションまたはバンドルは、通常どおりサブディレクトリーに配置されます。

--properties、-p path

ビルドされたプロジェクトのディレクトリーの外部にあるプロパティー・ファイルのパスを指定します。

--properties オプションを使用して、スタンドアロン・バンドルに含まれる変数を解決することができます。アプリケーションまたはプラットフォーム内のバンドルは、このオプションでサポートされていません。**--properties** オプションを使用して指定したプロパティー・ファイルは、バンドル内のプロパティーより優先されます。

変数の定義方法と使用方法について詳しくは、変数およびプロパティ・ファイルの定義を参照してください。

--resolve、-r path

以前に CICS Build Toolkit を実行して作成されたディレクトリー構造のパスを指定します。

このオプションは、CICS バンドル・パーツ内の変数を、バンドルのルート・フォルダー内の *variables.properties* ファイルで指定された対応する値に置き換えます。スタンドアロン CICS バンドルの場合、**--properties** オプションで指定された *variables.properties* ファイルで値を指定することも可能であり、その値が優先されます。アプリケーションの一部である CICS バンドルの場合、アプリケーション・バインディング・ルート・フォルダー内の *variables.properties* ファイルに値を指定することも可能で、この指定が優先されます。*variables.properties* ファイルは、ISO-8859-1 (Latin 1) 文字セット・エンコードでなければなりません。

--build オプションと **--resolve** オプションを同時に使用することはできません。

--target, -t .target-file-or-target-ID

使用するターゲット・プラットフォームを指定します。

ターゲット・プラットフォームは、参照先の OSGi バンドル・プロジェクトと OSGi アプリケーション・プロジェクトをビルドするために必要な Java ライブラリー (API) を定義します。CICS リリースなどのターゲット・プラットフォームを事前定義することもできますし、ユーザー作成の Eclipse *.target* ファイルを指定することができます。

以下のターゲット・プラットフォームはデフォルトで選択可能です。

- *com.ibm.cics.explorer.sdk.runtime41.target*
- *com.ibm.cics.explorer.sdk.runtime42.target*
- *com.ibm.cics.explorer.sdk.runtime51.target*
- *com.ibm.cics.explorer.sdk.runtime52.target*
- *com.ibm.cics.explorer.sdk.runtime53.target*
- *com.ibm.cics.explorer.sdk.runtime54.target*
- *com.ibm.cics.explorer.sdk.runtime55.target*
- *com.ibm.cics.explorer.sdk.web.liberty51.target*
- *com.ibm.cics.explorer.sdk.web.liberty52.target*
- *com.ibm.cics.explorer.sdk.web.liberty53.target*
- *com.ibm.cics.explorer.sdk.web.liberty54.target*
- *com.ibm.cics.explorer.sdk.web.liberty55.target*

--verbose、-v

CICS Build Toolkit の進行状況に関する付加的な詳細をコンソールに提供します。

--workspace、-w path

CICS Build Toolkit が使用する Eclipse ワークスペースのパスを指定します。

このオプションを省略した場合、CICS Build Toolkit は一時 Eclipse ワークスペースを作成します。このワークスペースは処理が完了した時点で削除されます。このオプションを使用してワークスペースを指定した場合、ワークスペースは完了時に自動削除されません。詳しくは、Eclipse ワークスペースの管理を参照してください。

使用例

次の例では、ID が `com.ibm.cics.server.examples.jcics` でバージョンが `1.0.1` である CICS バンドルをビルドし、CICS Transaction Server バージョン 5.4 ターゲットを使用して参照先 Java プロジェクトをビルドします。

```
cicsbt --input my/source/dir/*                               ¥
--build "com.ibm.cics.server.examples.jcics(1.0.1)"          ¥
--target com.ibm.cics.explorer.sdk.runtime54.target          ¥
--output my/output/dir
```

次の例では、2 つの異なる入力ディレクトリーにある、ID が `OSGiBundleProject` と `AnotherBundleProject` である最新バージョンの CICS バンドルをビルドし、CICS Transaction Server バージョン 5.1 ターゲットを使用して参照先 Java プロジェクトをビルドします。

```
cicsbt --input my/source/dir/* other/source/dir*             ¥
--build OSGiBundleProject AnotherBundleProject               ¥
--target com.ibm.cics.explorer.sdk.runtime51.target          ¥
--output my/output/dir
```

次の例では、以前にビルドされたプロジェクトの変数を解決します。

```
cicsbt --resolve unresolved/output/dir                         ¥
--output resolved/output/dir
```

次の例では、スタンドアロン・プロパティー・ファイルを使用して、以前にビルドされたバンドル・プロジェクトの変数を解決します。

```
cicsbt --resolve unresolved/output/dir                         ¥
--output resolved/output/dir                                  ¥
--properties props/my.properties
```

上記のサンプルを Windows で使用するには、CICS Build Toolkit 呼び出しと同じ行にすべてのオプションを入力します。パスにスペースが含まれる場合は、必ずパスを引用符で囲んでください。

CICS Build Toolkit の戻りコード

CICS Build Toolkit はビルドの結果を示す戻りコードを出します。戻りコードの 0 は正常終了、1 または 2 は警告、9 以上 16 以下はエラーを示します。

エラーが発生した場合、CICS Build Toolkit はエラーの重大度に応じて処理を停止することもあれば、完了することもあります。詳細な診断情報は、`-v` パラメーターを指定することで取得できます。

| CICS Build Toolkit の戻りコード | 説明 |
|---------------------------|---------------------------------|
| 0 | 操作は正常に完了しました。 |
| 1 | 警告。一部のソース・プロジェクトはインポートされませんでした。 |

| CICS Build Toolkit の戻りコード | 説明 |
|---------------------------|--|
| 2 | 警告。成果物はビルドされ、警告が出されました。 |
| 9 | エラー。無効なオプションが指定されました。 |
| 10 | エラー。ソース・プロジェクトはインポートされませんでした。 |
| 11 | エラー。ビルドするように指定された成果物がソース・プロジェクト内に見つかりませんでした。 |
| 12 | エラー。ビルド・ステップは失敗しました。 |
| 13 | エラー。指定されたターゲット・プラットフォームで問題が発生しました。 |
| 14 | エラー。ソース・プロジェクトが見つかりませんでした。 |
| 15 | エラー。解決ステップが失敗しました。 |
| 16 | エラー。未解決のプロジェクトが見つかりませんでした。 |

DFHDPLOY ユーティリティによる CICS バンドルおよびアプリケーションのデプロイメントおよびアンデプロイメントの自動化

DFHDPLOY ユーティリティは、CICS バンドルおよびクラウド対応 CICS アプリケーションのデプロイ、アンデプロイ、およびその状態の設定を行うためにスクリプトで使用できる一連のコマンドを提供します。

概要

DFHDPLOY ユーティリティは JCL から開始することが可能で、既存のデプロイメントおよび自動化プロシージャと統合することができます。DFHDPLOY ユーティリティを CICS Build Toolkit とともに使用して、継続的デリバリー環境の一部として、CICS バンドルとクラウド対応 CICS アプリケーションのビルドとデプロイメントを自動化します。1067 ページの『CICS Build Toolkit による CICS アプリケーション・ビルド自動化』を参照してください。

DFHDPLOY ユーティリティには、以下の機能を実行するコマンドがあります。

- CICSplex への接続
- CICS バンドルのデプロイ、アンデプロイ、およびその状態の設定
- クラウド対応 CICS アプリケーションのデプロイ、アンデプロイ、およびその状態の設定

DFHDPLOY ユーティリティを UrbanCode Deploy と組み合わせて使用することにより、継続的デリバリー環境を作成できます。詳しくは、UrbanCode の Web サイトで UrbanCode Deploy の CICS TS プラグイン の資料を参照してください。

使用法

DFHDPLOY では、既存のリソース定義で定義されたバンドルまたはアプリケーションのインストールはサポートされません。

注: ジョブ DFHDPLOY を実行するときは、ジョブの処理中にメモリ不足にならないように、REGION=100M 以上を指定します。

セキュリティ

DFHDPLOY ユーティリティーを実行するユーザーには、ユーティリティーが実行される可能性のある LPAR の BINDDIR、BUNDLEDIR、および APPLDIR の各コマンド・オプションで指定された zFS ディレクトリーへの読み取りアクセス権限が必要です。

CICSplex でセキュリティが有効になっている場合、DFHDPLOY ユーティリティーを実行するユーザーには、RACF などの外部セキュリティ・マネージャーで保護されたさまざまな CICSplex SM リソースへのアクセス権限が必要です。

バンドルをデプロイおよびアンデプロイする場合は、次のアクセス・レベルが必要です。

- BAS.APPLICTN に対する ALTER アクセス権限。
- BAS.DEF、OPERATE.APPLICTN、CSD.DEF に対する UPDATE アクセス権限。
- OPERATE.FILE、OPERATE.REGION に対する READ アクセス権限。

アプリケーションをデプロイおよびアンデプロイする場合は、次のアクセス・レベルが必要です。

- CLOUD.APPLICATION に対する ALTER アクセス権限。
- CLOUD.DEF に対する UPDATE アクセス権限。
- CLOUD.PLATFORM に対する READ アクセス権限。

CICSplex SM セキュリティの実装について詳しくは、だれが CICSplex SM リソースへのアクセスを必要とするかの判別を参照してください。

互換性

DFHDPLOY は、バージョン 5.1 以上の CICSplex SM CMAS に接続できますが、CICS TS for z/OSバージョン 5 の以前のリリースでは一部の機能に制限があります。

- アプリケーション・サポートが CICS TS for z/OSバージョン 5.1 で追加されました。
- バンドルとアプリケーションの AVAILABLE 状態と UNAVAILABLE 状態が CICS TS for z/OSバージョン 5.2 で追加されました。
- バンドルの PHASEIN サポートが CICS TS for z/OSバージョン 5.3 で追加されました。

サンプル・プログラム DFH\$DPLY

DFH\$DPLY サンプル・プログラムには、CICSplex 内のサンプル・バンドルとアプリケーションをデプロイ、アンデプロイ、および設定 (任意指定) するための、注釈付きの DFHDPLOY JCL が含まれています。サンプルは

CICSTS55.CICS.SDFHSAMP に提供されています。

DFHDPLOY ユーティリティー・コマンド

コマンドは JCL SYSIN データ定義から読み取られ、メッセージは SYSTSPRT データ定義に書き込まれます。1 つのコマンドを複数行に渡って指定することができます。セミコロン (;) コマンドの終了を示します。先頭の非空白文字がアスタリスク (*) で始まるコメントは無視されます。前後のスペースは無視されます。

アプリケーションが AVAILABLE または UNAVAILABLE から他の状態に移行すると、DFHDPLOY は、そのアプリケーション操作に関連付けられているタスクが UNAVAILABLE 段階にある間、それが完了するのを待機します。

タスクが指定された TIMEOUT 値の期間内に完了しなかった場合、ジョブは警告と 4 または 8 の戻りコードで失敗します。これらの戻りコードの違いについて詳しくは、DFHDPLOY ユーティリティーの戻りコードを参照してください。

SET CICSplex コマンド

後続のコマンドでアプリケーションとバンドルの管理に使用するために接続する CICSplex を定義します。DFHDPLOY はバージョン 5.1 以上の CICSplex SM CMAS に接続できます。

➡—SET CICSplex(*data-value*)—┐
└CMAS(*data-value*)┘

オプション

CICSplex(*data-value*)

接続して入力ストリーム内の後続のコマンドで使用する CICSplex (最大 8 文字) を指定します。

CMAS(*data-value*)

オプションで、DFHDPLOY コマンドを処理する CMAS (最大 8 文字) を指定します。

複数の CICSplex SM API 呼び出しを管理ネットワーク内の複数の CMAS に分離するために、このパラメーターを使用します。例えば、このパラメーターを使用して、ワークロードを保守ポイント CMAS に加えて他の CMAS に分散できます。

注: DFHDPLOY を正常に実行するには、保守ポイント CMAS がアクティブでなければなりません。

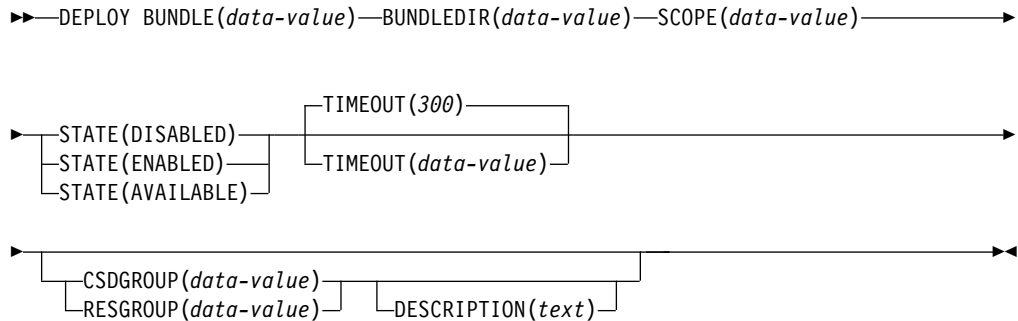
このオプションが指定されていない場合、DFHDPLOY コマンドを処理する CMAS は、EXEC CPSM CONNECT API 呼び出しの場合と同じ方法で決定されます。通常は、同じ MVS イメージおよび同じバージョンの CICSplex SM 上で実行されている CICSplex 内で最後に開始した CMAS になります。CMAS への接続について詳しくは、接続プロセスを参照してください。

注: SET CICSplex コマンドを含んだ DFHDPLOY JOBSTEP は、CICSplex 内の CMAS と同じ SEYUAUTH ライブラリーを参照する必要があります。このライブラリーが参照されない場合は、次のメッセージが表示されます。

DFHRL2004E Unable to connect to CICSplex(). SEYUAUTH library mismatch between DFHDPLOY and CMAS JCL.

DEPLOY BUNDLE コマンド

CICS バンドル・リソースを定義し、その状態を CICS システムまたは CICS システム・グループ内で、DISABLED、ENABLED、または AVAILABLE に変更します。



オプション

BUNDLE (data-value)

デプロイする CICS バンドルの名前 (最大 8 文字) を指定します。

BUNDLEDIR (data-value)

zFS 上の CICS バンドルの場所 (最大 255 文字) を指定します。

CSDGROUP (data-value)

バンドル・リソースの追加先となる CSD グループ (8 文字) を指定します。バンドル・リソースは、**SCOPE** オプションで指定される各 CICS システムの CSD で定義されます。**CSDGROUP** を指定しない場合、バンドルは、デプロイ中は BAS 内に定義され、その後は除去されます。**CSDGROUP** と **RESGROUP** は相互に排他的です。

注: **SCOPE** が複数のシスプレックスに属する CICS 領域のグループを指定していて、それらの CICS 領域の CSD が同じデータ・セット名を使用している場合、バンドル定義は DFHDPLOY ユーティリティーを実行中のシスプレックスの CSD にのみ作成されます。

DESCRIPTION (text)

バンドル定義の説明を指定するオプションの値 (最大 58 文字)。

RESGROUP (data-value)

バンドル・リソースの追加先となる BAS リソース・グループ (8 文字) を指定します。バンドル・リソースは BAS で定義されます。**RESGROUP** を指定しない場合、バンドルは、デプロイ処理中は BAS 内に定義され、その後は除去されます。**RESGROUP** と **CSDGROUP** は相互に排他的です。

SCOPE (data-value)

CICS バンドルをインストールする CICS システムまたは CICS システム・グループの名前 (最大 8 文字) を指定します。

STATE (DISABLED | ENABLED | AVAILABLE)

バンドルのターゲット状態を指定します。有効なオプションは次のとおりです。

- **DISABLED:** バンドル定義を作成し、その定義をインストールした後、バンドルの状態を無効かつ使用不可に変更します。
- **ENABLED:** バンドル定義を作成し、その定義をインストールした後、バンドルの状態を有効かつ使用不可に変更します。
- **AVAILABLE:** バンドル定義を作成し、その定義をインストールした後、バンドルの状態を有効かつ使用可能に変更します。

TIMEOUT (300 | data-value)

コマンドが完了するまでの最大時間 (秒数) を指定するオプションの数値 (1 以上 1800 以下)。適切な **TIMEOUT** 値の選択方法について詳しくは、DFHDPLOY ユーティリティのトラブルシューティングを参照してください。

コールド・スタート時のバンドルの自動インストール

CICS システムの初期化時に CICS バンドルが自動的にインストールされるように構成できます。

- CSD グループの場合は、**DFHCSDUP ADD** コマンドを使用してグループをリストに追加し、**GRPLIST** システム初期設定パラメーターでリストを指定します。
- BAS リソース・グループの場合は、**RESGROUP** を **RESDESC** オブジェクトに関連付けます。詳しくは、BAS でのリソースの自動的なインストールを参照してください。

使用例

この例では、MYPLEX に接続し、既存の WEBSITE バンドルが存在する場合にはそれを除去し、新しいバンドル WEBSITE をデプロイして有効な状態にします。

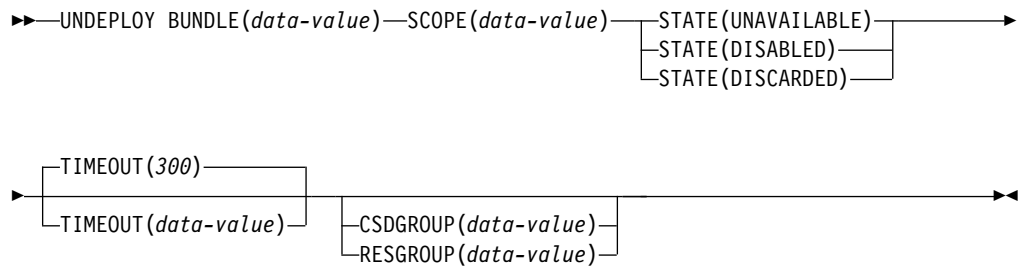
```
//DFHDPLOY
JOB CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
/*
//DFHDPLOY EXEC PGM=DFHDPLOY
/*
//STEPLIB DD DISP=SHR,DSN=CICSTS55.CICS.SDFHLOAD
//          DD DISP=SHR,DSN=CICSTS55.CPSM.SEYUAUTH
//SYSTSPRT DD SYSOUT=*
//SYSIN DD *
SET CICSplex(MYPLEX);
*
UNDEPLOY BUNDLE(WEBSITE) CSDGROUP(BANKING) SCOPE(SYS1)
STATE(DISCARDED);
*
DEPLOY BUNDLE(WEBSITE) BUNDLEDIR(/var/cicsts/bundles/Website_1.0.0/)
CSDGROUP(BANKING) SCOPE(SYS1) STATE(ENABLED) TIMEOUT(60);
/*
```

UNDEPLOY BUNDLE コマンド

CICS システムまたは CICS システム・グループ内で、CICS バンドル・リソースを、UNAVAILABLE、DISABLED、または DISCARDED のいずれかの指定ターゲット状態に変更します。

注: バンドルのデプロイ時に **CSDGROUP** または **RESGROUP** を指定したが、バンドルのアンデプロイ時に同じオプションを再び指定しなかった場合、バンドル定義は削

除されず、CSD グループまたは BAS リソース・グループにリンクされたままになります。DFHDPLOY では、既存のリソース定義で定義されたバンドルのインストールはサポートされません。



オプション

BUNDLE(data-value)

アンデプロイする CICS バンドルの名前 (最大 8 文字) を指定します。

CSDGROUP(data-value)

オプションで、バンドル定義を除去する CSD グループを指定します。これが指定されていない場合、バンドル定義は CSD から除去されません。 **CSDGROUP** と **RESGROUP** を同時に指定することはできません。

RESGROUP(data-value)

オプションで、バンドル定義を除去する BAS リソース・グループを指定します。これが指定されていない場合、バンドル定義はデータ・リポジトリから除去されません。 **RESGROUP** と **CSDGROUP** を同時に指定することはできません。

SCOPE(data-value)

CICS バンドルが除去される CICS システムまたは CICS システム・グループの名前 (最大 8 文字) を指定します。

STATE(UNAVAILABLE | DISABLED | DISCARDED)

バンドルのターゲット状態を指定します。有効なオプションは次のとおりです。

- **UNAVAILABLE**: バンドルを使用不可にします。
- **DISABLED**: バンドル使用不可にしてから無効にします。
- **DISCARDED**: バンドルを使用不可にしてから無効にし、その後バンドルのランタイム・オブジェクトを破棄し、定義を削除します。

TIMEOUT (300 | data-value)

コマンドが完了するまでの最大時間 (秒数) を指定するオプションの数値 (1 以上 1800 以下)。適切な **TIMEOUT** 値の選択方法について詳しくは、DFHDPLOY ユーティリティのトラブルシューティングを参照してください。

例

次の例は、MYPLEX に接続し、バンドル BUND1 をアンデプロイして無効状態にします。

```

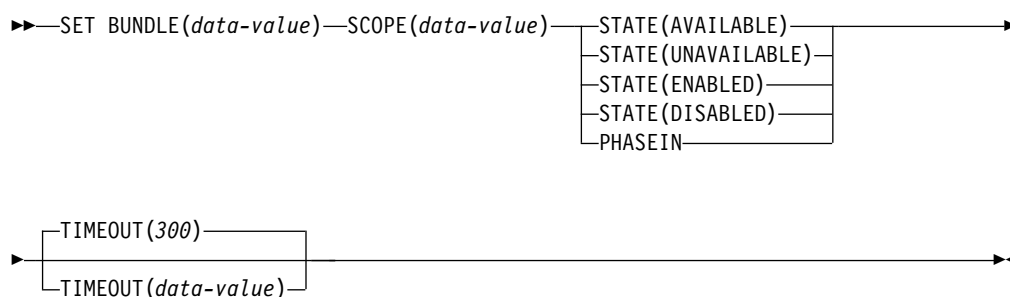
//DFHDPLY1
JOB CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
//*
//DFHDPLOY EXEC PGM=DFHDPLOY
//*

```

```
//STEPLIB DD DISP=SHR,DSN=CICSTS55.CICS.SDFHLOAD
//          DD DISP=SHR,DSN=CICSTS55.CPSM.SEYUAUTH
//SYSTSPRT DD SYSOUT=*
//SYSIN DD *
SET CICSplex(MYPLEX);
*
UNDEPLOY BUNDLE(BUND1) SCOPE(AOR1)
STATE(DISABLED) TIMEOUT(10) CSDGROUP(MYCSDGRP);
/*
```

