IMS
バージョン **15.1.0**

アプリケーション・プログラミング

SC43-4281-00

IMS

バージョン **15.1.0**

アプリケーション・プログラミング

SC43-4281-00

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、 1079 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、IMS 15 (プログラム番号 5635-A06)、IMS Database Value Unit Edition V15.01.00 (プログラム番号 5655-DS5)、IMS Transaction Manager Value Unit Edition V15.01.00 (プログラム番号 5655-TM4)、および新しい版で明記されていない限り、以降のすべてのリソ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： SC27-6778-00
IMS
Version 15.1.0
Application Programming
(November 1, 2017 edition)

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

目次

本書について

これらのトピックでは、IMS™ データベースまたは IMS トランザクションにアクセスするアプリケーション・プログラムの作成に関するガイダンス情報を提供します。各トピックで、プログラム要件の収集と分析の方法、および IMS アプリケーション・プログラムの開発とデバッグの方法について説明します。さらに、さまざまなプログラミング言語を使用して DL/I 呼び出しを発行する方法についても説明し、SQL および Java™ 開発の IMS ソリューションに関する情報も記載しています。また、さまざまなプログラミング言語を使用して EXEC DL/I 呼び出しを発行する方法についても説明しています。アプリケーション・プログラミング・インターフェース (API) 情報は、「*IMS V15* アプリケーション・プログラミング *API*」に記載されています。

この情報は、IBM® Knowledge Center で参照できます。

前提知識

本書は、以下のいずれかの環境における IMS アプリケーション・プログラミングのための手引きです。

- IMS Database Control (DBCTL) を含む IMS Database Manager (IMS DB)
- IMS Transaction Manager (IMS TM)
- CICS® EXEC DLI
- WebSphere® Application Server for z/OS®
- 分散プラットフォーム用 WebSphere Application Server
- Java 従属領域 (JMP および JBP)
- スタンドアロン Java アプリケーション開発用のすべての環境

本書は、IMS データベースにアクセスするアプリケーション・プログラムの作成、または IMS メッセージの処理に関するガイダンス情報を提供します。また、IMS と対話する DL/I、EXEC DLI、または JDBC の呼び出しを、さまざまなプログラミング言語を使用して行う方法についても説明しています。API (アプリケーション・プログラミング・インターフェース) に関する情報は、「*IMS V15* アプリケーション・プログラミング *API*」に記載されています。

z/OS の詳細については、IBM Knowledge Center の「z/OS basic skills」トピックを参照してください。

IMS の基本概念を理解するには、「*An Introduction to IMS*」(IBM Press 出版) をお読みになると役立ちます。

IBM では、IMS の学習に役立つような講習会や自習講座を数多く提供しています。利用可能な講習の詳しいリストについては、IBM Skills Gateway にアクセスして、IMS を検索してください。

新規および変更された情報の識別方法

IMS ライブラリーの PDF 資料のほとんどの新規および変更された情報は、左マージン内の文字 (改訂マーカー) によって示されています。「リリース計画」、ならびに「*Program Directory*」および「*Licensed Program Specifications*」の第 1 版 (-00) には、改訂マーカーは含まれていません。

改訂マーカーは、以下の一般的な規則に従っています。

- 技術的な変更のみにマークが付けられています。形式上の変更や文法的な変更には、マークは付けられていません。
- 段落、構文図、リスト項目、操作手順、または図などの要素の一部が変更された場合、その要素の一部だけの変更であっても、要素全体に改訂マーカーが付けられています。
- トピックの変更が 50% を超えた場合には、そのトピック全体に改訂マーカーが付けられています (そのため、新規トピックではなくても、新規トピックのように見えることがあります)。

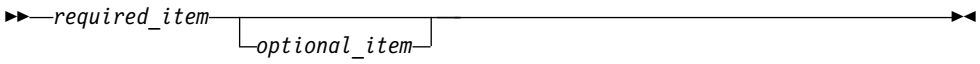
改訂マーカーは情報に加えられたすべての変更を示しているとは限りません。削除されたテキストとグラフィックスには、改訂マーカーでマークを付けることはできないためです。