SET BUNDLE コマンド

CICS システムまたは CICS システム・グループで、インストール済みのバンドルの状態を指定ターゲット状態に変更します。例えば、**SET BUNDLE** コマンドを実行すると、指定した **SCOPE** 内で、バンドルの状態を **DISABLED** に変更できます。このコマンドに **PHASEIN** オプションを指定して使用すると、アクティブ・タスクを中断せずに、上位バージョンの OSGi バンドルをフェーズインできます。



オプション

BUNDLE(data-value)

状態を変更するバンドルの名前 (最大 8 文字) を指定します。

PHASEIN

バンドルのルート・ディレクトリーにあるすべての OSGi バンドルの最上位のセマンティック・バージョンを判別し、そのバージョンがまだ登録されていない場合は、それを OSGi フレームワークに登録します。以前に登録されたバージョンは OSGi フレームワークから除去されます。それ以後のすべての要求では新しいバージョンが使用されますが、アクティブ・タスクでは、そのタスクが完了するまで古いバージョンが引き続き使用されます。バンドルは **ENABLED** 状態でなければならず、そうでない場合フェーズイン操作は失敗します。

PHASEIN オプションと **STATE** オプションを同時に使用することはできません。

注: **PHASEIN** オプションは、CICS TS V5.3 以降でのみ有効です。

SCOPE(data-value)

バンドルがインストールされる CICS システムまたは CICS システム・グループ (最大 8 文字) を指定します。

STATE(AVAILABLE | UNAVAILABLE | ENABLED | DISABLED)

バンドルのターゲット状態を指定します。有効なオプションは次のとおりです。

AVAILABLE

バンドルを有効にして、使用可能にします。

UNAVAILABLE

バンドルを使用不可にします。

ENABLED

バンドルを有効にします。

DISABLED

バンドル使用不可にしてから無効にします。

STATE オプションと **PHASEIN** オプションを同時に使用することはできません。

TIMEOUT (**300** | *data-value*)

コマンドが完了するまでの最大時間 (秒数) を指定するオプションの数値 (1 以上 1800 以下)。適切な **TIMEOUT** 値の選択方法について詳しくは、DFHDPLOY ユーティリティのトラブルシューティングを参照してください。

SET BUNDLE コマンドの CICS 処理について詳しくは、SET BUNDLE を参照してください。

例

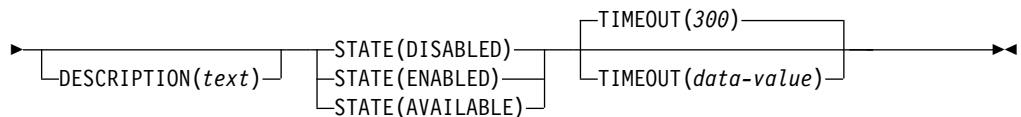
次の例は、MYPLEX に接続し、BUND1 の状態を ENABLED に設定します。

```
//DFHDPLY1
JOB CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
/*
//DFHDPLOY EXEC PGM=DFHDPLOY
/*
//STEPLIB DD DISP=SHR,DSN=CICSTS55.CICS.SDFHLOAD
//          DD DISP=SHR,DSN=CICSTS55.CPSM.SEYUAUTH
//SYSTSPRT DD SYSOUT=*
//SYSIN DD *
SET CICSplex(MYPLEX);
*
SET BUNDLE(BUND1) SCOPE(AOR1)
STATE(ENABLED) TIMEOUT(10);
/*
```

DEPLOY APPLICATION コマンド

CICS アプリケーションを定義し、現在の CICSplex 内にあるプラットフォームで、その状態を DISABLED、ENABLED、または AVAILABLE に変更します。

➡—DEPLOY APPLICATION(*data-value*)—APPLDIR(*data-value*)—BINDDIR(*data-value*)—➡



オプション

APPLICATION (*data-value*)

CICS によって使用されるアプリケーション定義の名前 (最大 8 文字) を指定します。

APPLDIR (*data-value*)

zFS 上のアプリケーション・バンドルの場所 (最大 255 文字) を指定します。

BINDDIR (*data-value*)

zFS 上のアプリケーション・バインディング・バンドルの場所 (最大 255 文字) を指定します。

DESCRIPTION (*text*)

アプリケーション定義の説明を指定するオプションの値 (最大 58 文字)。

STATE (*DISABLED* | *ENABLED* | *AVAILABLE*)

アプリケーションのターゲット状態を指定します。有効なオプションは次のとおりです。

- **DISABLED**: アプリケーション定義を作成し、その定義をインストールした後、アプリケーションの状態を無効かつ使用不可に変更します。
- **ENABLED**: アプリケーション定義を作成し、その定義をインストールした後、アプリケーションの状態を有効かつ使用不可に変更します。
- **AVAILABLE**: アプリケーション定義を作成し、その定義をインストールした後、アプリケーションの状態を有効かつ使用可能に変更します。

TIMEOUT (300 | *data-value*)

コマンドが完了するまでの最大時間 (秒数) を指定するオプションの数値 (1 以上 1800 以下)。適切な **TIMEOUT** 値の選択方法について詳しくは、DFHDPLOY ユーティリティのトラブルシューティングを参照してください。

使用例

次の例では、MYPLEX に接続し、アプリケーション APP1 をデプロイして使用可能な状態にします。

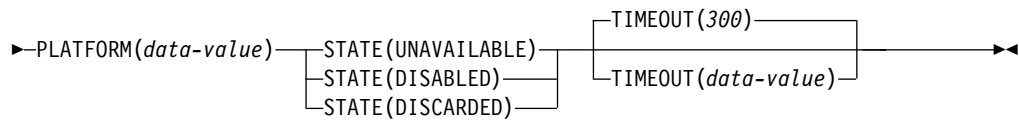
```
//DFHDPLY1
JOB CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
/*
//DFHDPLOY EXEC PGM=DFHDPLOY
/*
//STEPLIB DD DISP=SHR,DSN=CICSTS55.CICS.SDFHLOAD
// DD DISP=SHR,DSN=CICSTS55.CPSM.SEYUAUTH
//SYSTSPRT DD SYSOUT=*
//SYSIN DD *
SET CICSPLEX(MYPLEX);
*
DEPLOY APPLICATION(APP1) STATE(AVAILABLE)
APPLDIR(/var/cicsts/MYPLEX/platform1/applications/application1/)
BINDDIR(/var/cicsts/MYPLEX/platform1/bindings/binding1/);
/*
```

デプロイ中のアプリケーションの状態について詳しくは、CICS Explorer 製品資料内の『Checking the status of an application』を参照してください。

UNDEPLOY APPLICATION コマンド

現在の CICSplex 内のプラットフォームで、CICS アプリケーションを、UNAVAILABLE、DISABLED、または DISCARDED のいずれかの指定ターゲット状態に変更します。

►—UNDEPLOY APPLICATION(*data-value*)—VERSION(*data-value*)—►



オプション

APPLICATION (data-value)

アンデプロイするアプリケーション定義の名前 (最大 8 文字) を指定します。

PLATFORM (data-value)

アプリケーションをアンデプロイする CICS プラットフォームのプラットフォーム定義名 (最大 8 文字) を指定します。

STATE (UNAVAILABLE | DISABLED | DISCARDED)

アプリケーションのターゲット状態を指定します。有効なオプションは次のとおりです。

- UNAVAILABLE: アプリケーションを使用不可にして、現在のアプリケーション操作に関連付けられているタスクが完了するまで待機します。
- DISABLED: アプリケーションを使用不可にして、現在のアプリケーション操作に関連付けられているタスクが完了するまで待機し、その後アプリケーションを無効にします。
- DISCARDED: アプリケーションを使用不可にして、現在のアプリケーション操作に関連付けられているタスクが完了するまで待機し、アプリケーションを無効にしてからアプリケーション・オブジェクトと定義を破棄します。

注: あるアプリケーション・コンテキストにある CICS タスクが別の CICS アプリケーションのエントリー・ポイントにリンクされて、現在のアプリケーション・コンテキストがスタックに置かれる場合があります。DFHDPLOY は、スタック状態のアプリケーション・コンテキストがあるタスクが完了するまで待機することはありません。

TIMEOUT (300 | data-value)

コマンドが完了するまでの最大時間 (秒数) を指定するオプションの数値 (1 以上 1800 以下)。適切な **TIMEOUT** 値の選択方法について詳しくは、DFHDPLOY ユーティリティのトラブルシューティングを参照してください。

VERSION (data-value)

アンデプロイされるアプリケーションのクラウド・スタイル・バージョン番号を (x.y.z) という形式で指定します。x、y、および z は 0 から 255 の数字を示します。

例

次の例では、MYPLEX に接続し、アプリケーション APP1 をアンデプロイして破棄された状態にします。

```

//DFHDPLY1
JOB CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
/*
//DFHDPLOY EXEC PGM=DFHDPLOY
/*
//STEPLIB DD DISP=SHR,DSN=
CICSTS55.CICS
.SDFHLOAD
  
```

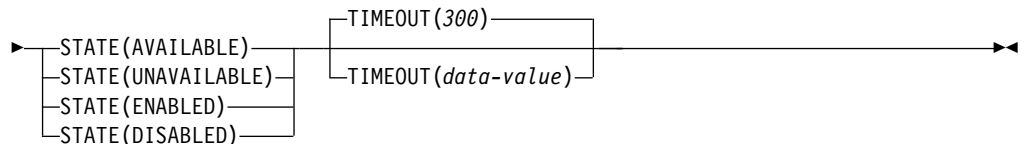
```
// DD DISP=SHR,DSN=
CICSTS55.CPSM
.SEYUAUTH
//SYSTSPRT DD SYSOUT=*
//SYSIN DD *
SET CICSPLEX(MYPLEX);
*
UNDEPLOY APPLICATION(APP1) VERSION(1.0.1)
PLATFORM(MYPLAT1)
STATE(DISCARDED) TIMEOUT(20);
/*
```

アンデプロイ中のアプリケーションの状態について詳しくは、CICS Explorer 製品資料内の『Checking the status of an application』を参照してください。

SET APPLICATION コマンド

現在の CICSplex 内のプラットフォームで、インストール済みのアプリケーションの状態を指定ターゲット状態に変更します。例えば、SET APPLICATION コマンドを実行すると、アプリケーションの状態を AVAILABLE から UNAVAILABLE に変更できます。

►►—SET APPLICATION(*data-value*)—VERSION(*data-value*)—PLATFORM(*data-value*)—►►



オプション

APPLICATION(*data-value*)

状態を変更する CICS アプリケーションのアプリケーション定義名 (最大 8 文字) を指定します。

PLATFORM(*data-value*)

アプリケーションの状態が変更される CICS プラットフォームのプラットフォーム定義名 (最大 8 文字) を指定します。

STATE(AVAILABLE | UNAVAILABLE | ENABLED | DISABLED)

アプリケーションのターゲット状態を指定します。有効なオプションは次のとおりです。

- AVAILABLE: アプリケーションを有効にして、使用可能にします。
- UNAVAILABLE: アプリケーションを使用不可にして、現在のアプリケーション操作に関連付けられているタスクが完了するまで待機します。
- ENABLED: アプリケーションを有効にします。
- DISABLED: アプリケーションを使用不可にして、現在のアプリケーション操作に関連付けられているタスクが完了するまで待機し、その後アプリケーションを無効にします。

注: あるアプリケーション・コンテキストにある CICS タスクが別の CICS アプリケーションのエントリー・ポイントにリンクされて、現在のアプリケーション

1088 CICS TS for z/OS: CICS アプリケーションの開発

CICS は、プログラムがトランザクションで使用されなくなると、そのプログラムの新規コピーを使用します。モジュールが使用中かどうかは、INQUIRE PROGRAM コマンドの RESCOUNT オプションで判別できます。値ゼロは、プログラムが使用されていないことを意味します。CICS は、単一トランザクション中、プログラムの使用が完了し、それ以降の使用がまだ開始していない時点で、プログラムを新規バージョンと置換することができます。

CICS は PRIVATE または SHARED オプションに応じて、DFHRPL または動的 LIBRARY 連結のいずれかから、または LPA 常駐のバージョンを使用して、新しいバージョンをロードします。PRIVATE がデフォルト設定です。

HOLD オプションが指定されている PROGRAM に NEWCOPY を指定すると、エラーが返されます。

CICS バンドルに定義およびインストールされた PROGRAM リソースに、NEWCOPY を指定すると、エラーが返されます。

PHASEIN

CICS は、すべての新しいトランザクション要求に対して、プログラムの新しいコピーを使用するようになりました。現在実行中のすべてのトランザクションに対しては、それらが終了する (RESCOUNT がゼロに等しくなる) まで、引き続き古いコピーを使用します。CICS は PRIVATE または SHARED オプションに応じて、DFHRPL または動的 LIBRARY 連結のいずれかから、または LPA 常駐のバージョンを使用して、新しいバージョンをロードします。PRIVATE がデフォルト設定です。

PHASEIN は REFRESH PROGRAM 機能を実行して、ローダー・ドメインに、プログラムの新しいバージョンがカタログされること、および指名されたプログラムのこのバージョンが将来のすべての ACQUIRE 要求で使用されなければならないことを知らせます。

HOLD オプションが指定されているプログラムに PHASEIN を指定すると、エラーが返されます。

JVM で実行されている Java プログラムに PHASEIN を指定すると、エラーが返されます。

CICS バンドルに定義およびインストールされた PROGRAM リソースに、PHASEIN を指定すると、エラーが返されます。代わりに SET BUNDLE PHASEIN コマンドを使用します。

SCOPE(*data-value*)

PROGRAM がインストールされている CICS システムまたは CICS システム・グループ (最大 8 文字) を指定します。

TIMEOUT (300 | *data-value*)

コマンドが完了するまでの最大時間 (秒数) を指定するオプションの数値 (1 以上 1800 以下)。適切な TIMEOUT 値の選択方法について詳しくは、DFHDPLOY ユーティリティのトラブルシューティングを参照してください。

注: **NEWCOPY** オプションと **PHASEIN** オプションは相互に排他的です。

SET PROGRAM コマンドの CICS 処理について詳しくは、SET PROGRAM を参照してください。

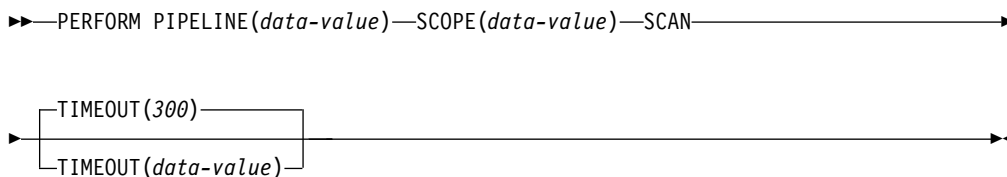
例

次の例では、MYPLEX に接続し、PROGRAM PROG1 の新規バージョンをフェーズインして、PROGRAM PROG2 に対して NEWCOPY を実行します。

```
//DFHDPLY1
JOB CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
/*
//DFHDPLOY EXEC PGM=DFHDPLOY
/*
//STEPLIB DD DISP=SHR,DSN=CICSTS55.CICS.SDFHLOAD
//          DD DISP=SHR,DSN=CICSTS55.CPSM.SEYUAUTH
//SYSTSPRT DD SYSOUT=*
//SYSIN DD *
SET CICSplex(MYPLEX);
*
SET PROGRAM(PROG1) SCOPE(AOR1)
PHASEIN TIMEOUT(300);
*
SET PROGRAM(PROG2) SCOPE(AOR2)
NEWCOPY TIMEOUT(300);
/*
```

PERFORM PIPELINE コマンド

このコマンドに **SCAN** オプションを指定して使用し、PIPELINE の WSDIR 属性で指定された Web サービス・バインディング・ディレクトリーのスキャンを開始します。



オプション

PIPELINE *(data-value)*

パイプラインの名前 (最大 8 文字) を指定します。ワイルドカードを使用できません。

SCAN

PIPELINE で WSDIR 属性が指定されていない場合、スキャン対象はありません。指定されたディレクトリー・ロケーションが有効であれば、CICS はディレクトリー内の Web サービス・バインディング・ファイルを調べて、それらをシステムにインストールするべきかどうかを判別します。

SCOPE *(data-value)*

パイプラインがインストールされている CICS システム、または CICS システム・グループ (最大 8 文字) を指定します。

TIMEOUT (300 | *data-value*)

コマンドが完了するまでの最大時間 (秒数) を指定するオプションの数値 (1 以上 1800 以下)。適切な **TIMEOUT** 値の選択方法について詳しくは、DFHDPLOY ユーティリティのトラブルシューティングを参照してください。

PERFORM PIPELINE コマンドの CICS 処理について詳しくは、SET PROGRAM を参照してください。

例

次の例では、MYPLEX に接続し、AOR1 内のすべての PIPELINE をスキャンしてから、AOR2 内の PIPELINE PIPE1 のみをスキャンします。

```
//DFHDPLY1
JOB CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
//*
//DFHDPLOY EXEC PGM=DFHDPLOY
//*
//STEPLIB DD DISP=SHR,DSN=CICSTS55.CICS.SDFHLOAD
//          DD DISP=SHR,DSN=CICSTS55.CPSM.SEYUAUTH
//SYSTSPRT DD SYSOUT=*
//SYSIN DD *
SET CICSPLEX(MYPLEX);
*
PERFORM PIPELINE(*) SCAN SCOPE(AOR1)
TIMEOUT(300);
*
PERFORM PIPELINE(PIPE1) SCAN SCOPE(AOR2)
TIMEOUT(300);
/*
```

DFHDPLOY ユーティリティー・スクリプトの例

以下の例を使用して、CICS アプリケーションや CICS バンドルをデプロイおよびアンデプロイするための独自の JCL を作成できます。このトピックには、成功したスクリプトや失敗したスクリプトの DFHDPLOY 出力の例も含まれています。

サンプル・プログラム DFH\$DPLY

バンドルをアンデプロイして DISCARDED 状態にする

アプリケーションをデプロイして AVAILABLE 状態にする

DFHDPLOY スクリプト内で条件付き処理を使用する

サンプル・プログラム DFH\$DPLY

DFH\$DPLY サンプル・プログラムには、CICSplex 内のサンプル・バンドルとアプリケーションをデプロイ、アンデプロイ、および設定 (任意指定) するための、注釈付きの DFHDPLOY JCL が含まれています。サンプルは CICSTS55.CICS.SDFHSAMP に提供されています。

バンドルをアンデプロイして **DISCARDED** 状態にする

次のスクリプト例は、MYPLEX に接続し、MYSCOPE 内の CICS 領域でバンドル MYBUND をアンデプロイして DISCARDED 状態にします。

```
//DFHDPLY1
JOB CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
//*
//DFHDPLOY EXEC PGM=DFHDPLOY,REGION=100M
//*
//STEPLIB DD DISP=SHR,DSN=CICSTS55.CICS.SDFHLOAD
//          DD DISP=SHR,DSN=CICSTS55.CPSM.SEYUAUTH
/*
```

```
//SYSTSPRT DD SYSOUT=*
//SYSIN DD *
*
SET CICSplex(MYPLEX);
*
UNDEPLOY BUNDLE(MYBUND) SCOPE(MYSCOPE)
TIMEOUT(40) STATE(DISCARDED);
/*
```

以下は、このスクリプトが正常に実行されたときの DFHDPLOY 出力です。

```
DFHRL2132I Analyzing CICS regions and CSD attributes.
DFHRL2093I BUNDLE(MYBUND) found in SCOPE(MYSCOPE).
DFHRL2129I BUNDLE(MYBUND) state is INSTALLED on 2 CICS regions in
SCOPE(MYSCOPE).
DFHRL2129I BUNDLE(MYBUND) state is ENABLED on 2 CICS regions in
SCOPE(MYSCOPE).
DFHRL2054I Setting BUNDLE state to DISABLED. DFHRL2042I Discarding
BUNDLE(MYBUND).
DFHRL2077I BUNDLE(MYBUND) has been discarded from SCOPE(MYSCOPE).
DFHRL2037I UNDEPLOY command successful.
```

以下は、このスクリプトがタイムアウトが原因で失敗したときに出される可能性のある DFHDPLOY 出力の例です。

```
DFHRL2132I Analyzing CICS regions and CSD attributes.
DFHRL2093I BUNDLE(MYBUND) found in SCOPE(MYSCOPE).
DFHRL2129I BUNDLE(MYBUND) state is INSTALLED on 2 CICS regions in
SCOPE(MYSCOPE).
DFHRL2129I BUNDLE(MYBUND) state is ENABLED on 2 CICS regions in
SCOPE(MYSCOPE).
DFHRL2054I Setting BUNDLE state to DISABLED.
DFHRL2039W The command failed to complete within the specified time out period
of 40 seconds.
DFHRL2129I BUNDLE(MYBUND) state is INSTALLED on 2 CICS regions in
SCOPE(MYSCOPE).
DFHRL2129I BUNDLE(MYBUND) state is ENABLED on 1 CICS regions in
SCOPE(MYSCOPE).
DFHRL2129I BUNDLE(MYBUND) state is DISABLED on 1 CICS regions in
SCOPE(MYSCOPE).
DFHRL2041I TIMEOUT has occurred before BUNDLE(MYBUND) has reached
STATE(DISABLED)
```

```
Status of BUNDLE(MYBUND) in SCOPE(MYSCOPE):
CICS ENABLESTATUS AVAILSTATUS
CICSSYS1 ENABLED NONE MYBUND
CICSSYS2 DISABLED NONE MYBUND
```

```
All Bundle part records for BUNDLE(MYBUND):
CICS BUNDLE ENABLESTATUS AVAILSTATUS BUNDLEPART
CICSSYS1 MYBUND DISABLED NONE SAMP
http://www.ibm.com/xmlns/prod/cics/bundle/TRANSACTION
CICSSYS1 MYBUND ENABLED NONE SAMPFILE
http://www.ibm.com/xmlns/prod/cics/bundle/FILE
CICSSYS2 MYBUND DISABLED NONE SAMP
http://www.ibm.com/xmlns/prod/cics/bundle/TRANSACTION
CICSSYS2 MYBUND DISABLED NONE SAMPFILE
http://www.ibm.com/xmlns/prod/cics/bundle/FILE
```

```
DFHRL2055I Errors have occurred, processing terminated.
```

アプリケーションをデプロイして **AVAILABLE** 状態にする

次のスクリプトの例では、MYPLEX に接続し、アプリケーション MYAPP をデプロイして AVAILABLE 状態にします。

```
//DFHDPLOY1
JOB CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
//*
//DFHDPLOY EXEC PGM=DFHDPLOY,REGION=100M
//*
//STEPLIB DD DISP=SHR,DSN=CICSTS55.CICS.SDFHLOAD
//          DD DISP=SHR,DSN=CICSTS55.CPSM.SEYUAUTH
//SYSIN DD *
*
SET CICSplex(MYPLEX);
*
DEPLOY APPLICATION(MYAPP)
APPLDIR(/var/cicsts/MYPLEX/myplatform/myapplications/myapplication)
BINDDIR(/var/cicsts/MYPLEX/myplatform/mybindings/mybinding)
TIMEOUT(400) STATE(AVAILABLE);
/*
```

以下は、このスクリプトが正常に実行されたときの DFHDPLOY 出力です。

```
DFHRL2044I CICS application definition(MYAPP) created.
DFHRL2045I CICS application definition(MYAPP) installed.
DFHRL2046I Setting application state to ENABLED.
DFHRL2046I Setting application state to AVAILABLE.
DFHRL2012I DEPLOY command completed successfully.
```

```
DFHRL2007I Processing complete.
DFHRL2014I Disconnecting from CICSplex(MYPLEX).
```

以下は、このスクリプトが失敗したときに出される可能性のある DFHDPLOY 出力の例です。

```
DFHRL2044I CICS application definition MYAPP created.
DFHRL2045I CICS application definition MYAPP installed.
DFHRL2046I Setting application state to ENABLED.
DFHRL2064I The state of application MYAPP version 0.0.0 is SOMEDISABLED and
availability is UNAVAILABLE.
```

```
Application Management part records which did not reach desired state:
ENABLESTATUS AVAILSTATUS BUNDLE(version) REGIONTYPE
SOMEDISABLED NONE BUNDMIX2(0.0.0) RegionType1
SOMEDISABLED NONE BUNDMIX2(0.0.0) RegionType2
```

```
DFHRL2064I The state of application MYAPP version 0.0.0 is SOMEDISABLED and
availability is UNAVAILABLE.
```

```
Showing all Application Management part records. Record count = 4
ENABLESTATUS AVAILSTATUS BUNDLE(version) REGIONTYPE
SOMEDISABLED NONE BUNDMIX2(0.0.0) RegionType1
SOMEDISABLED NONE BUNDMIX2(0.0.0) RegionType2
ENABLED UNAVAILABLE ProgramEP(1.0.0) RegionType1
ENABLED UNAVAILABLE ProgramEP(1.0.0) RegionType2
```

```
DFHRL2055I Errors have occurred. 処理は終了しました。
DFHRL2014I Disconnecting from CICSplex MYPLEX.
```

DFHDPLOY スクリプト内で条件付き処理を使用する

JCL COND パラメーターおよび IF/THEN、ELSE、ENDIF ステートメントを使用することによって、DFHDPLOY ジョブの戻りコードを利用して後続のステップを条件付きで処理できます。「z/OS MVS JCL 解説書」の『COND パラメーター』および「z/OS MVS JCL 解説書」の『IF/THEN/ELSE/ENDIF ステートメント構成』を参照してください。

次のスクリプトは、最初にアプリケーションのデプロイが失敗して、戻りコード 4 が返された後、SET APPLICATION コマンドを使用してアプリケーション MYAPP を AVAILABLE 状態に遷移する処理を再試行します。

```
//DPLYAPPL
JOB CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
//*
//DFHDPLOY EXEC PGM=DFHDPLOY,REGION=100M
//*
//STEPLIB DD DISP=SHR,DSN=CICSTS55.CICS.SDFHLOAD
//          DD DISP=SHR,DSN=CICSTS55.CPSM.SEYUAUTH
*
//SYSTSPRT DD SYSOUT=*
//SYSIN DD *
*
SET CICSPLEX(MYPLEX);
*
DEPLOY APPLICATION(MYAPP)
APPLDIR(/var/cicsts/MYPLEX/myplatform/myapplications/myapplication)
BINDDIR(/var/cicsts/MYPLEX/myplatform/mybindings/mybinding)
TIMEOUT(10)
STATE(AVAILABLE);
*
//DPLYFAIL IF (DFHDPLOY.RC = 4) THEN
//DFHDPLOY2 EXEC PGM=DFHDPLOY,REGION=100M
//*
//STEPLIB DD DISP=SHR,DSN=CICSTS55.CICS.SDFHLOAD
//          DD DISP=SHR,DSN=CICSTS55.CPSM.SEYUAUTH
*
//SYSTSPRT DD SYSOUT=*
//SYSIN DD *
* If we TIMED out with an RC=4 then retry making MYAPP
* AVAILABLE.
*
SET CICSPLEX(MYPLEX);
*
SET APPLICATION(MYAPP)
VERSION(4.4.4)
PLATFORM(MYPLATFM)
STATE(AVAILABLE);
//*
// ENDIF
```

DFHDPLOY ユーティリティーの戻りコード

DFHDPLOY ユーティリティーを実行中にエラーが発生すると、メッセージが SYSTSPRT データ定義の宛先に書き込まれ、ゼロ以外の戻りコードでステップが完了します。戻りコードが 8 以上であれば、入力ストリームのコマンドはそれ以上処理されません。

SET CICSPLEX コマンドの結果として接続される CICSPlex SM アドレス空間 (CMAS)、およびアプリケーションまたはバンドルがデプロイまたはアンデプロイされる CICS システムに、関連するメッセージがある場合があります。

| DFHDPLOY の戻りコード | 説明 |
|-----------------|-----------------------------|
| 0 | 入力ストリームのすべてのコマンドは正常に完了しました。 |

| DFHDPLOY の戻りコード | 説明 |
|-----------------|--|
| 4 | <p>処理中に 1 つ以上の警告が発生したか、または TIMEOUT で指定された時間内に一部のコマンドが正常に完了しませんでした。</p> <p>次の例は、4 の戻りコードで終了します。</p> <ul style="list-style-type: none"> • エントリー・ポイントのないバンドルを ENABLED 状態および AVAILABLE 状態に遷移するための試行。 • ターゲット状態が DISABLED のアプリケーションは、INCOMPLETE 状態で終了します。 <p>リソースがその最終ターゲット状態に遷移中にスクリプトがタイムアウトになった場合、この戻りコードは、遷移の成功を示していません。</p> <p>アプリケーションまたはバンドルの処理がさらに行われる可能性があります。入力ストリームのコマンドはそれ以上処理されません。</p> |
| 8 | <p>エラー。最後のコマンドの処理が正常に完了しなかったか、または最終ターゲット状態への遷移が開始する前に処理がタイムアウトになりました。最終ターゲット状態に到達しませんでした。</p> <p>入力ストリームのコマンドはそれ以上処理されません。</p> |
| 12 | <p>エラー。最後に処理されたコマンドを検証できませんでした。</p> <p>入力ストリームのコマンドはそれ以上処理されません。</p> |

JCL COND パラメーターおよび IF/THEN、ELSE、ENDIF ステートメントを使用することによって、DFHDPLOY ジョブの戻りコードを利用して後続のステップを条件付きで処理できます。「z/OS MVS JCL 解説書」の『COND パラメーター』および「z/OS MVS JCL 解説書」の『IF/THEN/ELSE/ENDIF ステートメント構成』を参照してください。例については、1093 ページの『DFHDPLOY スクリプト内で条件付き処理を使用する』を参照してください。

日本語または中国語 (簡体字) で出力メッセージを受け取る

DFHDPLOY の出力メッセージを日本語または中国語 (簡体字) で受け取るには、DFHDPLOY **EXEC** コマンドを使用します。メッセージを表示するために、クライアントで適切なクライアント・コード・ページを設定します。

手順

1. DFHDPLOY EXEC コマンドに適切な PARM パラメーターを指定します。

日本語の出力メッセージの場合、PARM='K' に設定します。

中国語 (簡体字) の出力メッセージの場合、PARM='S' に設定します。

以下に、日本語で出力メッセージを受け取るように構成されている DFHDPLOY EXEC コマンドの例を示します。

```
//DFHDPLOY EXEC PGM=DFHDPLOY,PARM='K'
```

2. メッセージを表示するために、クライアントで適切なクライアント・コード・ページを設定します。

次の表に、各言語の適切なクライアント・コード・ページをリストします。

| 言語 | PARM キーワード | クライアント・コード・ページ | ホスト・コード・ページ |
|-----------|------------|----------------|-------------|
| 日本語 | K | 1390/1399 | 939 |
| 中国語 (簡体字) | S | GB18030/GB2312 | 935 |

UrbanCode Deploy によるデプロイメント

IBM UrbanCode Deploy および CICS TS プラグインを使用して、CICS リソースのデプロイメントとアンデプロイメントを自動化することができます。他の CICS ツールと組み合わせて使用することにより、UrbanCode Deploy はワークフローの効率性を改善して、継続的デリバリー環境に寄与します。

要件

UrbanCode Deploy は、CICS TS for z/OS バージョン 4 以上でサポートされ、構成された CICS 管理クライアント・インターフェース (CMCI) ポートを必要とします。

機能

このプラグインには、以下の操作を自動化するための手順が含まれています。

- CSD リソース、グループ、およびリストのインストール
- BAS リソース、リソース記述、およびグループのインストール
- リソースの廃棄
- リソースの有効化と無効化
- リソースのオープンとクローズ
- リソースの新規コピーとフェーズ
- パイプラインのスキャン
- リソースの有効状況またはオープン状況の検査

プラグインは、コンポーネント・テンプレートも提供します。これにより、コンポーネントのプロセスやプロパティを類似のデプロイメント・シナリオで再使用できます。

詳細な使用方法など、詳しくは UrbanCode Deploy の CICS TS プラグイン の Web サイトを参照してください。

典型的なユース・ケース、機能、評価版のダウンロード方法など、IBM UrbanCode Deploy について詳しくは、UrbanCode Deploy の Web サイトを参照してください。

Node.js アプリケーションのデプロイ

NODEJSAPP バンドル・パーツを作成することによって、Node.js アプリケーションを CICS バンドルでデプロイできます。

始める前に

この作業を始める前に、Node.js アプリケーションの場所と必要な資産 (イメージや HTML ファイルなど) を特定する必要があります。

プロファイルを作成して、CICS が Node.js ランタイムを開始するために使用する構成情報を指定します。このプロファイルを使用して、Node.js アプリケーションがその機能を構成するためにアクセスできる環境変数を指定することもできます。

適切な CICS バンドル・プロジェクトを見つけるか、新しいプロジェクトを作成する必要があります。

Node.js アプリケーションを CICS バンドル・プロジェクトにコピーするか、別の zFS の場所で参照する必要があります。

このタスクについて

以下の手順に従って、CICS バンドルに NODEJSAPP バンドル・パーツを作成できます。

手順

1. 「Project Explorer (プロジェクト・エクスプローラー)」ビューの CICS Explorer で、CICS バンドル・プロジェクトを右クリックし、「新規」>「**Node.js アプリケーション (Node.js Application)**」をクリックします。
「**Node.js アプリケーションの作成 (Create Node.js Application)**」というタイトルのウィザードが開きます。
2. Node.js アプリケーション・バンドル・パーツの名前を入力します。この名前は、バンドルをインストールする CICS 領域内で固有でなければなりません。
「次へ」をクリックします。
3. 実行する初期 JavaScript ファイルを選択します。zFS にエクスポートされた JavaScript ファイルの完全修飾パスが 255 文字を超えていないことを確認する必要があります。「次へ」をクリックします。
4. Node.js アプリケーションのプロファイルを選択します。zFS にエクスポートされたプロファイルの完全修飾パスが 255 文字を超えていないことを確認する必要があります。「完了」をクリックします。

タスクの結果

完了すると、ウィザードによってバンドル・プロジェクト内に NODEJSAPP バンドル・パーツが作成されます。ローカル・ファイル・システムまたは Eclipse ワークスペースからプロファイルを選択した場合、プロファイルは CICS バンドル・プロジェクトにコピーされます。バンドルが使用可能な場合、CICS はプロファイルの構成を使用して Node.js ランタイムを開始し、Node.js アプリケーションの初期 JavaScript ファイルを実行します。

次のタスク

CICS バンドル・プロジェクトをデプロイできるようになりました。

CICS バンドル・プロジェクトのデプロイ

Node.js アプリケーションは一般的に外部モジュールに依存しているため、アプリケーションの実行前にそれらの依存関係を解決する必要があります。CICS プロジェクトをデプロイした後、IBM SDK for Node.js - z/OS によって提供される npm ツールを z/OS UNIX システム・サービス・シェルから実行する必要があります。例えば、依存関係が Node.js アプリケーションの `package.json` ファイルで記述されている場合、**npm install** コマンドを使用して、依存関係をリポジトリからダウンロードし、インストールします。

Node.js ランタイムのインストールの検査

このタスクでは、CICS で Node.js アプリケーションを実行できることを確認するステップと、予期しない結果が生じた場合に役立つ診断フィードバックがある場所について説明します。

始める前に

IBM SDK for Node.js - z/OSをインストールし、そのインストール・ディレクトリに対する読み取り権限と実行権限を CICS 領域ユーザー ID に付与します。

CICS バンドル・ディレクトリ `/usr/lpp/cicsts/cicsts55/samples/nodejs/nodejsivp` が存在していることを確認します。

手順

1. CICS バンドル・ディレクトリとその内容を、選択した新しい場所にコピーします。例えば、`cp -R /usr/lpp/cicsts/cicsts55/samples/nodejs/nodejsivp /u/jdoe/` のようにします。
2. CICS バンドルのコピーで、Node.js プロファイル `profiles/ivp_sample.profile` を編集します。以下の環境変数を更新します。
 - `NODE_HOME`=: IBM SDK for Node.js - z/OS インストール・ディレクトリに設定します。
 - `PORT`=: Node.js アプリケーションが Web ブラウザー要求のサービスに使用できる、使用可能な HTTP ポート番号に設定します。ポートを他のアプリケーションと共用することはできません。
 - `WORK_DIR`=: 出力診断ファイルが書き込まれるディレクトリに設定します。値 `.` は、CICS 領域ユーザー ID のホーム・ディレクトリを意味します。

3. サンプル・リソース定義 DFHNJIVP をグループ DFH\$NODJ から選択したグループにコピーします。
4. DFHNJIVP のコピーを編集して、ステップ 1 (1098 ページ) の CICS バンドルのコピーのディレクトリーに BUNDLEDIR 属性を設定します。
5. DFHNJIVP のコピーをインストールします。Node.js アプリケーションが開始し、HTTP 要求を listen します。
6. Web ブラウザーを使用して、HTTP 要求を Node.js アプリケーションに送信します。例えば、`http://hostname:port/` のようにします。ここで、hostname は CICS が実行されている z/OS 上の TCP/IP スタックの完全修飾ホスト名、port はステップ 2 (1098 ページ) で選択された値です。Web ブラウザーに、「おめでとうございます。Node.js IVP アプリケーション IVPSAMPLE は正常に実行されました。(Congratulations, you have successfully run Node.js IVP Application IVPSAMPLE.)」という応答が表示されます。

次のタスク

Web ブラウザーに予期される応答が表示されない場合は、Node.js アプリケーションのトラブルシューティングの診断情報を確認してください。

関連情報:



<https://www.ibm.com/uk-en/marketplace/sdk-nodejs-compiler-zos>

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。この資料の他の言語版を IBM から入手できる場合があります。ただし、これを入手するには、本製品または当該言語版製品を所有している必要がある場合があります。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。IBM 製品、プログラムまたはサービスに代えて、IBM の知的所有権を侵害することのない機能的に同等のプログラムまたは製品を使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒103-8510

東京都中央区日本橋箱崎町19番21号

日本アイ・ビー・エム株式会社

法務・知的財産

知的財産権ライセンス渉外

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。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様自身の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

*IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119 Armonk,
NY 10504-1785
United States of America*

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。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名前はすべて架空のものであり、類似する個人や企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。これらのサンプル・プログラムは特定物として現存するままの状態を提供されるものであり、いかなる保証も提供されません。IBM は、お客様の当該サンプル・プログラムの使用から生ずるいかなる損害に対しても一切の責任を負いません。

プログラミング・インターフェース情報

CICS には、プログラミング・インターフェースと見なすことのできる資料と、プログラミング・インターフェースと見なすことのできない資料があります。

オンライン製品資料の以下のセクションには、CICS Transaction Server for z/OS, バージョン 5 リリース 5 のサービスを取得するプログラムをお客様が作成するためのプログラミング・インターフェースが含まれています。

- アプリケーションの開発
- Developing system programs
- 保護の概要
- 外部インターフェースに向けた開発
- リファレンス: アプリケーション開発h
- リファレンス: システム・プログラミング
- リファレンス: 接続

オンライン製品資料の以下のセクションには、CICS Transaction Server for z/OS, バージョン 5 リリース 5 のプログラミング・インターフェースとして意図されていない (プログラミング・インターフェースと誤解される可能性のある) 情報が含まれています。

- Troubleshooting and support
- リファレンス: 診断

PDF 形式のマニュアルで CICS 資料にアクセスする場合は、CICS Transaction Server for z/OS, バージョン 5 リリース 5 のサービスを取得するプログラムをお客様が作成するためのプログラミング・インターフェースが以下のマニュアルに含まれています。

- アプリケーション・プログラミング・ガイドおよびアプリケーション・プログラミング・リファレンス
- Business Transaction Services
- Customization Guide
- C++ OO Class Libraries
- Debugging Tools Interfaces Reference
- Distributed Transaction Programming Guide
- External Interfaces Guide
- Front End Programming Interface Guide
- IMS Database Control Guide
- インストール・ガイド
- セキュリティー・ガイド
- Supplied Transactions
- CICSplex SM Managing Workloads
- CICSplex SM Managing Resource Usage
- CICSplex SM アプリケーション・プログラミング・ガイドおよび CICSplex SM アプリケーション・プログラミング・リファレンス
- Java Applications in CICS

PDF 形式のマニュアルで CICS 資料にアクセスする場合は、CICS Transaction Server for z/OS, バージョン 5 リリース 5 のプログラミング・インターフェース

として意図されていない (プログラミング・インターフェースと誤解される可能性のある) 情報が以下のマニュアルに含まれています。

- Data Areas
- Diagnosis Reference
- Problem Determination Guide
- CICSplex SM Problem Determination Guide

商標

IBM、IBM ロゴおよび [ibm.com](http://www.ibm.com/legal/copytrade.shtml)[®] は、世界の多くの国で登録された International Business Machines Corporation の商標または登録商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、<http://www.ibm.com/legal/copytrade.shtml> をご覧ください。

Adobe、Adobe ロゴ、PostScript、PostScript ロゴは、Adobe Systems Incorporated の米国およびその他の国における登録商標または商標です。

インテル、Intel、Intel ロゴ、Intel Inside、Intel Inside ロゴ、Intel Centrino、Intel Centrino ロゴ、Celeron、Intel Xeon、Intel SpeedStep、Itanium、および Pentium は、Intel Corporation または子会社の米国およびその他の国における商標または登録商標です。

Java およびすべての Java 関連の商標およびロゴは Oracle やその関連会社の米国およびその他の国における商標または登録商標です。

Linux は、Linus Torvalds の米国およびその他の国における登録商標です。

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。

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。

製品資料に関するご使用条件

これらの資料は、以下のご使用条件に同意していただける場合に限りご使用いただけます。

適用範囲

IBM Web サイトの「ご利用条件」に加えて、以下のご使用条件が適用されます。

個人使用

これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、非商業的な個人による使用目的に限り複製することができます。ただし、IBM の明示的な承諾をえずに、これらの資料またはその一部について、二次的著作物を作成したり、配布 (頒布、送信を含む) または表示 (上映を含む) することはできません。

商用使用

これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、お客様の企業内に限り、複製、配布、および表示することがで

きます。ただし、IBM の明示的な承諾をえずにこれらの資料の二次的著作物を作成したり、お客様の企業外で資料またはその一部を複製、配布、または表示することはできません。

権利 ここで明示的に許可されているもの以外に、資料や資料内に含まれる情報、データ、ソフトウェア、またはその他の知的所有権に対するいかなる許可、ライセンス、または権利を明示的にも黙示的にも付与するものではありません。

資料の使用が IBM の利益を損なうと判断された場合や、上記の条件が適切に守られていないと判断された場合、IBM はいつでも自らの判断により、ここで与えた許可を撤回できるものとさせていただきます。

お客様がこの情報をダウンロード、輸出、または再輸出する際には、米国のすべての輸出入関連法規を含む、すべての関連法規を遵守するものとします。

IBM これらの資料の内容についていかなる保証もしません。これらの資料は、特定物として現存するままの状態提供され、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なしで提供されます。

IBM オンラインでのプライバシー・ステートメント

サービス・ソリューションとしてのソフトウェアも含めた IBM ソフトウェア製品（「ソフトウェア・オファリング」）では、製品の使用に関する情報の収集、エンド・ユーザーの使用感の向上、エンド・ユーザーとの対話またはその他の目的のために、Cookie はじめさまざまなテクノロジーを使用することがあります。多くの場合、ソフトウェア・オファリングにより個人情報が収集されることはありません。IBM の「ソフトウェア・オファリング」の一部には、個人情報を収集できる機能を持つものがあります。ご使用の「ソフトウェア・オファリング」が、これらの Cookie およびそれに類するテクノロジーを通じてお客様による個人情報の収集を可能にする場合、以下の具体的事項をご確認ください。

CICSplex SM Web ユーザー・インターフェース（メイン・インターフェース）の場合： このソフトウェア・オファリングは、展開される構成に応じて、セッション管理、認証、お客様の利便性の向上、または利用の追跡または機能上の目的のために、それぞれのお客様のユーザー名、およびその他の個人情報を、セッションごとの Cookie および持続的な Cookie を使用して収集する場合があります。これらの Cookie を無効にすることはできません。

CICSplex SM Web ユーザー・インターフェース（データ・インターフェース）の場合： このソフトウェア・オファリングは、展開される構成に応じて、セッション管理、認証、または利用の追跡または機能上の目的のために、それぞれのお客様のユーザー名またはその他の個人情報を、セッションごとの Cookie を使用して収集する場合があります。これらの Cookie を無効にすることはできません。

CICSplex SM Web ユーザー・インターフェース（「Hello World」ページ）の場合： このソフトウェア・オファリングは、展開される構成に応じて、個人情報を収集しないセッションごとの Cookie を使用する場合があります。これらの Cookie を無効にすることはできません。