構文図の読み方

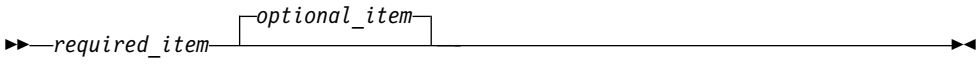
本書で使用されている構文図には、以下の規則が適用されています。

- 構文図は、経路を示す線に沿って、左から右、上から下に読み取ります。以下の規則が使用されます。
  - >>--- 記号は、構文図の始まりを示します。
  - ---> 記号は、構文図が次の行に続くことを示します。
  - >--- 記号は、この構文図が直前の行から続いていることを示します。
  - --->< 記号は、構文図の終わりを示します。
- 必須項目は、水平線 (メインパス) 上に表示されます。

- オプション項目は、メインパスより下に示されます。

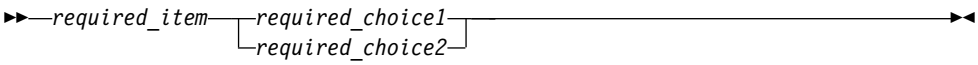


  メインパスより上にオプショナル項目が示されている場合は、その項目が構文エレメントの実行に影響することはなく、読みやすくするためのみの表記です。

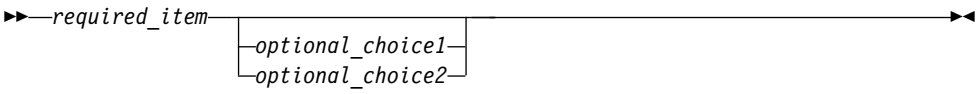


- 複数の項目から選択できる場合は、縦方向に並べて (スタック) 示されます。

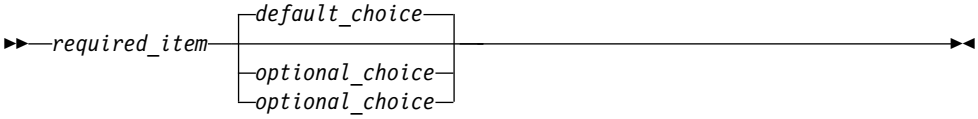
それらの項目の中から 1 つを選択する必要がある 場合は、スタックの中の 1 つの項目がメインパス上に表示されます。



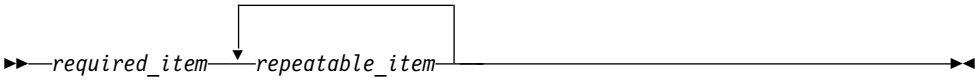
それらの項目から 1 つを選択することがオプションである場合は、スタック全体がメインパスの下に表示されます。



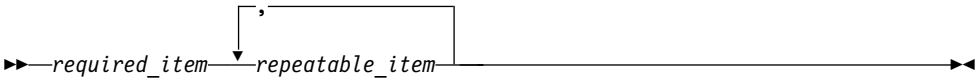
デフォルト項目が含まれている場合、その項目はメインパスより上に示され、他の選択項目はメインパスより下に示されます。



- メインパスの上方にある左に戻る矢印線は、項目が反復可能であることを示します。



反復矢印線にコンマが含まれている場合は、反復項目をコンマで区切る必要があります。



スタック上方の反復矢印線は、スタック内の項目を反復できることを示しています。

- 1 つの構文図を複数のフラグメントに分割しなければならない場合もあります。構文フラグメントはメインの構文図とは別に示されますが、フラグメントの内容は、図のメインパス上にあるものとして読む必要があります。

fragment-name:

- IMS では、b 記号は、該当位置にブランクが 1 つあることを示します。

- キーワード、および該当する場合はキーワードの最小の省略語は、大文字で表されます。これらは、示されているとおりに入力する必要があります。変数は、すべて小文字のイタリック文字で示されます (例えば、*column-name*)。これらは、ユーザーが指定する名前または値を表します。
- キーワードとパラメーターは、構文図で間に句読点が示されていない場合は、少なくとも 1 つのスペースで分離します。
- 句読記号、括弧、算術演算子、およびその他の記号は、構文図で示されたとおりに入力します。
- 脚注は、例えば (1) のように、数字を括弧で囲んで示してあります。

IMS 15 のアクセシビリティー機能

アクセシビリティー機能は、運動障害または視覚障害など身体に障害を持つユーザーが情報技術製品を快適に使用できるようにサポートします。

アクセシビリティー機能

以下のリストは、 IMS 15 を含む z/OS 製品の主なアクセシビリティー機能を示しています。これらの機能は、以下をサポートしています。

- キーボードのみの操作。
- スクリーン・リーダー (読み上げソフトウェア) およびスクリーン拡大鏡によって通常使用されるインターフェース。
- 色、コントラスト、フォント・サイズなど表示属性のカスタマイズ。

キーボード・ナビゲーション

IMS 15 ISPF パネル機能には、キーボードまたはキーボード・ショートカット・キーを使用してアクセスできます。

TSO/E または ISPF を使用して IMS 15 ISPF パネルをナビゲートする詳細については、「*z/OS TSO/E* 入門」、「*z/OS TSO/E* ユーザーズ・ガイド」、および「*z/OS* 対話式システム生産性向上機能 *(ISPF)* ユーザーズ・ガイド 第 *1* 巻」を参照してください。上記の資料には、キーボード・ショートカットまたはファンクション・キー (PF キー) の使用方法を含む、各インターフェースのナビゲート方法が記載されています。それぞれの資料では、PF キーのデフォルトの設定値とそれらの機能の変更方法についても説明しています。

関連のアクセシビリティー情報

IMS 15 のオンライン資料は、IBM Knowledge Center で参照できます。

IBM におけるアクセシビリティー

IBM のアクセシビリティーに対する取り組みについて詳しくは、*IBM Human Ability and Accessibility Center* (www.ibm.com/able) を参照してください。

ご意見の送付方法

お客様のご意見を送り返していただくことは、弊社が正確な情報を提供し、品質の高い情報を提供するうえで重要なことです。本書またはその他の IMS 関連資料についてコメントのある場合、次のいずれかの方法でお送りください。

- IBM Knowledge Center のトピックの下部にある「**Contact Us**」タブをクリックします。
- imspubs@us.ibm.com に E メールを送信します。必ず、資料タイトルと資料番号を記載してください。

弊社が迅速かつ正確に対応するために、ご意見をお送りいただく資料の内容、その掲載個所、改善のためのご提案について可能な限り多くの情報を記載してください。

第 1 部 アプリケーション・プログラミング設計

IMS のアプリケーション・プログラムを設計するには、アプリケーション・データを識別し、アプリケーション・プロセスの要件を分析する必要があります。また、データベースおよびメッセージ処理オプションに関する要件の収集、およびアプリケーション・プログラムのテストなど、その他のタスクを実行する必要があります。

第 1 章 アプリケーションの設計: 導入概念

このセクションでは、アプリケーション・プログラムの設計の概要を記載しています。ここでは、データベース処理に関する基本概念、および記載情報の適用対象となる作業を概説します。

情報のデータベースへの保管およびその処理

データベースにデータを保管したり、データベース内のデータを処理する場合の利点は、必要なデータは重複して存在する必要がなく、各プログラムは必要なデータだけ処理すればよいということにあります。

このことを理解するために、個別のファイル、結合ファイル、およびデータベースの 3 つそれぞれにデータを保管する方法について比較します。

個別のファイルへのデータの保管

組織の各部門のデータを個別のファイルに保管する場合は、各プログラムが必要なデータだけを使用するようにできますが、複数の場所に大量のデータを同時に保管する必要があります。個別のファイルを保持する場合の問題は次のとおりです。

- もっと有効に使えるはずのスペースを冗長データが占領します。
- 個別のファイルの維持は、より困難であり、複雑になります。

例えば、診療所で、外来患者部門、会計部門、眼科部門などの各部門ごとに個別のファイルを保管していると想定してください。

- 外来患者部門には、診療所を訪れた患者のデータが以下のような形式で保管されています。

  識別番号
  名前
  アドレス
  病名
  各病気の初診日
  患者が診察のために診療所を訪れた日付
  各病気のための治療法
  担当医
  治療費
- 会計部門でも、患者についての情報を保管しています。 会計部門で各患者について保管している情報は、以下のとおりです。

  識別番号
  名前
  アドレス
  治療費

支払額

- 眼科部門で各患者について保管している情報は、以下のとおりです。

  識別番号
  名前
  アドレス
  眼科に関連のある病気
  各病気の初診日
  患者の家族の名前
  患者と家族との関係

これらの部門がそれぞれ個別のファイルを保持している場合は、各部門で必要なデータだけを使用することができますが、その多くは冗長データとなります。例えば、診療所のすべての部門が少なくとも患者の番号、名前、住所を使用しています。 さらにデータを更新する場合にも問題があります。1 つの部門で一部のデータを変更した場合、それぞれの個別のファイルの同じデータを更新しなければなりません。そのため、それぞれの部門のファイルのデータを最新の状態にしておくことが難しくなります。 あるファイルに現行データが存在していても、別のファイルには無効になったデータが残っている場合があります。

結合ファイルへのデータの保管

データを保管するもう 1 つの方法は、すべてのファイルを 1 つのファイルに結合して、すべての部門で使用できるようにすることです。医療業務を例にあげると、各部門で使用される患者のレコードは、以下のようなフィールドを含んでいます。

識別番号
名前
アドレス
病名
各病気の初診日
患者が診察のために診療所を訪れた日付
各病気のための治療法
担当医
治療費
支払額
患者の家族の名前
患者と家族との関係

結合ファイルを使用すると、すべてのデータが 1 つの場所にあるので、更新の際に生じる問題は解決されますが、必要な部分を得るためには、このデータを処理するプログラムがそのファイル・レコード全体にアクセスしなければならないというような新しい問題も発生します。例えば、会計プログラムは、患者の番号、治療費、および支払額だけを処理するためにも、他のすべてのフィールドにアクセスしなければなりません。さらに、患者のレコード内の任意のフィールド形式を変更すると、そのフィールドを使用したプログラムだけでなく、すべてのアプリケーション・プログラムが影響を受けます。

結合ファイルを使用する場合には、すべてのプログラムがレコード内のすべてのフィールドにアクセスすることになるので、セキュリティーという面でも問題が生じる可能性があります。

データベースへのデータの保管

データベースにデータを保管すると、個別のファイルと結合ファイルの両方の利点が得られます。すなわち、すべてのデータは重複して存在する必要がなく、各プログラムは必要なデータにアクセスします。 このことは、以下のような意味を持ちます。

- フィールドを更新する場合には、一箇所でだけ更新作業を行えばよい。
- さまざまな情報を一箇所にだけ保管するので、ある場所には更新済みの情報があり、別の場所には更新前の情報があるということはない。
- 各プログラムは、必要なデータにだけアクセスする。
- プログラムがプライベートな情報または保護された情報にアクセスするのを防止できる。

さらに、データベースにデータを保管すると、他の方法にはない次のような 2 つの利点があります。

- 一部のデータベース・レコードの形式を変更しても、変更された情報を使用しないプログラムには影響しない。
- データがどのような方法で保管されても、プログラムには影響しない。

プログラムは物理データとは関係がないので、データベースはすべてのデータを一度に保管することができます。また、各プログラムが必要なデータだけを使用することもできます。 データベースでは、データが保管されるときの形式と、アプリケーション・プログラムがそのデータを参照するときの形式とは異なっています。