CICS Explorer の場合:

このソフトウェア・オファリングは、展開される構成に応じて、セッション管理、お客様の利便性の向上、または利用の追跡または機能上の目的のために、それぞれのお客様のユーザー名、およびその他の個人情報を、セッションごとの設定および持続的な設定を使用して収集する場合があります。これらの設定を無効にすることはできませんが、ユーザー・パスワードの暗号化形式でのディスクへの保管は、サインオン中にチェック・ボックスにチェック・マークを付けることによるユーザーの明示的な操作によってのみ有効化することができます。

この「ソフトウェア・オファリング」が Cookie およびさまざまなテクノロジーを使用してエンド・ユーザーから個人を特定できる情報を収集する機能を提供する場合、お客様は、このような情報を収集するにあたって適用される法律、ガイドライン等を遵守する必要があります。これには、エンドユーザーへの通知や同意の要求も含まれますがそれらには限られません。

このような目的での Cookie を含む様々なテクノロジーの使用の詳細については、『IBM オンラインでのプライバシー・ステートメント』(<http://www.ibm.com/privacy/details/jp/ja/>) の『クッキー、ウェブ・ビーコン、その他のテクノロジー』および『IBM Software Products and Software-as-a-Service Privacy Statement』(<http://www.ibm.com/software/info/product-privacy>) を参照してください。

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) 5

アクティブ区分画面 553

アジャイル・サービス・デリバリー 19

アセンブラー言語 575, 904

アプリケーション 106

言語環境プログラムの要件 584

作業用ストレージ 575

準拠していないルーチン 584

制約事項 576

他の言語との混合 587, 684

プログラミング手法 576, 584, 587

31 ビット・アドレッシング 576

64 ビット・レジスター 576

CALL ステートメント 587

DFHECALL マクロ 904

アセンブリー言語 575

作業用ストレージ 575

プログラミング手法 587

64 ビット・アドレッシング・モード・アプリケーション 591

アセンブル 881

アセンブル、TYPE=DSECT 473

宛先 ID 333

アテンション ID 508

アテンション・フィールド 561

アドレッシング、CICS 域の 565

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 1002

アプリケーション 19, 37, 922

インストール 987

管理パート 51

作業単位への分割 856

使用可能化 987

状況 51

デプロイ 987

アプリケーション設計

ストーム・ドレーン作用 234

BIND オプション 1060

アプリケーションのエントリー・ポイント 39, 45

アプリケーションの処理単位

設計 854

アプリケーションのテスト

アプリケーション・テーブル項目の準備 1043

システムの準備 1044

システム・テーブル項目の準備 1043

順次装置の使用 330, 1042

順次端末サポート 1042

単一スレッド・テスト 1042

マルチスレッド・テスト 1042

レグレッション・テスト 1042

アプリケーションのパッケージ化 922

アプリケーション・コンテキスト

(application context) 45

アプリケーション・コンテキスト伝搬 45

アプリケーション・バインディング 37

アプリケーション・パフォーマンスのモニター 242

アプリケーション・バンドル 922, 987

アプリケーション・プログラム

インストール 990

機能シップ 247

作成 2

使用、BMS マップ・セットの 1010

設計 69

相互通信に関する考慮事項 245

テスト 1042

トランザクション・ルーティング 247

パフォーマンスの設計 103

非同期処理 262

分散トランザクション処理 262

分散プログラム・リンク 248

論理レベル 163

アプリケーション・プログラム、Java 715

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 1014

アプリケーション・プログラムの論理レベル 663

アプリケーション・プロジェクト 987

アンエスケープ 382, 400

域、動的ストレージ 104

異常終了 222

異常終了、ユーザー・タスク、EDF 948

異常終了コード 639

異常終了処理 865

異常終了出口機能 236

異常終了出口プログラム 238

異常終了出口ルーチン 238

異常終了のリカバリー 236

一時記憶域

暗黙エンキュー 873

一時記憶域の例 631

概要 630

項目の書き込み 631

項目の更新 631

項目の削除 631

項目の読み取り 631

サンプル 631

チャンネルおよびコンテナへのマイグレーション 161

データ 379

トランザクション間通信で使用する 859

ブラウズ・トランザクション、CEBR 956

補助 95

メイン 95

CICS Service の使用における 630

一時記憶域キュー 3, 95

トランザクション間の類縁性 192

文書テンプレートとして 389

一時記憶域の例

一時記憶域における 631

CICS Service の使用における 631

一時データ 379, 442

一時データ管理の例 629

概要 628

キュー 98, 195

キューの削除 629

区画外 98, 112

区画内 97

サンプル 629

データの書き込み 629

データの読み取り 629

CICS Service の使用における 628

一時データ、区画内

暗黙エンキュー 872

トランザクション間通信での使用 859

一時データ管理

キュー 375, 376

自動トランザクション開始 (ATI) 377

一時データ管理の例

一時データにおける 629

CICS Service の使用における 629

一時データ・キュー 852

大量のデータの 862

文書テンプレートとして 389

一時データ・キュー、区画内 3

一時データ・トリガー・レベル 862

| | | |
|---------------------------|---------------------------------|-----------------------------|
| イベント | エンキュー | 開始データへのアクセス (続き) |
| モニター・ポイント 242 | アプリケーション・プログラムによる | CICS Service の使用における 625 |
| 入り口点、トレース 241 | 明示エンキュー 874 | 回線通信量の削減 566 |
| 印刷 431 | 一時記憶域キューに対する暗黙エンキ | 開発 |
| 一時データ 442 | キュー 873 | 制約事項 747 |
| 行の長さ 435 | 一時データ宛先に対する暗黙エンキ | ベスト・プラクティス 714, 748 |
| BMS ルーティングによる 443 | ュー 872 | 開発環境 751 |
| CICS API に関する考慮事項 444 | リカバリー可能ファイルの暗黙エンキ | 開発者ツールのインストール 751 |
| START コマンド 441 | キュー 871 | 外部 CICS インターフェース (EXCI) 263 |
| 印刷キー 447 | リカバリー不能ファイルの暗黙エンキ | 外部形式制御 563 |
| 印刷キー、ハードウェア 447 | キュー 869 | 外部コントローラー 332 |
| 印刷形式設定 331 | DL/I データベースに対する暗黙エン | 外部呼び出しインターフェース |
| 印刷出力の要求 432 | キュー 873 | (EXCI) 245 |
| 印刷制御ビット 434 | VSAM 内部 108 | 改ページ 522 |
| インストール、アセンブラー・アプリケー | VSAM ファイル内の 108 | 概要 |
| ション・プログラムの | エンコード 719 | チャンネルを使用した動的ルーティング |
| 例、ジョブ・ストリームの 1014 | エントリー・ポイント 923 | 150 |
| インストール、アプリケーション・プログラ | オーバーレイ 107 | 会話型処理 857 |
| ムの | オープン・トランザクション環境 | 会話型プログラミング 334 |
| 使用、ユーザー独自のジョブ・ストリ | (OTE) 78 | 会話パートナー 313 |
| ームの 1026 | CONCURRENCY(REQUIRED) プログ | カウンター名 |
| C 1021 | ラム 86 | 名前付きカウンター 410 |
| COBOL 1016 | OPENAPI プログラム 87 | 書き込み、レコードの 301, 306 |
| PL/I 1021 | 制約事項 89 | 拡張相対バイト・アドレス (XRBA) 271 |
| インストール、HTML テンプレート | 大きな COMMAREA 121, 125, 128, 147 | 拡張相対バイト・アドレッシング (XRBA) |
| 1038 | 大文字変換 | アップグレード 272 |
| インターバル制御機能 352 | 国別文字の 916 | 拡張読み取り専用 DSA (ERDSA) 1005 |
| インターバル制御コマンドの取り消し | 大文字変換、BMS での 507 | 確定応答プロトコル |
| 352 | オブジェクト | 端末管理 328 |
| タスクの開始 352 | 削除 606 | 重ね書き、EDF 画面の 946 |
| 遅延、タスク処理の 352 | 作成 604 | 仮想記憶域 104 |
| 満了時刻 353 | 使用 605 | 仮想記憶域環境 103 |
| 要求 ID の指定 352 | オブジェクトの削除 | 仮想ルックアサイド機能 (VLF) 1004 |
| DELAY コマンド 353 | C++ オブジェクト内 606 | カップリング・ファシリティー・データ・ |
| POST コマンド 353 | オブジェクトの作成 604 | テーブル 283 |
| START コマンド 353 | C++ オブジェクト内 604 | カップリング・ファシリティー・リスト構 |
| インターバル制御機能の START 要求 | オブジェクトを使用 | 造 |
| 861 | C++ オブジェクト内 605 | 現行値 411 |
| インターフェース・スタブ、EXEC 904, | オブジェクト・レベル | 画面からの読み取り、形式設定 506 |
| 915 | 条件、エラー、および例外における | 画面コピー、BMS 447 |
| インターフェース・モジュール | 645 | 画面フィールド、3270 507 |
| 使用 1013 | プラットフォームの相違における 645 | 空のデータ・セット 265 |
| CPI コミュニケーション 903 | オプション | 間接キュー 377 |
| EXEC 903 | ファンクション・キーの、EDF 948 | 完全キー 617 |
| EXEC CICS または EXEC DLI コマ | HANDLE CONDITION コマンド 223 | 管理パート 51 |
| ンドを使用するプログラム 1026 | オペレーティング・システム待機 110 | 管理バンドル 19 |
| SAA リソース・リカバリー 903 | | キー |
| インターリーブ、メッセージ・ルーティン | | 完全 617 |
| グによる会話の 543 | | 総称 292, 617 |
| インバウンド | | 代替 (2 次) 265 |
| データ・ストリーム 564 | | ハードウェア印刷 447 |
| エスケープ・シーケンス 382, 400 | | 物理 280 |
| エラー 1076, 1094 | | キー順データ・セット (KSDS) 265 |
| エラーと例外 | | 記号 382, 394 |
| JCICS 716 | | 値の定義 399, 400 |
| | | 出力マップ 503 |

記号 (続き)

- 入力マップ 503
- 文書テンプレート内での位置指定 394
- 記号カーソル位置決め 497
- 記号参照 382, 394
- 記号置換 382
- 記号テーブル 382
- 記号リスト 382, 399, 400
- 記号レジスター DFHEIPLR 580
- 疑似会話型処理 857
- 既存の JAR ファイルからの OSGi プラグイン・プロジェクトの作成 841
- 既存の Java プロジェクトのプラグイン・プロジェクトへの変換 840
- 既存のバイナリ JAR ファイルからの OSGi プラグイン・プロジェクトの作成 844
- 機能 (PF) キー、CEBR トランザクション 959
- 機能、EDF 927
- 機能管理ヘッダー
- 説明 328
- 機能シップ 245, 247
- 機能伝送 302
- 基本 マッピング・サポート
 - アセンブル、TYPE=DSECT 473
 - 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 1033
 - 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 1010
 - インストール、区分セットの 1040
 - インストール、シンボリック記述マップ・セットの 1035
 - インストール、物理マップ・セットの 1033
 - インストール、マップ・セットの 1029, 1031
 - 大文字変換 507
 - カーソル位置 497
 - カーソルの検出 509
 - 画面コピー 447
 - 完全 460
- 基本 マッピング・サポート
 - 標準 460
 - グループ・フィールド 476
 - コピー機能 567
 - 最小 460
 - サポートされる端末 460
 - 出力マップの初期化 487
 - 出力例 462
 - 準備、マップの 1029
 - シンボリック記述マップ・セット、BMS 用 1035
 - シンボリック・マップ 473
 - データのマップへの移動 488
 - データ・ストリーム 563

基本 マッピング・サポート (続き)

- ディスプレイからのデータの受信 500
- 入力データのマップ 505
- パス長の最小化 526
- 反復フィールド 477
- フィールド 464
- 複合フィールド 476
- 複数マップ画面 567
- 物理マップ 473
- プラットフォーム間のサポート 460
- プログラムにおける使用、シンボリック・マップ・セットの 1035
- ページ作成操作 567
- ページ・ルーティング操作 567
- 変更データ・タグ (MDT) 563
- マクロ 465
- マクロを書くための規則 470
- マップ 463, 563, 564, 565, 567
- マップの作成 465
- マップのためのストレージ 486
- マップ・セット 474
- 無効データ 498
- メッセージ長の削減 527
- リンク・エディット 473
- BMS サポート・レベル 460
- DFHASMVS プロシージャ 1033
- DFHLNKVS プロシージャ 1033
- DFHMAPS、マップのインストール用プロシージャ 1039
- DFHMDMF マクロ 465, 466
- DFHMDI マクロ 465, 468
- DFHMSD マクロ 465, 469
- DFHPSD、区分セット定義の 1040
- EOC 状態 513
- GRPNAME オプション 476
- MAPSET リソース定義 473
- MDT 507
- OCCURS オプション 477
- PROGRAM リソース定義 473
- SEND MAP コマンド 485
- TYPE=DSECT アセンブル 473
- 基本機能 70
- 基本クラス
 - 概要 606
 - ファウンデーション・クラスの概要における 606
- 基本マッピング・サポート
 - シンボリック・マップ 1031
 - タイプ、マップ・セットの 1031
 - パフォーマンスの考慮事項 526
 - 物理マップ 1031
 - マップのアセンブル 472
- 基本マッピング・サポート (BMS)
 - 端末ページング 862

キャンセル

- 満了していない開始要求のキャンセルにおける 625
- キュー
 - 一時データ 375
 - 区画外 376
 - 区画内 375
- キューの削除 629
 - 一時データにおける 629
 - CICS Service の使用における 629
- 競合、端末の 314
- 共通作業域 (CWA) 114
 - 保護 115
- 共通ライブラリー
 - デプロイ
 - Liberty 1051
- 行の長さ、印刷時の 435
- 共用、トランザクション間のデータの 114
- 共用ストレージ 98
- 共用制御、レコードの
 - VSAM RLS 275
- 共用データ・テーブル 282
- 局所性、参照の 105
- 緊急時再始動中のトランザクション・バックアウト
 - XRCINIT (初期設定および終了) 出口 877
- 空間、表示 548
- 区画外一時データ 98, 112
- 区画外キュー 376
- 区画内一時データ 97
 - 暗黙エンキュー 872
- 区画内一時データ・キュー 3
- 区画内キュー 375
- 国別文字
 - 大文字変換 916
- 区分画面 548
- 区分画面、アクティブ 553
- 区分画面の定義 550
- 区分セット
 - インストール 1040
 - ロード、16MB 境界より上のアドレスに 1029
- 区分データ・セット (PDS)
 - 文書テンプレートとして 388
- クライアント領域 248
- クライアント・コード・ページ 387
- クラス
 - 基本 606
 - サポート 610
 - リソース 608
 - リソース識別 607
 - singleton 612
- グループ・フィールド 476
- グローバル・ユーザー出口 364
- XFCBFAIL 878

| | | |
|--------------------------------------|--------------------------|----------------------------|
| グローバル・ユーザー出口 (続き) | 更新、レコードの 301, 619 | コマンド言語変換プログラム (続き) |
| XFCBOUT 878 | 更新セット 265 | NOLINKAGE オプション 888 |
| XFCBOVER 878 | 更新操作、BDAM 303 | NONUM オプション 888 |
| XFCLDEL 878 | 構成 | NOOPSEQUENCE オプション 888 |
| 形式設定画面からの読み取り 506 | アプリケーション 919 | NOOPTIONS オプション 888 |
| 経路リスト 534 | 機動的なサービス・デリバリー 919 | NOPROLOG オプション 888 |
| セグメント・チェーン項目形式 537 | 肯定 316 | NOSEQ オプション 888 |
| 標準入力形式 537 | 項目の書き込み 631 | NOSEQUENCE オプション 888 |
| LIST オプション 537 | 一時記憶域における 631 | NOSOURCE オプション 886, 888 |
| 現行チャンネル | CICS Service の使用における 631 | NOSPIE オプション 888 |
| 概要 131, 136 | 項目の更新 631 | NOVBREF オプション 888 |
| 例、LINK コマンドを使用した 131 | 一時記憶域における 631 | NOXREF オプション 888 |
| 例、XCTL コマンドを使用した 134 | CICS Service の使用における 631 | NUM オプション 888 |
| 言語環境 | 項目の削除 631 | OPMARGINS オプション 888 |
| アセンブラー言語 584 | 一時記憶域における 631 | OPSEQUENCE オプション 888 |
| 異常終了処理 682 | CICS Service の使用における 631 | OPTIONS オプション 888 |
| PL/I 695 | 項目の読み取り 631 | PROLOG オプション 888 |
| 言語環境プログラムの条件処理ルーチン 682 | 一時記憶域における 631 | QUOTE オプション 888 |
| 言語の混合 684 | CICS Service の使用における 631 | SEQ オプション 888 |
| サポートされるコンパイラー 679 | 子エンクレーブ 687 | SEQUENCE オプション 888 |
| サポート・レベル 679 | コピー機能 | SOURCE オプション 886, 888 |
| 条件処理 682 | BMS 567 | SP オプション 888 |
| 条件ハンドラー、ユーザー作成 682 | コピーブックの変換 903 | SPACE オプション 888 |
| ストレージ 683 | コマンド、SYNCPPOINT 215 | SPIE オプション 888 |
| ダイナミック・リンク・ライブラリー 687 | コマンド行 1076, 1094 | SYSEIB オプション 888 |
| ダンプ宛先 680 | コマンド言語変換プログラム 884, 899 | VBREF オプション 886, 888 |
| メッセージ宛先 680 | オプション 899, 901 | XOPTS キーワード 899 |
| 呼び出し可能サービス 680 | 行番号 886 | XREF オプション 888 |
| ランタイム・オプション 687, 691 | APOST オプション 888 | コマンド・レベル・インタープリター |
| および CICS LINK 687 | CBLCARD オプション 888 | 起動 970 |
| 子エンクレーブにおける 687 | CICS オプション 888 | セキュリティの考慮事項 964 |
| CBLPSHPOP 658 | COBOL2 オプション 888 | ENTER キー 969 |
| CEEEXITA ユーザー出口 690 | COBOL3 オプション 888 | 混合アドレッシング・モード・トランザクション 169 |
| CEECSTX ユーザー出口 690 | CPSM オプション 888 | コンテキスト |
| AID 処理 682 | DBCS オプション 888 | コンテナ、BTS またはチャンネルの 149 |
| CEEBINT 692 | DEBUG オプション 888 | コンテナ 3 |
| CICSVAR 環境変数 691 | DLI オプション 888 | 概要 120, 121 |
| DLL 687 | EDF オプション 888 | 基本的な例 123 |
| HANDLE AID コマンド 682 | EPILOG オプション 888 | コンテキスト、BTS またはチャンネル 149 |
| HANDLE CONDITION コマンド 682 | EXCI オプション 888 | 作成 729 |
| HLL ユーザー出口 692 | FLAG オプション 888 | 存続期間 142 |
| PL/I 695 | GDS オプション 888 | チャンネルの設計 144 |
| VS COBOL II 660 | GRAPHIC オプション 888 | プログラムに渡されたコンテナの検出 141 |
| 言語環境プログラムの ランタイム・オプション 687 | LEASM オプション 888 | マイグレーション 158 |
| STORAGE 654 | LENGTH オプション 888 | 一時記憶 161 |
| 言語環境プログラムの CBLPSHPOP ランタイム・オプション 658 | LINECOUNT オプション 888 | 動的にルーティングされたアプリケーション 161 |
| 言語の混合 684 | LINKAGE オプション 888 | LINK コマンド 158 |
| 検査、プログラムの 927 | MARGINS オプション 888 | RETURN コマンド 160 |
| 検出可能フィールド 560 | NATLANG オプション 888 | XCTL コマンド 159 |
| 検証 1062 | NOCBLCARD オプション 888 | 読み取り専用 147 |
| コード・ページ 719 | NOCPSPM オプション 888 | リンクから返されたものの検出 141 |
| 高輝度 561 | NODEBUG オプション 888 | |
| | NOEDF オプション 888 | |
| | NOEPILOG オプション 888 | |
| | NOFEPI オプション 888 | |
| | NOLENGTH オプション 888 | |

コンテナ (続き)
JCICS からの使用 147
JCICS サポート 728
コンパイル 881
コンポーネント
複数、対話式 127
複数のチャンネルに 1 つのコンポーネン
ト 127
1 つのチャンネル - 複数のプログラム
126

[サ行]

サーバー
プログラム 251
領域 248, 251
サーバー・サイド・インクルード・コマン
ド 395
再開 849
サイズ、プログラムの 103
再入可能性 76
作業セット 105
先読みキューイング 315
索引、代替 265
削減、回線通信量の 566
削除、レコードの 304, 620
削除演算子 605
サブスペース 373
サブルーチン 105
サポートされるコンパイラ
Language Environment 679
サポート・クラス 610
ファウンダーション・クラスの概要に
おける 610
参照修正 669
参照セット 107
参照の有効範囲 652
サンプル
コンテナ、基本的な 123
チャンネル、基本的な 123
チャンネルとコンテナ 731
チャンネルを構成するクライアント・プ
ログラム 146
複数の対話式コンポーネント 127
複数のチャンネルに 1 つのコンポーネン
ト 127
1 つのチャンネル - 1 つのプログラム
126
1 つのチャンネル - 複数のプログラム
126
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 148
シーケンスの制御、リソースへのアクセス
357
時間フィールド、EIB の 5
磁気スロット読み取り装置、10/63 559

識別
BDAM レコード 280
VSAM レコード 270
時刻サービス 635
事後初期設定 (PLT) プログラム
(初期設定プログラム)
定義 852
システム障害
再始動のための設計 865
システム情報へのアクセス 4
システム・トレース入り口点 241
システム・リカバリー・テーブル (SRT)
定義 851
システム・ログ・ストリーム
基本定義 851
事前印刷用紙 379
事前変換済みコード 1025
実行診断機能 886, 890, 927
実行単位 663
実動の手順 1063
指定機能文字 560
自動削除 604
自動作成 604
自動条件処理 (callHandleEvent)
条件、エラー、および例外における
643
CICS 条件における 643
自動タスク開始 69, 314
自動トランザクション開始 377
自動トランザクション開始 (ATI) 862
シナリオ
複数の対話式コンポーネント 127
複数のチャンネルに 1 つのコンポーネン
ト 127
1 つのチャンネル - 1 つのプログラム
126
1 つのチャンネル - 複数のプログラム
126
ジャーナル
レコード 99, 212
ジャーナル ID 215
ジャーナル管理
出力の同期化 213
ジャーナル処理 112, 212
ジャーナル・タイプ ID 215
修正、実行の、EDF 946
重大エラー処理 (abendTask)
条件、エラー、および例外における
645
CICS 条件における 645
終了および初期設定の出口
トランザクション・バックアウトのた
めの 877
主記憶装置 104
出力データのチェーニング 327
出力マップ、記号 503