データベース階層の例

IMS DB では、レコードは階層形式で保管され、アクセスされます。階層とは、レコード内のデータの各部分がレコード内のデータの他の部分とどのような関係にあるかを示したものです。

IMS は、同じ対象項目に関連する個別の情報の関係を定義することによって、データベース・レコード内の個々の情報を結び付けます。これがデータベース階層です。

医療の階層の例

次の図に示す医療データベースには、診療所が患者に関して保有している情報が入っています。医療階層構造例の階層は、全機能データベースと高速機能高速処理データベース (DEDB) で使用されています。

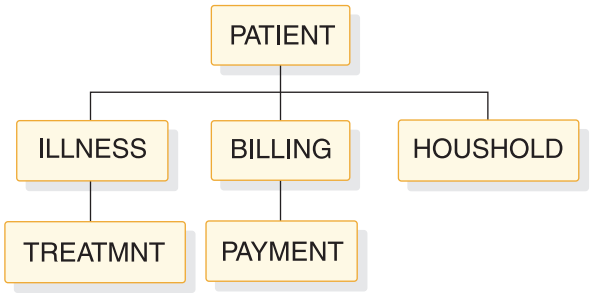


図 *1.* 医療の階層構造

前掲の図に表記されている各データは、階層内のセグメント と呼びます。各セグメントには、1 つまたは複数の情報フィールド が含まれます。例えば、PATIENT というセグメントには、患者に直接関連のある情報、すなわち患者の識別番号、名前、住所がすべて含まれます。

定義 : セグメント とは、アプリケーション・プログラムがデータベースからリトリーブできる最小単位のデータのことです。フィールド とは、最小単位のセグメントのことです。

医療データベースでは PATIENT セグメントをルート・セグメントといいます。ルート・セグメントの下のセグメントは、ルート・セグメントの従属 セグメントあるいは子セグメントです。例えば、ILLNESS、BILLING、および HOUSHOLD はすべて PATIENT の子セグメントになります。 ILLNESS、BILLING、および HOUSHOLD は、PATIENT の直接従属と呼ばれます。 TREATMNT と PAYMENT も PATIENT の従属ですが、これらは、低レベルの階層にあるため、直接従属セグメントではありません。

1 つのデータベース・レコードは、単一ルート・セグメント (ルート・セグメント・オカレンス) と、そのすべての従属セグメントです。医療の例では、1 つのデータベース・レコードが、1 人の患者についてのすべての情報になります。

定義 : ルート・セグメント とは、最高位レベルのセグメントのことです。従属セグメントとは、ルート・セグメントよりも下位のセグメントのことです。ルート・セグメント・オカレンスとは、データベース・レコードおよびそのすべての従属セグメントのことです。

各データベース・レコードは、1 つのルート・セグメント・オカレンスだけを持ちますが、低レベルで複数のオカレンスを持つ場合もあります。 例えば、患者についてのデータベース・レコードには、PATIENT セグメント・タイプのオカレンスが 1 つだけ含まれますが、この中には、その患者についての複数の ILLNESS と TREATMNT セグメント・オカレンスが含まれる場合があります。

以下に示すいくつかの表は、この階層構造における各セグメントのレイアウトを示しています。

セグメントのフィールド名は、各表の最初の行にあります。各フィールド名の下の数値は、そのフィールドに定義された長さをバイト数で表しています。

- **PATIENT** セグメント

次の表は、PATIENT セグメントを示しています。

このセグメントには、次の 3 つのフィールドがあります。

– 患者番号 (PATNO)
– 患者名 (NAME)
– 患者の住所 (ADDR)

PATIENT には PATNO という固有キー・フィールドがあります。PATIENT セグメントは、患者番号の昇順で保管されています。このデータベースにおいて最も低い患者番号は 00001 で、最も高い患者番号は 10500 です。

表 *1. PATIENT* セグメント

フィールド名	フィールド長
PATNO	10
NAME	5
ADDR	30

- **ILLNESS** セグメント

  次の表は、ILLNESS セグメントを示しています。

  このセグメントは、次の 2 つのフィールドから構成されます。

  – 患者が病気のために来院した日付 (ILLDATE)
  – 病名 (ILLNAME)