出力マップの初期化 487
手動条件処理 (noAction)
条件、エラー、および例外における
643
CICS 条件における 643
手法、プログラミングの 103
順次端末サポート 330, 1042
照会トランザクション 70
状況フラグ・バイト、経路リスト 537
条件、エラー、および例外
オブジェクト・レベル 645
自動条件処理 (callHandleEvent) 643
重大エラー処理 (abendTask) 645
手動条件処理 (noAction) 643
パラメーター・レベル 646
メソッド・レベル 646
例外処理 (throwException) 644
条件、例外 217
条件処理 863
常駐モード (RMODE)
オプション、CICS アプリケーション
の 1002
初期設定 (PLT) プログラム
定義 852
初期設定および終了の出口
トランザクション・バックアウトのた
めの 877
ジョブ入力サブシステム・コンポーネン
ト、MVS の 449
新規演算子 605
シンボリック記述マップ・セット
プログラムにおける使用 1010
垂直タブ 438
水平タブ 438
据え置きジャーナル出力 214
スタブ、EXEC インターフェース 904,
915
ストーム・ドレーン作用 234
ストレージ
共用可能 359
コンテナ
存続期間 142
静的 107
チャンネル
存続期間 142
プログラム 95
メイン 104
ユーザー・キー 360
CICS キー 360
user 93
ストレージ、動的 590
ストレージ域、動的 104
ストレージ管理
その他 650
ストレージ制御 358
ストレージ保護 358

スプール
 コマンド 4
 ファイル 449
スレッド 718
スレッドとタスク
 JCICS サポート 741
スレッド・セーフ・プログラム 78, 82
 ADDRESS CWA 82
 EXTRACT EXIT 82
 GETMAIN SHARED 82
制御
 BDAM の排他 303
 VSAM ブロックの 274
制御の渡し、戻りを前提とした
 (LINK) 163
制限 747
静的 LIBRARY (DFHRPL) 991
静的ストレージ 107
制約事項 747
 アセンブラ言語 576
 31 ビット・アドレッシング 576
 64 ビット・レジスター 576
 C および C++ 594
 COBOL 654, 665, 669
 PL/I 694
セキュリティ 800
 EDF 929
セキュリティに関する考慮事項
 再開のための 850
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 108
選択フィールド 561
相互通信 245
総称キー 292, 617
総称削除 304
相対バイト・アドレス 617
相対バイト・アドレス (RBA) 265, 271
相対バイト・アドレッシング (RBA)
 アップグレード 272
相対レコード番号 617
相対レコード番号 (RRN) 265, 271
相対レコード・データ・セット
 (RRDS) 265
送達不能なメッセージ 540
装置依存サポート 483
装置依存マップ 482
その他
 ポリモアフィック動作の例 649

[タ行]

ターゲット・ブラットフォーム 705
待機、オペレーティング・システム 110
待機手順 850
代替
 キー 265

代替 (続き)
 索引 265
ダイナミック・リンク・ライブラリー 687
タイム・スタンプ 1060
大容量の COMMAREA 120, 121, 125,
 128, 147, 728
大容量の COMMAREA としてのチャネ
 ル 120, 121, 125, 128, 147, 728
対話式デバッグ
 CECI トランザクション 964
 CECS トランザクション 970
 CEDF トランザクション 927
対話式問題制御システム 242
タスク関連ユーザー出口 364
タスク制御 355
 リソースへのアクセス・シーケンス
 357
妥当性、参照の 106
タブ、垂直 438
タブ、水平 438
単位、作業の 215
 短い作業単位の優先 856
単一画面モード、EDF 941
単一スレッド化 76
単一スレッド・テスト 1042
端末
 オプション 491
 競合 314
 共用 975
 検索対象 633
 サポート、順次 330
 データの受信 633
 データの送信 633
 待ち 316
端末、拡張データ・ストリーム 1033
端末からのデータの受信 633
 端末管理における 633
 CICS Service の使用における 633
端末管理
 印刷形式設定 331
 概要 633
 会話パートナー 313
 確定応答 328
 機能管理ヘッダー (FMH) 328
 コマンド 313
 先読みキューイング 315
 サンプル 633
 出力データのチェーニング 327
 情報の検索 633
 端末からのデータの受信 633
 端末管理の例 633
 端末に関する情報の検索 633
 端末へのデータの送信 633
 中断プロトコル 315
 データの受信 633
 データの送信 633

端末管理 (続き)
 テーブル・ユーザー域 (TCTUA) 118
 入力データのチェーニング 326
 パートナー、会話 313
 半二重モード 313
 ブラケット・プロトコル、LAST オプ
 ション 329
 フリップフロップ・モード 313
 プロトコル、中断 315
 マップ入力データ 505
 論理装置の機能 326
 論理レコードの提示 327
 割り込み 316
 CICS Service の使用における 633
 FMH、アウトバウンド 329
 FMH、インバウンド 328
端末管理の例
 端末管理における 633
 CICS Service の使用における 633
端末制御コマンド 312
端末との通信
 外部設計上の考慮事項 850
端末に関する情報の検索
 端末管理における 633
 CICS Service の使用における 633
端末へのデータの送信 633
 端末管理における 633
 CICS Service の使用における 633
端末を使用しないトランザクション
 EDF 944
チェーニング 317
チェーニング、データの 326, 327
チャネル
 概要 120, 121
 基本的な例 123
 現行、例、LINK コマンドを使用 131
 現行、例、XCTL コマンドを使用 134
 現行の 131, 136
 構成 146
 作成 130, 729
 設計 144
 存続期間 142
 大容量の COMMAREA として 120
 典型的なシナリオ
 複数の対話式コンポーネント 127
 複数のチャネルに 1 つのコンポー
 ネント 127
 1 つのチャネル - 1 つのプログラ
 ム 126
 1 つのチャネル - 複数のプログラ
 ム 126
 動的および分散ルーティング 150
 プログラムに渡されたコンテナの検
 出 141
 マイグレーション 158
 一時記憶 161

- チャンネル (続き)
 - マイグレーション (続き)
 - 動的にルーティングされたアプリケーション 161
 - LINK コマンド 158
 - RETURN コマンド 160
 - XCTL コマンド 159
 - 有効範囲 139
 - 読み取り専用コンテナ 147
 - 利点 128
 - リンクから返されたコンテナの検出 141
 - BTS アクティビティーとの比較 148
 - CICS 変換プログラム 136
 - JCICS からの使用 147
 - JCICS サポート 728
 - LINK および XCTL コマンドの 94
 - LINK コマンド 170
 - RETURN コマンド 172
 - RETURN コマンドにおける 119
- チャンネル (channel) 3
- チャンネルの構成 146
- チャンネルの作成 130
- チャンネルの設計 144
- チャンネルの有効範囲
 - 例、XCTL コマンドを使用した 139
- チャンネルを使用した動的ルーティング 150
- チャンネルを使用する START データのマイグレーション 160
- 中断データ・セット 379
- 調整
 - CICS アプリケーション 1066
- 直接端末 543
- 直列化可能クラス、JCICS 717
- 通常レコードの削除
 - ファイル制御における 620
 - レコードの削除における 620
- 通信域 (COMMAREA) 3
- データ
 - 開始データへのアクセスにおける 625
 - 初期設定 105
 - 他のプログラムへの受け渡し 165
 - 端末に関する情報の検索における 633
 - チェンニング 326
 - 定義 105
 - トランザクション内の格納 92
 - レコード 99
- データ、ディスプレイからの読み取り 505
- データ域の拡張性 614
 - バッファ・オブジェクトにおける 614
 - IccBuf クラスにおける 614
- データ域の所有権 613
 - バッファ・オブジェクトにおける 613
 - IccBuf クラスにおける 613
- データ交換ブロック 886
- データ終了標識文字 331
- データの受け渡し、他のプログラムへの 165
- データの書き込み 629
 - 一時データにおける 629
 - CICS Service の使用における 629
- データの格納、トランザクション内の 92
- データのマップへの移動 488
- データの有効範囲 652
- データの読み取り 629
 - 一時データにおける 629
 - CICS Service の使用における 629
- データベースおよびファイル
 - アプリケーション要件に関する質問 847
 - トランザクション間通信で使用する 859
 - 排他制御 868
 - ロック 868
- データベースへのアクセス 820
- データ変換 150
 - およびチャンネル 150
 - SOAP の例 156
 - 必要な理由 150
- データ・ストリーム
 - 圧縮 566
 - アドレス反復命令 (SBA) 566
 - インバウンド 564
 - バッファ・アドレス設定命令 566
 - RA 命令 566
 - SBA オーダー 566
- データ・セット 309
 - 空 265
 - 順次 112
 - バッチ・データ交換 332
 - ブロック化 278
 - BDAM 278, 280
 - CICS アプリケーション・プログラムからのアクセス 291
 - user 99
- データ・テーブル
 - カップリング・ファシリティー 283, 860
 - 共用 282
 - トランザクション間通信 859, 860
 - ユーザー保守の 859
- テーブル
 - リカバリーの 851
- ディスプレイ
 - 画面 120
- ディスプレイからの読み取り 505
- 出口プログラム
 - 文書テンプレートとして 392
 - 連絡域 393
- テスト
 - C++ 例外およびファウンデーション・クラスにおける 640
- デッドロック 108, 290
 - 予防 274
 - ADCD 異常終了 875
 - AFCW 異常終了 875
- デッドロック、トランザクション回避 860, 874
- DTIMOUT の影響 874
- デバッグ 927
- デバッグ・モード 654
- デフォルト
 - 条件の処理 217
- デプロイ 987, 1094
 - 共通ライブラリー Liberty 1051
 - JVM サーバーへのアプリケーション 1044
- デプロイメント 987
- 伝搬 45
- 同期化アクション
 - ジャーナル出力 213
- 同期点
 - ロールバック 864
- 同期点処理 215, 216
- 同期点追跡、DPL 251
- 統合 CICS 変換プログラム 881
- 統合変換プログラム 5, 881
- 同時ブラウズ 296
- 動的
 - ストレージ域 104
 - トランザクション・バックアウト・プログラム 236
 - トランザクション・ルーティング 167
 - プログラム 107
- 動的 LIBRARY 991
- 動的 LIBRARY リソースの使用 991
- 動的起動、変換プログラムの 888
- 動的削除 605
- 動的作成 605
- 動的ストレージ、拡張 590
- 動的トランザクション・バックアウト
 - 使用の決定 864
- 動的にルーティングされたアプリケーション
 - チャンネルおよびコンテナへのマイグレーション 161
- 動的プログラム・リンク
 - 類縁性 207
- 都市コード 1096
- トラブルシューティング 1076, 1094
- トランザクション 854
 - デッドロック 288
 - ルーティング 245, 247
 - ルーティング、動的 167

トランザクション (続き)
類縁性 162, 247, 352, 355, 359, 368
トランザクション ID
CEBR 956
CECI 970
CEDF トランザクション 927
トランザクション開始の例
トランザクションの非同期の開始にお
ける 625
CICS Service の使用における 625
トランザクション間通信 858
リソースの使用 858
COMMAREA の使用 858
トランザクション間の通信 858
リソースの使用 858
トランザクション間の類縁性
安全なプログラミング手法 179
BTS コンテナの使用 183
COMMAREA 179
ENQMODEL での DEQ の使用
183
ENQMODEL での ENQ の使用
183
TCTUA 180
一時記憶 192
一時記憶域データ共用 192
関係 および存続時間
グローバル関係 206
端末関係 209
ユーザー ID 関係 210
BAPPL 関係 204
関係および存続期間 204
危険性のある プログラミング手法
一時データ 195
グローバル・ユーザー出口 176
DELAY および CANCEL REQID
コマンド 200
INQUIRE および SET コマンド
176
POST コマンド 202
RETRIEVE WAIT および START
コマンド 196
START および CANCEL REQID
コマンド 198
危険なプログラミング手法 184
共用ストレージの使用 185
タスク存続期間ストレージの共用
188
CWA 184
DEQ の使用 190
ENQ の使用 190
LOAD PROGRAM HOLD の使用
186
WAIT EVENT の使用 189
プログラミング手法 177
類縁性存続期間 203

トランザクション間の類縁性 (続き)
類縁性トランザクション・グループ
204
トランザクション再始動
DTB 後の使用の決定 865
トランザクション作業域 92
トランザクション障害
呼び出される機能 863
トランザクションとシステム間の類縁性
176
トランザクションの開始
トランザクションの非同期の開始にお
ける 625
CICS Service の使用における 625
トランザクションの非同期の開始 624
開始データへのアクセス 625
トランザクション開始の例 625
トランザクションの開始 625
満了していない開始要求のキャンセル
625
CICS Service の使用における 624
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 176
トランザクションとシステム間の類縁
性 176
トランザクション分離 358
トランザクション・グループ
類縁性 204
トランザクション・リスト・テーブル
(XLT)
定義 852
トリガー・フィールド 559
トレース 799
説明 240
トレース入り口点 241
トレース出力の活動化 639

[ナ行]

内部設計フェーズ 863
名前付きカウンター 409
オプション・テーブル 411
概要 409
カウンター名 410
カップリング・ファシリティ・リス
ト構造 411
現行値 410
最小値 410
最大値 410
名前付きカウンター・フィールド 410
プール 411
CICS API 413
DFHNC001 411
DFHNCO マクロ 411
日時サービス
日時サービスの例 635

日時サービス (続き)
CICS Service の使用における 635
日時サービスの例
日時サービスにおける 635
CICS Service の使用における 635
入力順データ・セット (ESDS) 265
入力データ
チェーニング 326
入力データのマップ 505
入力マップ、記号 503
ヌル値の使用 567
ヌル・パラメーターを持つ DFHNCTR
CALL の例 424
ネイティブ・ランタイム・ライブラリー
679

[ハ行]

ページ遅延 540
パートナー、会話 313
パートナーの範囲 262
排他制御、レコードの
BDAM 303
VSAM 274
VSAM RLS 275
排他制御、VSAM
(エンキューも参照) 869
排他的リソース 108
バイト配列の処理 719
バインド・オプション
カーソル固定 1062
検証 1062
バインド・オプション
カーソル固定 1062
反復可能読み取り 1062
分離レベル 1062
バインド・プロセス
オプションと考慮事項 1060
プログラムの変更後 1059
バックアウト、リソースの 215
バッチ・データ交換 332
宛先 ID 333
確定応答 334
DEFRESP オプション 334
ISSUE WAIT コマンド 334
NOWAIT オプション 334
バッチ・モード JVM 838
バッファ 613, 616
バッファ・オブジェクト
データ域の拡張性 614
データ域の所有権 613
IccBuf コンストラクター 614
IccBuf メソッド 615
IccResource サブクラスの処理 615
パラメーター
null 424

パラメーター受け渡し 651
パラメーター受け渡し規則
 その他 651
パラメーター・レベル
 条件、エラー、および例外における 646
 プラットフォームの相違における 646
範囲、パートナーの 262
バンドル 705
半二重モード 313
反復可能読み取り 1062
反復フィールド 477
非 CICS プリンター 431, 443
日付サービス 635
日付フィールド、EIB の 5
非同期 API 7, 9, 10, 11, 12, 571
非同期ジャーナル出力 213
非同期処理 245, 262
非同期要求 113
非ブロック化引数 280
表示
 レジスター、EDF の 949
表示スペース 548
表示特性 488
表示窓 548
表題、メッセージの 541
ファイル制御
 概要 264
 更新、レコードの 619
 削除、レコードの 620
 サンプル 620
 通常レコードの削除 620
 ファイル制御の例 620
 レコードの書き込み 618
 レコードの更新 619
 レコードの再書き込み 619
 レコードの削除 620
 レコードのブラウズ 620
 レコードの読み取り 617
 ロックされたレコードの削除 620
 BDAM データ・セット 280
 CICS Service の使用における 616
 ESDS レコードの書き込み 619
 ESDS レコードの読み取り 618
 KSDS レコードの書き込み 619
 KSDS レコードの読み取り 617
 RRDS レコードの書き込み 619
 RRDS レコードの読み取り 618
ファイル制御の例
 ファイル制御における 620
 CICS Service の使用における 620
ファイル制御リカバリー管理プログラム
 出口 876
ファイルの順次読み取り 620

ファウンデーション・クラスの異常終了コード
 条件、エラー、および例外における 639
ファウンデーション・クラスの概要 606
 リソース・オブジェクトでのメソッドの呼び出し 613
 リソース・オブジェクトの作成 611
ファンクション・キー 931, 969
フィールド
 グループ 476
 反復 477
 複合 476
 ブランク 564
 BMS 464
 複合フィールド 476
 複数の基底レジスター 580
 複数のスレッド 718
 複数バージョン 923
 複数バージョンのアプリケーション 923
 複数ページ出力 567
 複数マップ画面 567
 ブックマーク 403, 405
物理キー 280
物理マップ・セット
 インストール 1033
浮動マップ 521
ブラウズ 311
 レコード 295
 DELAY 311
 SUSPEND 311
ブラウズ操作
 BDAM 299
プラグイン・プロジェクトの作成 706
フラグ・バイト、経路リスト 537
ブラケット・プロトコル、LAST オプション 329
プラットフォーム 19, 922
 アプリケーション状況 51
 アプリケーション・バインディング 37
プラットフォームの相違
 オブジェクト・レベル 645
 条件、エラー、および例外における 645
 パラメーター・レベル 646
 メソッド・レベル 646
プラットフォーム・バンドル 19
ブランク・フィールド 564
ブリッジ (3270)
 ADS 記述子 475
フリップフロップ・モード 313
プリンター
 非 CICS 431, 443
 3270 433
 オプション 434
CICS 431

プリンター (続き)
 特性の判別 439
SCS 436
プログラミング上の制約事項
 アセンブラ言語 576
C および C++ 594
COBOL 654, 665, 669
PL/I 694
VS COBOL II 660
プログラミング手法
 アセンブラ言語 576, 584, 587
 アセンブリー言語 587
 一般 103
C および C++ 594
COBOL 654, 661, 665
PL/I 693, 694, 695, 697
プログラミング・モデル 69
プログラム
 サイズ 103
 テスト 927
 文書テンプレートとして 390
プログラム制御
 概要 623
 サンプル 623
 他のプログラムへのデータの受け渡し 165
 プログラムの論理レベル 163
 別のプログラムへのリンク 163
 CICS Service の使用における 623
プログラム設計
 会話型 334
プログラムに渡されたコンテナの検出 141
プログラムのコンパイル 636
 コンパイル、実行、およびデバッグにおける 636
プログラムの実行
 コンパイル、実行、およびデバッグにおける 638
プログラムのデバッグ 638
 コンパイル、実行、およびデバッグにおける 638
プログラム分離スケジューリング 873
プログラムへのリンク、戻りを前提とした 163
プログラム・エラー・プログラム (PEP)
 設計上の考慮事項 865
プログラム・ストレージ 95
プログラム・ラベル、EDF の 946
ブロック化データ・セット 278
ブロック参照 280
フロントエンド・プログラミング・インターフェース (FEPI) 245
分散アプリケーションの設計 74
分散プログラム・リンク 162
オプション 249

分散プログラム・リンク (続き)

クライアント領域 248

サーバー領域 248, 251

サーバー・プログラム 251

独立同期点 251

プログラミング上の考慮事項 256

例外条件 259

COMMAREA オプション 251

DPL API サブセット 256

REMTENAME オプション 251

REMOTESYSTEM オプション 251

SYSID オプション 251

TRANSID オプション 251

文書 380

および CICS Web サポート 380

コード・ページ変換 387

再利用 406

削除 409

作成 397

取得 406

データの置換 405

データの追加 403

ブックマーク 403, 405

文書テンプレート 380

一時記憶域キュー 389

一時データ・キュー 389

埋め込みコマンド

#echo 394, 395

#include 395

#set 382, 394, 395

および CICS Web サポート 380, 387, 388

記号 380, 382, 394

記号リスト 382

キャッシング 385

出口プログラムでの指定 393

区分データ・セット (PDS) 388

セットアップ 387

出口プログラム 392, 393

文書内での位置指定 397, 403, 405

連絡域 393

CICS ファイル 389

CICS プログラム 390, 391

DFHDHTL マクロ 390, 391

HFS ファイル (HFS file) 388

文書テンプレートの記号 380

文書テンプレートのキャッシング 385

文書テンプレートの連絡域 393

分離レベル 1062

ページ作成操作 567

ページ不在 105

ページング

影響の軽減 105

ページ・オーバーフロー 542

ページ・ルーティング操作 567

ベスト・プラクティス

開発 714, 748

ヘッダー・ファイル

インストール済みの内容 637

変換 5, 881

COBOL 669

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 881

動的起動 888

変換プログラム・データ・セット 884,
899

ペン検出可能フィールド 560

変更データ・タグ 507, 563

変数、CECI/CECS 971

補助一時記憶域 95

補助トレース 110

ホスト・コード・ページ 387

ポリシー 19

ポリモアフィック動作 647

その他 647

ポリモアフィック動作の例 649

ポリモアフィック動作の例

その他 649

ポリモアフィック動作における 649

[マ行]

マイグレーション

チャンネルを使用する START データの
160

待ち、端末 316

マッピング 715

マップ

記号出力 503

記号入力 503

作成 465

出力の初期化 487

セット 107, 474

装置依存 482

データの移動 488

浮動 521

リンク・エディット 473

BMS 463, 564, 565, 567

マップのリンク・エディット 473

マップ・セット

追加、CSECT 1039

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 1010

ロード、16MB 境界より上のアドレス
に 1029

マルチスレッド化 76

マルチスレッド・テスト 1042

満了時刻

指定 353

満了していない開始要求のキャンセル

トランザクションの非同期の開始にお
ける 625

CICS Service の使用における 625

メイン一時記憶域 95

メソッド・レベル

条件、エラー、および例外における

646

プラットフォームの相違における 646

メッセージ、送達不能な 540

メッセージの表題 541

メッセージ・ルーティング 534

目的

ファイル制御の例における 622

文字セット 387

モジュラー・プログラム 107

戻りコード 1076, 1094

[ヤ行]

ユーザー

ストレージ 93

データ・セット 99

トレース入り口点 241

ユーザー置換可能モジュール 364

ユーザー出口

緊急時再始動 876

トランザクション・バックアウト 876

ユーザー保守テーブル 282

ユーザー・キー・ストレージ 360

ユーザー・ログ・レコード・リカバリー・
プログラム

出口 876

要求応答単位 (RU) 326

呼び出し、EDF の 928

呼び出し可能サービス 680

呼び出し規則 651

読み取り、形式設定画面からの 506

読み取り、ディスプレイからのデータの
505

読み取り、レコードの 291

読み取り専用 DSA (RDSA) 1005

読み取り専用コンテナ 147

[ラ行]

ライト・ペン検出可能フィールド 560

ライブラリー・ルックアサイド機能

(LLA) 1004

ランタイム・ライブラリー

ネイティブ 679

Language Environment 679

ランナウェイ・タスク 110

リカバリー

順次端末サポート 330

リカバリー (続き)
 同期点 215
 問題回避 100
 リソースの 108
リカバリー可能性
 必要な CICS 定義 851
リカバリーと再始動プログラムのテスト
 852
リカバリーと再始動プログラムの文書化
 852
リソース
 アクセス・シーケンスの制御 357
 排他制御 108
 リカバリー可能 108
リソース識別クラス 607
 ファウンデーション・クラスの概要に
 おける 607
リソース定義
 JCICS の 716
リソース・アダプター 789
リソース・オブジェクト
 作成 611
リソース・オブジェクトでのメソッドの呼
 び出し
 ファウンデーション・クラスの概要に
 おける 613
 CICS リソースの使用における 613
リソース・オブジェクトの作成 611
 ファウンデーション・クラスの概要に
 おける 611
 CICS リソースの使用における 611
 singleton クラス 612
リソース・クラス 608
 ファウンデーション・クラスの概要に
 おける 608
リソース・リカバリー
 SAA 互換性 855
リバース割り込み 316
リモート・トランザクション、EDF の
 943
リモート・リンクされているプログラム
 DPL 944
 EDF 944
領域間通信 263
リンク
 OSGi サービス 1052
リンクから返されたコンテナの検出 141
リンク・エディット 881, 887
リンク・パック域 (LPA) 1004
ルーティング、トランザクション 245
ルーティング端末 543
類縁性 175
 安全なプログラミング手法 177
 危険性のあるプログラミング手法 177
 危険なプログラミング手法 177
 推奨されるプログラミング手法 177

類縁性グループ 204
類縁性の関係
 ユーザー ID 210
 BAPPL 204
 LOCKED 207
 LU 名 209
ルックアサイド・トランザクション 550
例 720
 チャンネルを使用する CICS サーバー・
 プログラム 147
例外 639
例外条件
 説明 217
 HANDLE CONDITION コマンド
 222, 223
 IGNORE CONDITION コマンド 227
例外処理 (throwException)
 条件、エラー、および例外における
 644
 CICS 条件における 644
例外トレース入り口点 241
レグレーション・テスト 1042
レコード
 更新 301
 削除 304
 作成 301, 306
 識別 270, 280
 ジャーナル 212
 追加 306
 長さ 99
 ブラウズ 291
 読み取り 291
 ロック 274
 ロック (RLS) 275
 BDAM データ・セットへの追加 308
レコード記述フィールド 309
レコードの書き込み
 ファイル制御における 618
 CICS Service の使用における 618
 ESDS レコードの書き込み 619
 KSDS レコードの書き込み 619
 RRDS レコードの書き込み 619
レコードの更新
 ファイル制御における 619
 CICS Service の使用における 619
レコードの再書き込み 619
レコードの削除
 通常レコードの削除 620
 ファイル制御における 620
 ロックされたレコードの削除 620
 CICS Service の使用における 620
レコードの追加 306
レコードのブラウズ 620
 ファイル制御における 620
 CICS Service の使用における 620

レコードの読み取り
 ファイル制御における 617
 CICS Service の使用における 617
 ESDS レコードの読み取り 618
 KSDS レコードの読み取り 617
 RRDS レコードの読み取り 618
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
 267
レコード・ロック 275
レベル、アプリケーション・プログラム論
 理の 163
ローカル・コピー・キー 447
ロード・ライブラリー
 2 次エクステンツのサポート 1013
ロギング 113
ロック
 アプリケーション・プログラムでの
 868
 リカバリー可能ファイルの暗黙ロック
 871
 リカバリー不能ファイルの暗黙ロック
 869
ロックされたレコードの削除
 ファイル制御における 620
 レコードの削除における 620
論理作業単位 (LUW)
 使用する同期点 215
論理装置 (LU)
 機能 326
論理デバイス・コンポーネント 556
論理メッセージの規則 516
論理レコードの提示 327
論理レベル、アプリケーション・プログラ
 ムの 163, 663

[ワ行]

割り込み 316

[数字]

10/63 磁気スロット読み取り装置 559
2 次エクステンツ、CICS ロード・ライブ
 ラリー 1013
31 ビット・アドレッシング
 アセンブラ言語 576
 COBOL 654
31 ビット・モード・トランザクション
 169
32 K 以上の COMMAREA 120, 728
32 K 超の COMMAREA 728
3262 プリンター 431
3270 画面フィールド 507
3270 情報表示システム 1033

3270 ディスプレイ 317
3270 ファミリー
 アウトバウンド・データ・ストリーム 345
 アテンション ID 348
 アテンション・キー 348
 インバウンド・フィールド形式 350
 書き込み制御文字 336
 拡張属性 341
 カラー、基本 339
 輝度 339
 基本カラー 339
 属性、拡張 341
 端末への書き込み 336
 データ・ストリーム 335
 データ・ストリーム、アウトバウンド 345
 データ・ストリーム中のオーダー 342
 データ・ストリーム・オーダー 342
 入力 348
 バッファ 336
 表示特性 338
 フィールド 338
 フィールド形式、インバウンド 350
 フィールド属性 339
 不定形式モード 351
 変更データ・タグ 339
 保護 339
 読み取り 349
 AID 348
 MDT 339
3270 ブリッジ
 ADS 記述子 475
3270 プリンター 433
 オプション 434
3289 プリンター 431
3290 ディスプレイ 548
32K 以上の COMMAREA 128, 147
3601 論理装置 556
3770 バッチ論理装置 556
3770 バッチ・データ交換論理装置 556
3790 バッチ・データ交換論理装置 556
64 ビット・アドレッシング・モード
 アセンブリ言語 591

A

abend
 パラメーター・レベルにおける 646
ABEND コマンド 236
abendTask
 CICS 条件における 642
ACCEPT ステートメント、COBOL 654
ACCUM オプション 434, 515
ACK 316
ACTPARTN オプション 516, 553

ADCD 異常終了 875
ADDRESS COMMAREA コマンド 165
ADDRESS コマンド 4
ADDRESS 特殊レジスター 661
ADS 記述子 475
AFCW 異常終了 875
AFFINITY 属性 204
AFFLIFE 属性 204
AFTER オプション 353
AIX、CICS
 プラットフォームの相違における 645
ALARM オプション 492
ALLOCATE コマンド 111
 待機禁止、NOSUSPEND オプション 111
ALLOCERR 状態 451
ALTPAGE 値 521
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 1002
AMODE(64)
 アセンブリ言語 591
APAK トランザクション 439
APCG 104
API
 DPL のサブセット 256
APOST オプション 888
APPLCTN 987
APPLDEF 987
application/x-www-form-urlencoded 382
argc 594
argv 594
ASIS オプション 507
ASKTIME コマンド 352
assign
 ファイル制御の例における 622
ASSIGN コマンド 4, 323, 439
 オプション 323
 DESTCOUNT オプション 542
 MAPCOLUMN オプション 525
 MAPHEIGHT オプション 525
 MAPLINE オプション 525
 MAPWIDTH オプション 525
 MSR オプション 559
 PAGENUM オプション 542
AT オプション 353
ATI 69, 314, 377
ATNI 247
ATTENTION キー 316
AUTOPAGE オプション 519
AZI6 247

B

BAKR (分岐およびスタック) アセンブラ
 一命令 576

BASE オプション 487
BDAM 309, 617
 更新操作 303
 データ・セット 278, 280
 排他制御 303
 ブラウズ操作 299
BDI 431
beginInsert
 レコードの書き込みにおける 618
BGAM 313
BIND オプション
 アプリケーション設計での 1060
 プログラム準備での 1060
 RETAIN 1061
BIND タイム・スタンプ 1060
BMS 431
 ルーティング 443
BMS コマンド 312
BMS による端末ページング 862
BOTTOM コマンド、CEBR トランザクシ
 ョン 960
BRACKET オプション 329
BROWSE TEMP STORAGE オプシ
 ョン、CEDF 948
BTS アクティビティ 148
buffer
 トランザクション開始の例における
 626, 627
buffer (パラメーター)
 ポリモアフィック動作における 648
build 1076
BUILDCHAIN 326

C

C および C++ 593
 作業用ストレージ 593
 サポート 593
 制約事項 594
 他の言語との混合 684
 地域サポート 601
 引数 599
 プログラミング手法 594
 EIB、アクセス 600
 XPLink 602, 603
C および C++ での地域サポート 601
C 言語に関する考慮事項
 LENGTH オプションのデフォルト 99
 struct、シンボリック記述マップ・セ
 ット 1031
CALL DL/I インターフェース、
 COBOL 658
CALL ステートメント
 アセンブラ言語 587
callHandleEvent
 CICS 条件における 642

CANCEL コマンド 352

catch

例外処理 (throwException) における
644

C++ 例外およびファウンデーション・
クラスにおける 639, 641

CBLCARD オプション 888

CDUMP 594

CEBR トランザクション 956

一時記憶域のブラウズ 956

一時データ 963

開始 956

セキュリティの考慮事項 956

表示 958

ブラウズ・トランザクション 956

BOTTOM コマンド 960

CEBR の開始 956

COLUMN コマンド 960

FIND コマンド 961

GET コマンド 961

LINE コマンド 961

PURGE コマンド 962

PUT コマンド 962

QUEUE コマンド 962

SYSID コマンド 962

TERMINAL コマンド 962

TOP コマンド 963

CECI トランザクション

アンパーサンド (&) 971

概要 964

画面レイアウト 964

共用ストレージ 975

コマンド行 964

コマンド構文検査 966

コマンド実行開始 967

コマンド実行完了 967

コマンド入力 964

コマンド入力行 964

状況域 965

情報域 968

端末の共用 975

ファンクション・キー値域 969

プログラム制御 974

変更方法 970

変数 971

本体 968

メッセージ行 969

ENQ コマンド 975

CECS トランザクション 970

CEDF トランザクション 927, 928

一時記憶域のブラウズ 948

画面の重ね書き 946

疑似会話型プログラム 943

起動 928

機能 927

機能 (PF) キーによるオプション 948

CEDF トランザクション (続き)

実行の修正 946

セキュリティ 929

単一画面モード 941

端末を使用しないトランザクション
944

二重画面モード 943

表示 931

表示レジスター 949

ファンクション・キー 931

プログラム・ラベル 946

ヘッダー 932

本体 932

ユーザー・タスクの異常終了 948

リモート・トランザクション 943

リモート・リンクされているプログラ
ム 944

CECI の呼び出し 949

DPL 944

EDF トランザクション 928

CEEBINT、言語環境プログラム HLL ユ

ーザー出口 692

CEEBXITA ユーザー出口 690

CEECSTX ユーザー出口 690

CEEDOPT CSECT 687

CEEENTRY マクロ 584

CEEHDLR サービス 682

CEEROPT CSECT 687

CEEUOPT CSECT 687

CEEWUCHA サンプル・ユーザー条件処
理ルーチン 660, 695

CESF、GOODNIGHT トランザクション
332

CHANNEL

オプション 119, 162, 167

LINK コマンド 162

RETURN コマンド 162

XCTL コマンド 162

char*

C++ 例外およびファウンデーション・
クラスにおける 641

CICS

テスト環境 5

プラットフォームの相違における 645

CICS (AIX 版)

プラットフォームの相違における 645

cics bt 1076

CICS Build Toolkit 1076

CICS Db2 環境

準備 1055

テスト 1054

CICS Explorer SDK

Java アプリケーションの開発 705

CICS Service の使用

一時記憶域の例 631

一時データ管理の例 629

CICS Service の使用 (続き)

開始データへのアクセス 625

キューの削除 629

項目の書き込み 631

項目の更新 631

項目の削除 631

項目の読み取り 631

端末からのデータの受信 633

端末管理の例 633

端末に関する情報の検索 633

端末へのデータの送信 633

データの書き込み 629

データの読み取り 629

トランザクション開始の例 625

トランザクションの開始 625

日時サービスの例 635

ファイル制御の例 620

満了していない開始要求のキャンセル
625

レコードの書き込み 618

レコードの更新 619

レコードの削除 620

レコードのブラウズ 620

レコードの読み取り 617

CICS 値データ域 99

CICS 域のアドレッシング 565

CICS オプション 888

CICS キー・ストレージ 360

CICS 条件

自動条件処理 643

自動条件処理 (callHandleEvent) 643

重大エラー処理 645

重大エラー処理 (abendTask) 645

手動条件処理 643

手動条件処理 (noAction) 643

条件、エラー、および例外における
642

例外処理 644

例外処理 (throwException) 644

abendTask 645

callHandleEvent 643

noAction 643

throwException 644

CICS ダンプ・ユーティリティ・プログ
ラム 242

CICS での Java プログラミング
データベースへのアクセス 820

JCICS の使用 715

インターフェース 716

エラーと例外 716

クラス 716

スレッド 718

直列化可能クラス 717

引数 717

JavaBeans 715

JCICS コマンド解説 720

CICS での Java プログラミング (続き)
 JCICS の使用 (続き)
 JCICS ライブラリー構造 716
 PrintWriter 718
 Task.err 718
 Task.out 718
CICS の定義
 リカバリーの 851
CICS の非 OSGi Java プロジェクトの配置 1053
CICS バンドル 19, 705, 922
CICS ファイル
 文書テンプレートとして 389
CICS プリンター 431
 特性の判別 439
CICS プログラム
 文書テンプレートとして 390
 DFHDHTL マクロ 390, 391
CICS 文書のコード・ページ 387
CICS 保守テーブル 282
CICS リソース 611
CICS リソースの使用 611
 ファウンデーション・クラスの概要における 611
 リソース・オブジェクトでのメソッドの呼び出し 613
 リソース・オブジェクトの作成 611
 singleton クラス 612
cicsbt 1076, 1077, 1079
cicsbt resolve 1071
CICSCondition
 C++ 例外およびファウンデーション・クラスにおける 642
CICSDATAKEY オプション 93, 364
CICSVAR 環境変数 691
CICS-MainClass 宣言のマニフェストへの追加 708
CLASS オプション 455
CLEAR
 キー 555
 PARTITION AID 値 555
 PARTITION キー 555
clear
 ポリモフィック動作における 649
 ポリモフィック動作の例における 649
CLEAR キー 120
CLOCK 594
cmmCICS
 ストレージ管理における 651
cmmDefault
 ストレージ管理における 651
cmmNonCICS
 ストレージ管理における 651
CMT 282
CNOTCOMPL オプション 327

COBOL 653
 アドレッシング、CICS データ域の 661
 グローバル変数 669
 コンパイラ・オプション 654
 作業用ストレージ 653
 サブプログラムの呼び出し 662, 665
 サポート 653
 参照修正 669
 実行単位 663
 制約事項 106, 654, 665, 669
 他の言語との混合 684
 デバッグ・モード 654
 ネストされたプログラム 674
 バッチ・コンパイル 672
 ブランク行 669
 プログラミング上の制約事項
 VS COBOL II 660
 プログラミング手法 654, 661, 665
 変換 669
 予約語テーブル 654
 31 ビット・アドレッシング 654
 ADDRESS 特殊レジスター 661
 CALL DL/I インターフェース 658
 CBLPSHPOP ランタイム・オプション 658
 DFHNCTR 呼び出しの例 424
 REPLACE ステートメント 669
COBOL のグローバル変数 669
COBOL のネストされたプログラム 674
COBOL プログラムのバッチ・コンパイル 672
COBOL2 オプション 888
COBOL3 オプション 888
CODEREG オペランド 580
COLUMN
 コマンド、CEBR トランザクション 960
COM アセンブラー命令 576
command-line 1076, 1094
COMMAREA 92, 93, 105
 オプション 119, 162, 165, 166
 チャネルおよびコンテナへのマイグレーション 158
 動的にルーティングされたアプリケーション 161
 LINK コマンド 158
 RETURN コマンド 160
 XCTL コマンド 159
 LINK コマンド 162
COMMAREA (通信域) 3, 858
COMMAREA の現代化 121
COMMAREA > 32K 120, 128, 147
Communications Server 313

condition
 手動条件処理 (noAction) における 643
 リソース・クラスにおける 609
CONNECT PROCESS コマンド 259
CONNECT コマンドのサンプル 1091
CONSISTENT オプション
 READ コマンド 292
CONVERSE コマンド 317, 330, 334
COPY ステートメント 903
CPI
 参照 2
CPI コミュニケーション・インターフェース・モジュール、DFHCPLC 903
CPI コミュニケーション・スタブ 262
CPI-C 245, 262
CPSM オプション 888
CQRY トランザクション 70
CSECT、マップのアセンブルへの追加 1039
CSNAP 594
CSPG トランザクション 443, 447, 518, 519
CSPP トランザクション 70
CSYSGRP 19
CTDLI 594
CTEST 594
CTLCHAR オプション 434, 435
CTRACE 594
cursor
 端末に関する情報の検索における 633
CURSOR オプション 497, 516
 ACCUM オプション 491
 SEND MAP コマンド
 ACCUM オプション 491
cut
 IccBuf コンストラクターにおける 615
CVDA 99, 244
CWA 114
CWAKEY パラメーター 115
C++ 例外 639
C++ 例外およびファウンデーション・クラス
 条件、エラー、および例外における 639

D

dataAreaOwner
 データ域の所有権における 613
dataAreaType
 データ域の拡張性における 614
dataItems
 ポリモフィック動作の例における 649
DATAONLY オプション 491, 493, 565

DATAREG オペランド 580
dateSeparator (パラメーター)
日時サービスの例における 635
dayOfMonth
日時サービスの例における 636
dayOfWeek
日時サービスの例における 636
daysSince1900
日時サービスの例における 636
DBCS オプション 888
DBRM の作成 1057
DCLGEN 操作 1063
DD 名リスト、変換プログラムの動的起
動における 898
DDS 483
DEBKEY オプション 299
DEBREC オプション 280, 299
DEBUG オプション 888
DEFRESP オプション 334
端末管理 328
delay
サポート・クラスにおける 611
DELAY コマンド 352, 353
delete
オブジェクトの削除における 606
ストレージ管理における 650, 651
DELETE ステートメント、COBOL 654
deleteLockedRecord 620
ロックされたレコードの削除における
620
DELETEQ TD コマンド 375
DELETEQ TS コマンド 379
deleteRecord
通常レコードの削除における 620
deleteRecord メソッド 620
DEPLOY APPLICATION コマンド 1084
DEPLOY APPLICATION コマンドのサ
ンプル 1091
DEPLOY BUNDLE コマンド 1080
DEPLOY BUNDLE コマンドのサンプル
1091
DEQ コマンド 355
DEQUEUE コマンド 443
DESTCOUNT オプション 542
DESTID オプション 333
DESTIDLENG オプション 333
DFH3QSS 5
DFHAID 594
DFHAPXPO 687
DFHASMVS プロシージャ 1033, 1035,
1040
DFHBMSCA 491, 594
DFHBMSCA 定義 509
DFHBMSUP 475
DFHCOMMAREA 165, 654
DFHCPLC 262

DFHCPLC、CPI コミュニケーション・イ
ンターフェース・モジュール 903
DFHCPLRR、SAA リソース・リカバリ
ー・インターフェース・モジュール 903
DFHDHTL マクロ 390, 391
DFHDHTXD 393
DFHDHTXH 393
DFHDHTXL 393
DFHDHTXO 393
dfhdploy 1077, 1079, 1091, 1094, 1096
DFHDPLOY 定義ユーティリティー
コマンド
DEPLOY APPLICATION 1084
DEPLOY BUNDLE 1080
PERFORM 1090
SET 1083, 1087, 1088
SET CICSplex 1079
UNDEPLOY APPLICATION 1085
UNDEPLOY BUNDLE 1081
DFHDPLOY の例 1091
DFHDPLOY ユーティリティー 1094
DFHDYPDS 158
DFHEAG インターフェース・モジュ
ール、アセンブラの 915
DFHEAG0 インターフェース・モジュ
ール、アセンブラの 915
DFHEAI インターフェース・モジュ
ール、アセンブラの 904
DFHEAI0 インターフェース・モジュ
ール、アセンブラの 904
DFHEAP1\$ 変換プログラム、アセンブ
ラーの 887
DFHECP1\$ 変換プログラム、COBOL の
887
DFHEDF グループ 1044
DFHEDP1\$、変換プログラム、C の 887
DFHEGTAL プロシージャ 1011, 1014
DFHEIBLK 654
DFHEIEND マクロ 895, 896
DFHEIENT マクロ 587, 895, 896
デフォルト 580
CODEREG 580
DATAREG 580
EIBREG 580
DFHEIPLR 記号レジスター 580
DFHEIRET マクロ 587, 890, 894
DFHEISTG マクロ 895, 896
DFHEITAL プロシージャ 1011, 1014
DFHEIVAR 654
DFHELII 904
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 904
DFHEPP1\$ 変換プログラム、PL/I の
887
DFHEXEC 904