  キー・フィールドは ILLDATE です。 1 人の患者が同日に複数の病気のために来院する可能性もあるので、このキー・フィールドは一意ではありません。つまり、複数の ILLNESS セグメントが同じキー・フィールド値を持つ場合があります。

  通常、インストール時にデータベース管理者 (DBA) は、キーが等しいかまたはキーを持たない場合のデータベース・セグメントの配置順序を決定します。DBA は、DBD の SEGM ステートメントの RULES キーワードを使用して、セグメントの順序を指定できます。

  等しいキーをもつ、またはキーをもたないセグメントの場合、RULES はそのセグメントの挿入位置を決定します。RULES=LAST のとき、等しいキーをもつ複数の ILLNESS セグメントは、そのキーのなかで先入れ先出し法 (FIFO) で保管されます。固有キーをもつ ILLNESS セグメントは、RULES に関係なく、日付フィールドの昇順に保管されます。ILLDATE は、YYYYMMDD という形式で指定されます。

表 *2. ILLNESS* セグメント

フィールド名	フィールド長
ILLDATE	8
ILLNAME	10

- **TREATMNT** セグメント

次の表は、TREATMNT セグメントを示しています。

このセグメントは、次の 4 つのフィールドから構成されます。

– 治療日 (DATE)
– 患者に投与された薬品 (MEDICINE)
– 患者が受け取った薬品の量 (QUANTITY)
– その治療を処方した医師の名前 (DOCTOR)

TREATMNT セグメントのキー・フィールドは DATE です。1 人の患者が同日に複数の処方と治療を受ける可能性があるため、DATE は、非ユニーク・キー・フィールドです。 ILLNESS と同じように、TREATMNT は RULES=LAST を持つものとして指定されています。 TREATMNT セグメントも、先入れ先出し法で保管されます。DATE は ILLDATE と同じように YYYYMMDD 形式で指定されます。

表 *3. TREATMNT* セグメント

フィールド名	フィールド長
日付	8
MEDICINE	10
QUANTITY	4
DOCTOR	10

- **BILLING** セグメント

  次の表は、BILLING セグメントを示しています。このセグメントには、現在の請求金額を表す 1 つのフィールドだけがあります。BILLING にはキー・フィールドはありません。

表 *4. BILLING* セグメント

フィールド名	フィールド長
BILLING	6

- **PAYMENT** セグメント

  次の表は、PAYMENT セグメントを示しています。このセグメントには、月の請求金額を表す 1 つのフィールドだけがあります。PAYMENT セグメントにはキー・フィールドがありません。

表 *5. PAYMENT* セグメント

フィールド名	フィールド長
PAYMENT	6

- **HOUSHOLD** セグメント

  次の表は、HOUSHOLD セグメントを示しています。

  このセグメントには、以下の 2 つフィールドが含まれます。

  – 患者の世帯構成員の名前 (RELNAME)
  – 各世帯構成員と患者の関係 (RELATN)

HOUSHOLD セグメントのキー・フィールドは RELNAME です。

表 *6. HOUSHOLD* セグメント

フィールド名	フィールド長
RELNAME	10
RELATN	8

銀行口座の階層の例

銀行口座の階層構造は、主記憶データベース (MSDB) で使用されるアプリケーション・プログラムの使用の例です。医療の階層構造の例では、特定の患者のデータベース・レコードを、特定の PATIENT セグメントとその下のすべてのセグメントが構成しています。MSDB では、銀行口座の例のように、データベース・レコード全体が 1 つのセグメントになっています。データベース・レコードには、そのセグメントに含まれている以外のフィールドは含まれません。