DFHEXTAL プロシージャ 1011, 1014
DFHFCT マクロ 282
DFHLNKVS プロシージャ 1033, 1040
DFHMAPS プロシージャ 1010, 1037
DFHMAPT
インストール用プロシージャ、
HTML テンプレートの 1038
DFHMDf マクロ 465, 466
表示特性 488
DSATTS オプション 488
MAPATTS オプション 488
DFHMDI マクロ 465, 468
DFHMIRS プログラム 251
DFHMSCAN ユーティリティー・プログ
ラム 991
DFHMSD マクロ 465, 469
BASE オプション 487
STORAGE オプション 487
DFHMSD、マップ・セットのアセンブル
用マクロ 1032
DFHMSRCA 559, 594
DFHNC001
デフォルトの名前付きカウンター・ブ
ール 411
DFHNCO マクロ
名前付きカウンター・オプション・テ
ーブル 411
DFHNCOPT
名前付きカウンター・オプション・テ
ーブル 411
DFHNCTR
ヌル・ポインターを持つ COBOL 呼
び出しの例 424
DFHPDI マクロ 550
DFHPEP プログラム 236
DFHPLT マクロ 852
DFHPSD マクロ 550, 1040
DFHRESP 変換プログラム機能 218, 886
DFHURLDS 537
DFHVALUE 886
DFHXLT マクロ 852
DFHYITDL プロシージャ 1011, 1023
DFHYITEL プロシージャ 1011, 1023
DFHYITPL プロシージャ 1011, 1021
DFHYITVL プロシージャ 1011, 1017
DFHYXTDL プロシージャ 1011, 1023
DFHYXTEL プロシージャ 1011, 1023
DFHYXTPL プロシージャ 1011, 1021
DFHYXTVL プロシージャ 1011, 1017
DFHZITCL プロシージャ 1011, 1017
DFHZITPL プロシージャ 1011, 1021
DIB 886
DISCONNECT コマンドのサンプル 1091
DISPLAY ステートメント、COBOL 654
DLI オプション 888
DLL 687

DL/I
 暗黙エンキュー 873
 参照 2
 スケジューリング
 プログラム分離スケジューリング 873
 同期点 216
 トランザクション間通信 859
DOCTEMPLATE リソース定義 380, 387
 EXITPGM 属性 392
 FILE 属性 389
 HFSFILE 属性 388
 MEMBERNAME 属性 388
 PROGRAM 属性 390
 TDQUEUE 属性 389
 TSQUEUE 属性 389
DOCTOKEN 397, 409
DOCUMENT CREATE コマンド 380, 397
 DELIMITER オプション 399, 400
 DOCSIZE オプション 406
 DOCTOKEN 397
 LISTLENGTH オプション 399
 SYMBOLLIST オプション 380, 382, 397, 399, 400
DOCUMENT DELETE コマンド 409
 DOCTOKEN 409
DOCUMENT INSERT コマンド 380, 403, 405, 406
DOCUMENT RETRIEVE コマンド 380, 406
 CLNTCODEPAGE オプション 387, 406
 HOSTCODEPAGE オプション 387
DOCUMENT SET コマンド 380
 DELIMITER オプション 399, 400
 LISTLENGTH オプション 399
 SYMBOL オプション 380, 382, 399, 400
 SYMBOLLIST オプション 380, 382, 399, 400
doSomething
 オブジェクトの使用における 605
DPL 162, 216, 245, 248, 944
DSA 104
DSATTS オプション 488
DSNTIAR 1059
DTP 245, 262
DUMP TRANSACTION コマンド 242
DUPKEY 状態 296

E

EAR ファイル 1050
ECBLIST 355
ECI 789

EDF 886, 890, 927
EDF オプション 888
EIB 217, 314, 886
 説明 5
 端末管理フィードバック 325
 EIBCALEN フィールド 166
 EIBCOMPL フィールド 317
 EIBFN フィールド 167
EIBREG オペランド 580
empty
 一時記憶域における 630
 一時データにおける 628
 キューの削除における 629
 項目の削除における 631
ENDBR コマンド 295
endInsert
 レコードの書き込みにおける 618
endl
 端末管理の例における 634
ENQ コマンド 111, 355
ENQBUSY 状態 111
ENQUEUE コマンド 443
EOC 状態 326, 513
EODI 文字 331
EODS 状態 326
EPILOG オプション 888
EQUAL オプション 292
erase
 端末管理の例における 634
 端末へのデータの送信における 633
ERASE オプション 435, 492, 516
ERASEAUP オプション 492, 516, 529
ERDSA 1005
ESDS
 拡張アドレス方式へのアップグレード 272
 ファイル制御における 617
ESDS (入力順データ・セット) 265
ESDS ファイル 617
ESDS レコードの書き込み
 ファイル制御における 619
 レコードの書き込みにおける 619
ESDS レコードの読み取り
 ファイル制御における 618
 レコードの読み取りにおける 618
EXCI 890
 オプション 888
 通信 263
 CALL 263
EXCI - 外部呼び出しインターフェース 245
EXEC インターフェース・スタブ 904, 915
EXEC インターフェース・ブロック 886
EXEC インターフェース・モジュール 903, 904, 915, 1026

EXECKEY 93, 115
EXECKEY パラメーター 360
EXPLAIN 1065
EYUVALUE 886

F

familyConformanceError
 C++ 例外およびファウンデーション・クラスにおける 642
FEPI
 参照 2
FEPI - フロントエンド・プログラミン
 グ・インターフェース 245
FETCH 594
FETCHABLE オプション 697
file (パラメーター)
 ファイル制御の例における 621
FIND コマンド、CEBR トランザクシ
 ョン 961
FLAG オプション 888
FLOAT コンパイラー・オプション 693
flush
 端末管理の例における 634
FMH 328
 アウトバウンド 329
 インバウンド 328
 オプション 329
FMHPARM オプション 516
FOR オプション 353
Form
 ポリモアフィック動作における 648
format (パラメーター)
 日時サービスの例における 635
FORMATTIME コマンド 352
FORMFEED オプション 437, 516
FREE コマンド 330
FREEKB オプション 492
freeKeyboard
 端末へのデータの送信における 633
FREEMAIN コマンド 359
FROM オプション 491
FRSET オプション 492
fsAllowPlatformVariance
 プラットフォームの相違における 645
fsEnforce
 プラットフォームの相違における 645

G

GDDM 494
GDS オプション 888
GENERIC オプション 270, 292
get
 ポリモアフィック動作における 649

get (続き)
ポリモアフィック動作の例における
650
GET コマンド、CEBR トランザクション
961
GETMAIN コマンド 93
CICS DATAKEY オプション 93, 364
INITIMG オプション 106, 359
NOSUSPEND オプション 359
SHARED オプション 92, 98, 359
TASK DATAKEY オプション 93
USER DATAKEY オプション 93, 364
GETMAIN64 コマンド
SHARED オプション 98
GOODNIGHT トランザクション、
CESF 332
GRANT コマンド 1065
GRAPHIC オプション 888
GRPNAME オプション 476
GTEQ オプション 270, 292

H

HANDLE ABEND 228
HANDLE ABEND LABEL の制約事項、
アセンブラ言語での 576
HANDLE ABEND コマンド 222, 236,
864, 865
HANDLE AID コマンド 508
HANDLE CONDITION ERROR コマン
ド 226
HANDLE CONDITION コマンド 222,
229, 863
handleEvent
自動条件処理 (callHandleEvent) に
おける 643
HFS ファイル
文書テンプレートとして 388
HOLD オプション 162
HONEOM オプション 435
HTML テンプレート
インストール 1038

I

IBM 提供のクラス
ユーザー定義リソースの例 245
IBM 表示画面定義機能 II (SDF II) 1029
IBM WRLKC リンケージ・エディターの
入力 695
Icc
ファウンデーション・クラスの概要に
おける 606
メソッド・レベルにおける 646

IccAbendData
singleton クラスにおける 612
IccAbsTime
基本クラスにおける 607
サポート・クラスにおける 610
日時サービスにおける 635
IccBase
基本クラスにおける 606, 607
サポート・クラスにおける 610
ストレージ管理における 650, 651
リソース識別クラスにおける 607
リソース・クラスにおける 609
IccBase クラス
概要 606
IccBuf
一時記憶域の例における 632
一時データ管理の例における 630
項目の読み取りにおける 631
サポート・クラスにおける 611
端末管理の例における 634
データ域の拡張性における 614
データ域の所有権における 613
データの読み取りにおける 629
トランザクション開始の例における
626, 627, 628
バッファ・オブジェクトにおける
613
ファイル制御の例における 621, 622
ポリモアフィック動作の例における
649
C++ 例外およびファウンデーション・
クラスにおける 641
IccBuf クラスにおける 613
IccBuf コンストラクターにおける
614, 615
IccBuf メソッドにおける 615
IccResource サブクラスの処理におけ
る 615, 616
「read」メソッドから返される IccBuf
参照内のデータの有効範囲内 652
IccBuf クラス
コンストラクター 614
データ域の拡張性 614
データ域の所有権 613
バッファ・オブジェクトにおける
613
メソッド 615
IccBuf コンストラクター 614
IccBuf メソッド 615
IccResource サブクラスの処理 615
IccBuf コンストラクター 614
バッファ・オブジェクトにおける
614
IccBuf クラスにおける 614
IccBuf 参照 652
IccBuf メソッド 615

IccBuf メソッド (続き)
バッファ・オブジェクトにおける
615
IccBuf クラスにおける 615
IccClock
日時サービスにおける 635
日時サービスの例における 635, 636
IccCondition
C++ 例外およびファウンデーション・
クラスにおける 642
IccConsole
オブジェクト・レベルにおける 645,
646
バッファ・オブジェクトにおける
613
singleton クラスにおける 612
IccConsole クラス
概要 612
IccControl
基本クラスにおける 607
サポート・クラスにおける 611
トランザクション開始の例における
626, 627
メソッド・レベルにおける 646
singleton クラスにおける 612
IccControl クラス
概要 607, 612
IccDataQueue
一時記憶域における 630
一時データ管理の例における 629, 630
一時データにおける 628, 629
データの書き込みにおける 629
バッファ・オブジェクトにおける
613
ポリモアフィック動作の例における
649
リソース・クラスにおける 609
IccResource サブクラスの処理におけ
る 616
IccDataQueueId
一時データ管理の例における 629
一時データにおける 628
IccEvent
サポート・クラスにおける 611
IccException
オブジェクト・レベルにおける 646
サポート・クラスにおける 611
パラメーター・レベルにおける 647
メソッド・レベルにおける 646
C++ 例外およびファウンデーション・
クラスにおける 641, 642
IccException クラス
CICSCondition タイプ 642
familyConformanceError タイプ 642
internalError タイプ 642
invalidArgument タイプ 641

IccException クラス (続き)
 invalidMethodCall タイプ 642
 objectCreationError タイプ 641

IccFile
 通常レコードの削除における 620
 バッファ・オブジェクトにおける 613
 ファイル制御における 616, 617
 ファイル制御の例における 620
 リソース識別クラスにおける 608
 レコードの書き込みにおける 618
 レコードの更新における 619
 レコードの参照における 620
 レコードの読み取りにおける 617
 ロックされたレコードの削除における 620
 C++ 例外およびファウンデーション・クラスにおける 642
 ESDS レコードの書き込みにおける 619
 ESDS レコードの読み取りにおける 618
 KSDS レコードの書き込みにおける 619
 KSDS レコードの読み取りにおける 617, 618
 RRDS レコードの書き込みにおける 619
 RRDS レコードの読み取りにおける 618
 singleton クラスにおける 612

IccFile クラス
 deleteLockedRecord 620
 deleteRecord メソッド 620
 isReadable メソッド 618
 keyLength メソッド 618
 keyPosition メソッド 618
 readRecord メソッド 617
 recordFormat メソッド 618
 recordIndex メソッド 618
 recordLength メソッド 618
 registerRecordIndex 618
 registerRecordIndex メソッド 617
 rewriteRecord メソッド 619
 writeRecord メソッド 618

IccFileId
 基本クラス 607
 ファイル制御における 616, 617
 リソース識別クラスにおける 607, 608

IccFileId クラス
 概要 607, 617
 読み取り、レコードの 616

IccFileIterator
 バッファ・オブジェクトにおける 613
 ファイル制御における 616

IccFileIterator (続き)
 ファイル制御の例における 620, 622
 レコードの参照における 620

IccFileIterator クラス
 概要 616
 readNextRecord メソッド 620
 readPreviousRecord 620

IccFile::readRecord
 「read」メソッドから返される IccBuf
 参照内のデータの有効範囲内 652

IccJournal
 オブジェクト・レベルにおける 645, 646
 バッファ・オブジェクトにおける 613

IccKey
 通常レコードの削除における 620
 ファイル制御における 616
 レコードの書き込みにおける 618
 レコードの参照における 620
 レコードの読み取りにおける 617
 KSDS レコードの書き込みにおける 619
 KSDS レコードの読み取りにおける 617

IccKey クラス 617
 読み取り、レコードの 616

IccMessage
 サポート・クラスにおける 611

IccProgram
 バッファ・オブジェクトにおける 613
 プログラム制御における 623
 リソース・クラスにおける 609

IccProgram クラス
 プログラム制御 623

IccProgramId
 リソース識別クラスにおける 608

IccRBA
 ファイル制御における 616
 レコードの書き込みにおける 618
 レコードの参照における 620
 レコードの読み取りにおける 617
 ESDS レコードの書き込みにおける 619
 ESDS レコードの読み取りにおける 618
 RRDS レコードの書き込みにおける 619

IccRBA クラス
 読み取り、レコードの 616

IccRecordIndex
 C++ 例外およびファウンデーション・クラスにおける 642

IccRequestId
 トランザクション開始の例における 626, 627
 パラメータ受け渡し規則内 651

IccResource
 基本クラスにおける 606, 607
 ポリモフィック動作における 648, 649
 ポリモフィック動作の例における 649
 リソース・クラスにおける 609
 「read」メソッドから返される IccBuf
 参照内のデータの有効範囲内 652

IccResource クラス
 概要 606, 607
 サブクラスの処理 615

IccResource サブクラスの処理
 バッファ・オブジェクトにおける 615
 IccBuf クラスにおける 615

IccResourceId
 基本クラスにおける 606, 607
 リソース識別クラスにおける 608
 C++ 例外およびファウンデーション・クラスにおける 641

IccResourceId クラス
 概要 606, 607

IccRRN
 通常レコードの削除における 620
 ファイル制御における 616
 レコードの書き込みにおける 618
 レコードの参照における 620
 レコードの読み取りにおける 617
 RRDS レコードの読み取りにおける 618

IccRRN クラス
 読み取り、レコードの 616

IccSession
 バッファ・オブジェクトにおける 613

IccStartRequestQ
 開始データへのアクセスにおける 625
 トランザクション開始の例における 626, 627, 628
 トランザクションの非同期の開始における 624
 バッファ・オブジェクトにおける 613
 パラメータ受け渡し規則内 651
 singleton クラスにおける 612

IccStartRequestQ クラス
 概要 612

IccSysId
 プログラム制御における 623

IccSystem
 singleton クラスにおける 612

IccSystem クラス
概要 612

IccTask
サポート・クラスにおける 611
トランザクション開始の例における 627
パラメーター・レベルにおける 646
C++ 例外およびファウンデーション・クラスにおける 641
singleton クラスにおける 612

IccTask クラス
概要 612

IccTask::commitUOW
「read」メソッドから返される IccBuf
参照内のデータの有効範囲内 652

IccTempStore
一時記憶域における 630
一時記憶域の例における 631, 632
一時データにおける 629
項目の書き込みにおける 631
項目の更新における 631
項目の削除における 631
項目の読み取りにおける 631
自動条件処理 (callHandleEvent) における 643
バッファ・オブジェクトにおける 613
ポリモアフィック動作の例における 649
リソース・クラスにおける 609
C++ 例外およびファウンデーション・クラスにおける 641

IccResource サブクラスの処理における 615, 616

IccTempstore
IccResource サブクラスの処理における 615

IccTempStoreId
一時記憶域における 630
一時記憶域の例における 631, 632
基本クラスにおける 607

IccTempStore::readItem
「read」メソッドから返される IccBuf
参照内のデータの有効範囲内 652

IccTempStore::readNextItem
「read」メソッドから返される IccBuf
参照内のデータの有効範囲内 652

IccTermId
基本クラスにおける 606
端末管理における 633
端末管理の例における 633
端末に関する情報の検索における 633
626

C++ 例外およびファウンデーション・クラスにおける 641, 642

IccTermId クラス
概要 606

IccTerminal
端末からのデータの受信における 633
端末管理における 633
端末管理の例における 633
端末に関する情報の検索における 633
バッファ・オブジェクトにおける 613
リソース・クラスにおける 608, 609
singleton クラスにおける 612

IccTerminalData
端末管理における 633
端末管理の例における 633
端末に関する情報の検索における 633

IccTerminal::receive
「read」メソッドから返される IccBuf
参照内のデータの有効範囲内 652

IccTime
基本クラスにおける 607
サポート・クラスにおける 610
パラメーター受け渡し規則内 651

IccTime クラス
概要 607

IccTimeInterval
基本クラスにおける 607
サポート・クラスにおける 610
トランザクション開始の例における 626, 627

IccTimeOfDay
基本クラスにおける 607
サポート・クラスにおける 610

IccTransId
基本クラスにおける 606
トランザクション開始の例における 626
パラメーター受け渡し規則内 651

IccTransId クラス
概要 606

IccUserControl
一時記憶域の例における 632
一時データ管理の例における 629
端末管理の例における 634
トランザクション開始の例における 626
日時サービスの例における 635
ファイル制御の例における 621
プログラム制御における 623
ポリモアフィック動作の例における 649
C++ 例外およびファウンデーション・クラスにおける 640
singleton クラスにおける 612

Icc::initializeEnvironment
ストレージ管理における 650

ICTL (入力形式制御) アセンブラー命令 576

ICVR パラメーター 1044

Id
リソース識別クラスにおける 607

IGNORE CONDITION コマンド 222, 227

IGREQID 状態 516

IGYCCICS 654

IGZWRLKA 660

IMMEDIATE オプション 166, 314, 329

IMS.RESLIB (IMS ライブラリー) 1026

INBFMH 状態 326

initializeEnvironment
ストレージ管理における 650, 651
メソッド・レベルにおける 646

INITIMG オプション 106, 359

INPUTMSG オプション 162, 167

INQUIRE TERMINAL コマンド 325, 439

INQUIRE コマンド 4

INRTN オプション 553

insert
一時記憶域の例における 632
IccBuf コンストラクターにおける 615

instance
singleton クラスにおける 612

internalError
C++ 例外およびファウンデーション・クラスにおける 642

INTERVAL オプション 353

invalidArgument
C++ 例外およびファウンデーション・クラスにおける 641

invalidMethodCall
C++ 例外およびファウンデーション・クラスにおける 642

INVITE オプション 314

INVOKE APPLICATION 923

INVPARTN 状態 555

INVPARTNSET 状態 555

INVREQ 条件 516

IOERR 条件の処理 867

IPCS 242

IPIC 接続 796, 797, 799

IRC 263

isAddable
ESDS レコードの書き込みにおける 619
KSDS レコードの書き込みにおける 619
RRDS レコードの書き込みにおける 619

iscics 5

ISCINVREQ 247

ISR2
 トランザクション開始の例における
 626
isReadable
 ESDS レコードの読み取りにおける
 618
 KSDS レコードの読み取りにおける
 618
 RRDS レコードの読み取りにおける
 618
isReadable メソッド 618
ISSUE ABORT コマンド 333
ISSUE ADD コマンド 332
ISSUE COPY コマンド 316, 447
ISSUE DISCONNECT コマンド 316
ISSUE END コマンド 333
ISSUE ERASE コマンド 316, 333
ISSUE NOTE コマンド 333
ISSUE PRINT コマンド 447
ISSUE QUERY コマンド 332
ISSUE RECEIVE コマンド 332
ISSUE REPLACE コマンド 333
ISSUE SEND コマンド 332
ISSUE WAIT コマンド 333, 334
ITMP
 トランザクション開始の例における
 626

J

Java EE アプリケーションへのアクセス
 の制御 757
Java Message Service 779
Java アプリケーションの開発 705
Java アプリケーションの接続性 838
Java アプリケーションのデプロイ 705
Java でのプログラミング 715
Java の開発
 CICS Explorer SDK 705
javadoc 839
JCA 796, 797, 799, 800, 839
 チャンネル 792
 トレース 794
 リソース・アダプター 787, 794, 795
 CCI 785, 787, 788, 793, 794, 795
 ECI 788, 792, 793, 794
JCAServlet 797, 799
JCICS
 異常終了 726
 一時記憶 742
 インターフェース 716
 エラー処理 726
 エラーと例外 716
 およびチャンネル 147
 クラス 716
 クラス・ライブラリー 715

JCICS (続き)
 現行チャンネルの受け取り 730
 現行チャンネルのブラウズ 730
 コマンド解説 720
 コンテナーからのデータの取得 730
 コンテナーの作成 729
 条件処理 722
 診断サービス 731
 ストレージ・サービス 741
 スレッドとタスク 741
 スレッドの使用 718
 端末管理 743
 チャンネルとコンテナー 728
 チャンネルの作成 729
 直列化可能クラス 717
 引数 717
 ファイル制御 735
 プログラム制御 739
 プログラム例 731
 変換
 データから XML へ 743
 XML からデータへ 743
 ライブラリー構造 716
 リソース定義 716
 例外処理 721, 723
 例外マッピング 723
 ABEND 処理 721, 723
 ADDRESS 732
 APPC 727
 BMS 727
 CANCEL コマンド 740
 DEQ コマンド 741
 DOCUMENT サービス 731
 ENQ コマンド 741
 HANDLE コマンド 722
 HTTP サービス 738
 INQUIRE SYSTEM 734
 INQUIRE TASK 734
 INQUIRE TERMINAL または
 NETNAME 735
 JavaBeans 715
 Javadoc 715
 JCICS Class Reference 715
 PrintWriter 718
 RETRIEVE コマンド 740
 START コマンド 740
 Task.err 718
 Task.out 718
 UOW 745, 768, 780
 Web サービス 746
JCICS エンコード 719
JCICS からのチャンネルの使用 147
JCICS を使用した Java の開発
 概要 715
JES 4, 449

JES (ジョブ入力サブシステム)
 スプーラー・コマンド 451
 出口 454
 入力 454
 JES スプールからのデータの検索 449
 RESP および RESP2 オプション 451
JMS 779
JMS クライアント 779
JOURNALNAME 215
JOURNALNUM 215
JTYPEID 215
JUSFIRST オプション 529
JUSLAST オプション 529
JUSTIFY オプション 529
JVM サーバー 703
 デプロイ先 1044
 ベスト・プラクティス 714, 748
 OSGi サービス 1052
 OSGi バンドルのインストール 1045
 WAR ファイルのデプロイ 1050

K

key (パラメーター)
 ファイル制御の例における 621
keyLength
 KSDS レコードの書き込みにおける
 619
 KSDS レコードの読み取りにおける
 618
keyLength メソッド 618
keyPosition
 KSDS レコードの書き込みにおける
 619
 KSDS レコードの読み取りにおける
 618
keyPosition メソッド 618
KSDS
 ファイル制御における 617
 KSDS (キー順データ・セット) 265
 KSDS ファイル 617
 KSDS レコードの書き込み
 ファイル制御における 619
 レコードの書き込みにおける 619
 KSDS レコードの読み取り
 ファイル制御における 617
 レコードの読み取りにおける 617

L

L8 および L9 モード TCB 86, 87
 制約事項 89
Language Environment 679
LAST オプション 329, 492
 ブラケット・プロトコル 329

LDC 556
LDCMNEM オプション 556
LDCNUM オプション 556
LEASM オプション 888
LENGERR 条件 317
LENGTH オプション 99, 317, 888
LENGTHLIST オプション
タスク関連ストレージ域の複数のダン
プ 243
Liberty 751, 1050
Liberty JVM サーバー 1050
LIBRARY 991
line
端末に関する情報の検索における 633
LINE コマンド
CEBR トランザクション 961
LINECOUNT オプション 888
LINK
チャネルおよびコンテナーへのマイグ
レーション 158
LINK PROGRAM 263
LINK コマンド 93, 103, 104, 163
CHANNEL オプション 162, 167
COMMAREA オプション 162, 165,
166
IMMEDIATE オプション 166
INPUTMSG オプション 162, 167
TRANSID オプション 166
LINKAGE オプション 888
LIST オプション 534
LLA (ライブラリー・ルックアサイド機
能) 1004
LOAD コマンド
HOLD オプション 162
LOCKED 類縁性の関係 207
LPA 1004
LU タイプ 4
装置 315
バッチ・データ交換 333
論理レコードの提示 327
LU (論理装置)
機能 326

M

main
一時記憶域の例における 631
一時データ管理の例における 629
ストレージ管理における 650, 651
端末管理の例における 634
トランザクション開始の例における
626
日時サービスの例における 635
ファイル制御の例における 621
プログラム制御における 623
ヘッダー・ファイル内 638

main (続き)
ポリモアフィック動作の例における
649
C++ 例外およびファウンデーション・
クラスにおける 640
MAPATTS オプション 488
MAPCOLUMN オプション 525
MAPFAIL 状態 506, 513
MAPHEIGHT オプション 525
MAPLINE オプション 525
MAPONLY オプション 491, 493, 565
MAPPED オプション 533
MAPSET オプション 491
MAPSET リソース定義 473
MAPWIDTH オプション 525
MARGINS オプション 888
MASSINSERT オプション 306, 309
MDT 507, 563
MERGE ステートメント、COBOL 654
MGMTPART 51
MOM 779
MONITOR POINT コマンド 242
MONITOR コマンド 242
monthOfYear
日時サービスの例における 636
MSGINTEG オプション 334
MSR 559
MSR オプション 516, 559
MVS サブスペース 373
MVS トランザクション 169
MVS/ESA
ストレージ管理における 651
MXT パラメーター 109
MyTempStore
自動条件処理 (callHandleEvent) にお
ける 644

N

NATLANG オプション 888
new
ストレージ管理における 650, 651
NLEOM オプション 433, 434, 436, 516
noAction
CICS 条件における 642
NOAUTOPAGE オプション 519
NOBLCARD オプション 888
NOCPSM オプション 888
NODE オプション 452
NODEBUG オプション 888
Node.js
インストールの検査 1098
NOEDF オプション 888
NOEDIT オプション 533
NOEPILOG オプション 888

NOEPILOG 変換プログラム・オプション
580
NOFEPI オプション 888
NOFLUSH オプション 522, 529
NOHANDLE オプション 218, 225
NOJBUFSP 状態 111
NOLENGTH オプション 888
NOLINKAGE オプション 888
NONUM オプション 888
NOOPSEQUENCE オプション 888
NOOPTIONS オプション 888
NOPROLOG オプション 888
NOPROLOG 変換プログラム・オプショ
ン 580
NOQUEUE オプション 111
NOSEQ オプション 888
NOSEQUENCE オプション 888, 899
NOSOURCE オプション 888
NOSPACE 状態 226
NOSPIE オプション 888
NOSUSPEND オプション 111
GETMAIN コマンド 359
READ コマンド 292
READNEXT コマンド 304
READPREV コマンド 304
WRITE コマンド 306
NOTRUNCATE オプション 317
NOVBREF オプション 888
NOWAIT オプション 334
NOXREF オプション 888
NUM オプション 888
number
RRDS レコードの書き込みにおける
619
NUMREC オプション 304
NUMSEGMENTS オプション
タスク関連ストレージ域の複数のダン
プ 243

O

obj (パラメーター)
オブジェクトの使用における 605
objectCreationError
C++ 例外およびファウンデーション・
クラスにおける 641
OCCURS オプション 477
ON ユニット 868
OO COBOL
サポート 653
OPCLASS オプション 534
OPEN ステートメント、COBOL 654
OPENAPI 87
制約事項 89
operator=
ファイル制御の例における 622

operator= (続き)
 IccResource サブクラスの処理にお
 ける 615, 616
operator«
 IccResource サブクラスの処理にお
 ける 616
OPID オプション 534
OPIDENT 値 534
OPMARGINS オプション 888
OPSEQUENCE オプション 888
OPSYN (命令コード等価) アセンブラー
 命令 576
OPTIONS オプション 888
OPTIONS(MAIN) の指定 693
OSGi 748
OSGi サービス
 呼び出し 1052
OSGi バンドル 705
 インストール 1045
OSGi バンドルのデプロイ 1045
OTE、オープン・トランザクション環境
 78
OVERFLOW 状態 522

P

PA キー 439
PAGENUM オプション 542
PAGESIZE 値 521
PAGING オプション 434, 491, 498
PARTITIONSET オプション 552
PARTN オプション 553
PARTNFAIL 状態 555
PARTNPAGE オプション 554
PERFORM DUMP コマンド 242
PERFORM コマンド 4, 1090
PF (プログラム機能) キー 958
PLATDIR 19
PLATHOME 19
PLT プログラム 364
PLT (プログラム・リスト・テーブル)
 定義 852
plugin-cfg 806
PL/I 693
 言語環境プログラムの要件 695
 制約事項 694
 他の言語との混合 684
 取り出されるプロシージャー 697
 プログラミング手法 693, 694, 695,
 697
 プログラムとエラー処理
 ON ユニット 868
 FLOAT コンパイラー・オプション
 693
 OPTIONS(MAIN) の指定 693
POP HANDLE コマンド 222, 229, 236

POST コマンド 352, 353
PRGDLY オプション 540
print
 ポリモフィック動作における 648
PRINTERCOMP オプション 437
PROGRAM オプション 236
PROGRAM リソース定義 473
programId
 メソッド・レベルにおける 646
PROLOG オプション 888
PROTECT オプション 334
PURGE MESSAGE コマンド 517, 534
PURGE コマンド、CEBR トランザクシ
 ョン 962
PUSH HANDLE コマンド 222, 229, 236
put
 ポリモフィック動作における 649
 ポリモフィック動作の例における
 650
PUT コマンド、CEBR トランザクシ
 ョン 962

Q

QBUSY 状態 111
QUERY SECURITY コマンド 243
 NOLOG オプション 244
QUEUE コマンド、CEBR トランザクシ
 ョン 962
queueName
 開始データへのアクセスにおける 625
QUOTE オプション 888
QZERO 条件 377

R

RACF 243
RBA 617
RBA (相対バイト・アドレス) 265, 271
RDF 309
READ コマンド 301
 CONSISTENT オプション 292
 NOSUSPEND オプション 292
 REPEATABLE オプション 292
 UNCOMMITTED オプション 292
READ ステートメント、COBOL 654
readItem
 一時記憶域における 630
 一時記憶域の例における 632
 一時データにおける 628
 項目の読み取りにおける 631
 データの読み取りにおける 629
 IccResource サブクラスの処理にお
 ける 616

readItem (続き)
 「read」メソッドから返される IccBuf
 参照内のデータの有効範囲内 652
READNEXT コマンド 295
 NOSUSPEND オプション 304
readNextItem
 一時記憶域における 630
 「read」メソッドから返される IccBuf
 参照内のデータの有効範囲内 652
readNextRecord
 レコードの参照における 620
readNextRecord メソッド 620
READPREV コマンド 295
 NOSUSPEND オプション 304
readPreviousRecord 620
 レコードの参照における 620
READQ TD コマンド 111, 375
READQ TS コマンド 379
 ITEM オプション 379
readRecord
 レコードの更新における 619
 レコードの読み取りにおける 617
 ロックされたレコードの削除における
 620
 C++ 例外およびファウンデーション・
 クラスにおける 642
readRecord メソッド 617
 「read」メソッドから返される IccBuf 参
 照内のデータの有効範囲
 その他 652
receive
 端末からのデータの受信における 633
RECEIVE MAP コマンド 505
 ASIS オプション 507
RECEIVE PARTN コマンド 554
RECEIVE コマンド 314, 316, 317, 330,
 334
 MAPFAIL 状態 513
receive3270data
 端末からのデータの受信における 633
recordFormat
 ESDS レコードの書き込みにおける
 619
 ESDS レコードの読み取りにおける
 618
 RRDS レコードの書き込みにおける
 619
 RRDS レコードの読み取りにおける
 618
recordFormat メソッド 618
recordIndex
 ESDS レコードの書き込みにおける
 619
 ESDS レコードの読み取りにおける
 618

recordIndex (続き)
KSDS レコードの書き込みにおける 619
KSDS レコードの読み取りにおける 618
RRDS レコードの書き込みにおける 619
RRDS レコードの読み取りにおける 618
recordIndex メソッド 618
recordLength
ESDS レコードの書き込みにおける 619
ESDS レコードの読み取りにおける 618
KSDS レコードの書き込みにおける 619
KSDS レコードの読み取りにおける 618
RRDS レコードの書き込みにおける 619
RRDS レコードの読み取りにおける 618
recordLength メソッド 618
REGIONTYPE 19
registerData
トランザクション開始の例における 627
トランザクションの開始における 625
registerRecordIndex 618
レコードの書き込みにおける 618
ESDS レコードの書き込みにおける 619
ESDS レコードの読み取りにおける 618
KSDS レコードの書き込みにおける 619
KSDS レコードの読み取りにおける 617
RRDS レコードの書き込みにおける 619
RRDS レコードの読み取りにおける 618
registerRecordIndex メソッド 617
RELEASE 594
RELEASE オプション 518
RELEASE コマンド
HOLD オプション 162
RELTYPE キーワード 278
REMOTENAME オプション 251
REMOTESYSTEM オプション 251
remoteTermId
トランザクション開始の例における 626
RENT 属性 1005

REPEATABLE オプション
READ コマンド 292
replace
IccBuf コンストラクターにおける 615
REPLACE ステートメント 669
req
トランザクション開始の例における 627
req1
トランザクション開始の例における 626
req2
トランザクション開始の例における 626
REQID オプション 296, 352, 516, 541
REQUIRED 86
reset
レコードの参照における 620
RESETBR コマンド 295
RESP 値 217
RESP オプション 218, 219, 225, 863
NOHANDLE の非アクティブ化 223
RESP および RESP2 オプション
JES に対するインターフェース 451
RESP2 値 217, 218
RESP2 オプション 218
RETAIN オプション 1061
RETPAGE 状態 500, 529
RETRIEVE コマンド 352, 357
retrieveData
開始データへのアクセスにおける 625
RETURN
チャンネルおよびコンテナへのマイグレーション 160
RETURN コマンド 93, 167
CHANNEL オプション 119, 162
COMMAREA オプション 119, 162
IMMEDIATE オプション 314, 329
INPUTMSG オプション 162, 167
returnTermId
開始データへのアクセスにおける 625
returnTransId
開始データへのアクセスにおける 625
REWRITE コマンド 301
REWRITE ステートメント、COBOL 654
rewriteItem
一時記憶域における 630
一時記憶域の例における 632
項目の書き込みにおける 631
項目の更新における 631
rewriteRecord
レコードの更新における 619
rewriteRecord メソッド 619
RIDFLD オプション 278, 295
RMODE コンパイラ・オプション 654

RMODE (常駐モード)
オプション、CICS アプリケーションの 1002
ROLLBACK
使用に関する考慮事項 864
ROUTE コマンド 534
ページ・オーバーフロー 542
LIST オプション 534
TITLE オプション 541
ROUTEDMSGS オプション 534
RPTOPTS 687
RRDS (相対レコード・データ・セット) 265
RRDS ファイル
ファイル制御における 617
RRDS レコードの書き込み
ファイル制御における 619
レコードの書き込みにおける 619
RRDS レコードの読み取り
ファイル制御における 618
レコードの読み取りにおける 618
RRN 617
RRN (相対レコード番号) 265, 271
RTEFAIL 状態 537
RTESOME 状態 537
RU (要求応答単位) 326
run
一時記憶域の例における 632
一時データ管理の例における 629, 630
基本クラスにおける 607
端末管理の例における 634, 635
トランザクション開始の例における 626
日時サービスの例における 635, 636
ファイル制御の例における 621, 622
プログラム制御における 623
ポリモアフィック動作の例における 649
C++ 例外およびファウンデーション・クラスにおける 640
RUWAPPOOL 683
RVI 316

S

SAA リソース・リカバリー 215
SAA リソース・リカバリー・インターフェース 855
SAA リソース・リカバリー・インターフェース・モジュール、DFHCPLRR 903
SAM 313
SCS
プリンター 436
SCS 入力 439
SDF II 465, 474
SDF II (IBM 表示画面定義機能 II) 1029

SEGMENTLIST オプション
タスク関連ストレージ域の複数のダン
プ 243

send
端末管理の例における 634

SEND CONTROL コマンド 431, 493

SEND MAP コマンド 431, 485

ACCUM オプション 515

ALARM オプション 492

CURSOR オプション 491, 497

DATAONLY オプション 491

ERASE オプション 492

ERASEAUP オプション 492, 529

FREEKB オプション 492

FROM オプション 491

LAST オプション 492

MAPONLY オプション 491

MAPSET オプション 491

NOFLUSH オプション 522, 529

PAGING オプション 491, 498

SET
コマンド 498

SET オプション 491

TERMINAL オプション 491, 498

WAIT オプション 492

SEND PAGE コマンド 216, 517, 534

AUTOPAGE オプション 519

NOAUTOPAGE オプション 519

RELEASE オプション 518

SEND PARTNSET コマンド 552

SEND TEXT コマンド 431, 528

MAPPED オプション 533

NOEDIT オプション 533

SEND コマンド 316, 317, 330, 334

CNOTCOMPL オプション 327

CTLCHAR オプション 434

FMH オプション 329

INVITE オプション 314

LAST オプション 329

MSR オプション 559

sendLine
端末管理の例における 634

ファイル制御の例における 622

SEQ オプション 888

SEQUENCE オプション 888

SESSBUSY 状態 111

SET
コマンド 4

SET CICSplex コマンド 1079

SET DOCTEMPLATE NEWCOPY コマ
ンド 385

SET オプション 491, 498

SET コマンド 1083, 1087, 1088

setColor
端末管理の例における 634

setData
トランザクションの開始における 625

setHighlight
端末管理の例における 634

setKind
ファイル制御の例における 621

SETLOCALE 594

setQueueName
トランザクション開始の例における
627

トランザクションの開始における 625

setReturnTermId
トランザクション開始の例における
627

トランザクションの開始における 625

setReturnTransId
トランザクション開始の例における
627

トランザクションの開始における 625

setRouteOption
トランザクション開始の例における
627, 628

プログラム制御における 624

set...
端末へのデータの送信における 633

SHARED オプション 92, 98

GETMAIN コマンド 359

SHARED オプション 93

SIGNAL 条件 316

Singleton クラス
リソース・オブジェクトの作成におけ
る 612

CICS リソースの使用における 612

singleton クラス 612

SORT ステートメント、COBOL 654

SOURCE オプション 888

SP オプション 888

SPACE オプション 888

SPIE オプション 888

SPOLBUSY 状態 454

SPOOLCLOSE コマンド 449

SPOOLOPEN
例 457

SPOOLOPEN コマンド 444, 449

NODE オプション 452

TOKEN オプション 452

USERID オプション 452

SPOOLREAD コマンド 449

SPOOLWRITE コマンド 449

SQL
参照 2

SQL 戻りコード
-818 1061

SQLCA フォーマット設定ルーチン 1059

SSL 800

STAE オプション 868

start
トランザクション開始の例における
627

トランザクションの開始における 625

パラメーター受け渡し規則内 651

START TRANSID コマンド 867

START コマンド 352, 357, 441

START ステートメント、COBOL 654

START データ、チャネルを使用するマイ
グレーション 160

STARTBR コマンド 295

startRequestQ
トランザクション開始の例における
626, 627

startType
トランザクション開始の例における
627

STATIC オペランド 580

STATREG オペランド 580

STOP ステートメント、COBOL 654

STORAGE オプション 487

STORAGE ランタイム・オプション、言
語環境プログラム 654

struct、C/370 シンボリック記述マップ・
セット 1031

SUSPEND コマンド 355

SVC99 594

SYNCONRETURN オプション 251, 259

SYNCPOINT コマンド 215, 216, 517

ROLLBACK オプション 236

SYSEIB オプション 888

SYSID オプション 251

SYSID コマンド、CEBR トランザクシ
ョン 962

SYSIDERR 247

SYSIN 885

SYSARM、マップ・セットのアセンブル
用オペランド 1032

SYSPRINT 886

SYSPUNCH 886

SYSSTATE マクロ 580

SYSTEM 594

T

TASKDATAKEY オプション 92, 93, 363

TASKDATALOC オプション 92, 898

TCPIPService 796

TCTUA 118, 363

TCTUAKEY 118, 363

TCTUALOC 118

TERM オプション 483

TERMID 値 534

TERMINAL
オプション 498

TERMINAL オプション 498

TERMINAL コマンド、CEBR トランザクション 962
TEST コンパイラー・オプション 654
throw
 例外処理 (throwException) における 644
 C++ 例外およびファウンデーション・クラスにおける 639
throwException
 CICS 条件における 642
ti
 トランザクション開始の例における 626, 627
TIOATDL 値 533
TITLE オプション 541
TOKEN オプション 302, 452
TOP コマンド、CEBR トランザクション 963
traceRequests 799
TRANGRP 204
TRANISO 371
TRANSID オプション 166, 251
TRANSID オペランド
 使用 862
TRUNC コンパイラー・オプション 654
try
 例外処理 (throwException) における 644
 C++ 例外およびファウンデーション・クラスにおける 639, 640, 641
tryNumber
 C++ 例外およびファウンデーション・クラスにおける 640
TWA 92
TWSIZE オプション 92
type
 C++ 例外およびファウンデーション・クラスにおける 641
TYPE=DSECT アセンブル 473

U

ucd 1096
UMT 282
UNCOMMITTED オプション
 READ コマンド 292
UNDEPLOY APPLICATION コマンド 1085
UNDEPLOY APPLICATION コマンドのサンプル 1091
UNDEPLOY BUNDLE コマンド 1081
UNDEPLOY BUNDLE コマンドのサンプル 1091
UNIX
 ストレージ管理における 651
UNTIL オプション 353

UOW 215
 類縁性 207
UPDATE オプション 301
urbancode 1096
USERDATAKEY オプション 93, 364
USERID オプション 452, 455

V

VALIDATE 1062
VBREF オプション 888
VERSION キーワード 1064
VLF (仮想ルックアサイド機能) 1004
VOLUME オプション 333
VOLUMELENG オプション 333
VS COBOL II
 言語環境プログラムの呼び出し可能サービス 680
 サポート 653
 プログラミング 660
 Language Environment 660
 WORKING-STORAGE の制限 654
VSAM 617
 エンキュー 108
 データ・セット 309
 プロセッサ・オーバーヘッド 309
 MASSINSERT オプション 309
VSAM 排他制御 871

W

WAIT EVENT コマンド 189, 352
WAIT EXTERNAL コマンド 189, 355
WAIT JOURNALNUM コマンド
 ジャーナル出力との同期化 212
WAIT TERMINAL コマンド 316
WAIT オプション 213, 214, 316, 492
WAITCICS コマンド 189, 355
waitForAID
 端末管理の例における 634
WAR ファイル 1050
 インストール 1050
WAR ファイルのデプロイ 1050
Web サーバー 806
Web サーバー・プラグイン 806
WebSphere Developer Tools 797, 799
WebSphere MQ classes for Java
 OSGi JVM サーバー
 UOW のコミット 838
WebSphere MQ classes for JMS
 OSGi JVM サーバー
 プログラミング 833
WRITE JOURNALNAME コマンド 111, 213

WRITE JOURNALNUM コマンド 111, 213
 ジャーナル・レコードの作成 212
WRITE コマンド 306
 NOSUSPEND オプション 306
WRITE ステートメント、COBOL 654
writeItem
 一時記憶域における 630
 一時データにおける 628
 項目の書き込みにおける 631
 データの書き込みにおける 629
 リソース・オブジェクトでのメソッドの呼び出しにおける 613
 C++ 例外およびファウンデーション・クラスにおける 641
 IccResource サブクラスの処理における 616
WRITEQ TD コマンド 375
WRITEQ TS コマンド 379
writeRecord
 ファイル制御の例における 622
 レコードの書き込みにおける 618
 KSDS レコードの書き込みにおける 619
 RRDS レコードの書き込みにおける 619
writeRecord メソッド
 IccFile クラス 618
WRKAREA パラメーター 114

X

X8 および X9 TCB 603
XCTL
 チャネルおよびコンテナーへのマイグレーション 159
XCTL コマンド 93, 103, 104
 CHANNEL オプション 162, 167
 COMMAREA オプション 162, 165
 INPUTMSG オプション 162, 167
XLT (トランザクション・リスト・テーブル)
 定義 852
XOPTS キーワード 899
XPCFTCH 603
XPCREQ グローバル・ユーザー出口 248, 251
XPCTA 603
XPLink 602
 グローバル・ユーザー出口 603
 非 XPLink オブジェクト 603
 TCBs 603
XRBA (拡張相対バイト・アドレス) 271
XRCINIT グローバル・ユーザー出口 877
XREF オプション 888
XTC OUT 出口ルーチン 566

Z

z/OS Communications Server 313

[特殊文字]

#echo コマンド 394, 395

#include コマンド 395

#set コマンド 382, 394, 395