MSDB には、関連 MSDB と非関連 MSDB の 2 つの種類があります。関連 MSDB では、各セグメントは 1 つの論理端末によって「所有されて」います。「所有された」セグメントのみが、それを所有する端末から更新できます。非関連 MSDB では、セグメントは論理端末によって所有されません。以下の関連 MSDB および非関連 MSDB の例では、これら 2 つのタイプのデータベースの相違点を示します。

関連 ***MSDB***

関連 MSDB には、固定関連と動的関連があります。固定関連 MSDB に特定の銀行窓口に関する要約データを保管することができます。例えば、この窓口の端末に ID コードを割り当てることができます。これにより、その窓口の当日の取引の数と残高を記録できます。この種のアプリケーションでは、3 つのフィールドを持つセグメントが必要です。

TELLERID
窓口を識別する 2 文字のコード

TRANCNT
その窓口が処理した取引の数

TELLBAL
その窓口の残高

次の表は、このタイプのアプリケーション・プログラムのセグメントがどのようになっているのかを示しています。

表 *7.* 固定関連 *MSDB* の窓口セグメント

TELLERID	TRANCNT	TELLBAL

固定関連 MSDB の特性としては、次のようなことが含まれます。

- セグメントの読み取りと置き換えだけしか実行できません。セグメントの削除または挿入を実行することはできません。銀行窓口の例で、窓口は多くの処理済み

のトランザクションを変更できますが、利用者はいかなるセグメントも追加や削除はできません。セグメントの追加または削除が必要になることはありえません。

- 各セグメントは、1 つの論理端末に割り当てられます。セグメントを変更することができるのは、それを所有する端末だけですが、他の端末でそのセグメントを読むことはできます。銀行窓口の例では、窓口係が他の窓口の情報について更新はさせませんが、表示は許可します。窓口係は、自分の取引に関して責任を持ちます。
- セグメントを所有する論理端末の名前が、そのセグメントのキーになります。非 MSDB セグメントの場合とは異なり、MSDB キーはセグメントのフィールドではありません。このキーは、セグメントの記憶とアクセスのための手段として使用されます。
- 1 つの論理端末は、ある 1 つの MSDB の中で 1 つのセグメントのみ所有できます。

1 つの支店で、動的関連 MSDB に銀行の全窓口の活動を要約したデータを保管することができます。例えば、このセグメントには以下の情報が含まれます。

BRANCHNO
支店の識別番号

TOTAL
支店の現在残高

TRANCNT
支店のその日の取引件数

DEPBAL
支店の合計預け入れ金額 （ドル単位） を表す預け入れ残高

WTHBAL
支店の合計引き出し金額 （ドル単位） を表す引き出し残高

次の表では、支店の要約セグメントが動的関連 MSDB でどのように表現されるかを示しています。

表 *8.* 動的関連 *MSDB* における支店の要約セグメント

BRANCHNO	TOTAL	TRANCNT	DEPBAL	WTHBAL

動的関連 MSDB と固定関連 MSDB の相違点

- 所有している論理端末は、動的関連 MSDB でセグメントの削除と挿入を行うことができます。
- MSDB には、割り当てられていないセグメントのプールを用意できます。この種のセグメントは、論理端末によって挿入されたときにその論理端末に割り当てられ、論理端末によって削除されたときにプールに戻されます。

非関連 ***MSDB***

非関連 MSDB は、同じ時間枠の間に複数の端末によって更新されるデータを保管するために使用されます。例えば、個人の銀行残高に関するデータは、非関連

MSDB セグメントに記憶できるため、窓口係は、どの端末でも情報を更新することができます。プログラムのアクセスが必要となる可能性のあるデータは、以下のセグメント・フィールドにあります。

ACCNTNO
口座番号

BRANCH
口座が開設されている支店の名前

TRANCNT
この口座に関する今月の取引の数

BALANCE
現在の残高

次の表では、非関連 MSDB アプリケーション・プログラムの口座セグメントがどのように表現されるかを示しています。

表 *9.* 非関連 *MSDB* における口座セグメント

ACCNTNO	BRANCH	TRANCNT	BALANCE

非関連 MSDB の特性は以下のとおりです。

- 関連 MSDB の場合とは異なり、セグメントは端末によって所有されません。したがって、IMS プログラムと高速機能プログラムはこれらのセグメントを更新できます。セグメントの更新は、所有している論理端末以外でも可能です。
- ユーザーのプログラムは、セグメントを削除または挿入することはできません。
- セグメント・キーには、論理端末の名前を使用できます。非関連 MSDB には端末関連キーが付けられます。この場合、セグメントが論理端末によって所有されるのではなく、論理端末名がセグメントの識別に利用されます。
- キーが論理端末名でない場合には、任意の値にすることができ、セグメントの最初のフィールドに入ります。セグメントは、キー・シーケンスでロードされます。

プログラムのデータのビュー

IMS では、データベース記述 (DBD) およびデータベース・プログラム連絡ブロック (DB PCB) という 2 種類の制御ブロックを使用して、データベースにデータを保管する方法からアプリケーションを独立させます。

データベース記述 (DBD)

データベース記述 *(DBD)* は、データベースの物理構造です。 DBD は、表示形式および内容、またはフィールドも定義します。これらは、データベース内の各セグメント・タイプを構成します。

例えば、「医療の階層構造の例」に示した医療データベース階層の DBD は、階層の物理構造と階層内の 6 つのセグメント・タイプ
(PATIENT、ILLNESS、TREATMNT、BILLING、PAYMENT、および HOUSHOLD) を表しています。

関連資料 : DBD の生成の詳細については、「*IMS V15* データベース・ユーティリティー」を参照してください。

データベース・プログラム連絡ブロック (DB PCB)

データベース・プログラム連絡ブロック *(DB PCB)* は、アプリケーション・プログラムのデータベースのビューを定義する制御ブロックです。アプリケーション・プログラムは、データベース内のいくつかのセグメントだけを処理すればよいことが頻繁にあります。 PCB は、データベース内のどのセグメントにプログラムがアクセスできるかを定義します。言い換えれば、プログラムがどのセグメントをセンシティブと見なすかを定義します。

プログラムが利用できるデータ構造には、プログラムがセンシティブと見なすセグメントだけが入ります。 PCB は、アプリケーション・プログラムにデータ構造のセグメントを処理させる方法も定義します。これは、プログラムにセグメントを読み取らせるだけか、または更新もできるようにするかということです。

最高レベルのデータ可用性を得るためには、PCB が要求するのは、作業を完了するのに必要な最小数のセンシティブ・セグメントと最低限の機能にする必要があります。

単一アプリケーション・プログラムのすべての DB PCB は、プログラム仕様ブロック (PSB) に含まれます。プログラムは、各データ構造につき 1 つの DB PCB (1 つのデータ構造だけを処理する場合) だけを使用することも、複数の DB PCB を使用することもあります。

関連資料 : PSB の生成の詳細については、「*IMS V15* データベース・ユーティリティー」を参照してください。

次の図は、アプリケーション・プログラムのビュー定義の概念を示しています。診療所の患者に対する請求金額を計算し印刷する会計処理プログラムには、PATIENT、BILLING、および PAYMENT セグメントだけが必要です。このプログラムの DB PCB には、次の図に示すデータ構造を定義できます。

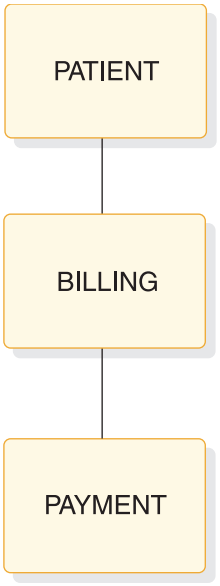


図 2. データベースのアカウンティング・プログラムのビュー

これに対して、患者の病名および治療法に関する情報を使用してデータベースを更新するプログラムは、PATIENT、ILLNESS、および TREATMNT セグメントを処理する必要があります。このプログラムの DB PCB には、次の図に示すデータ構造を定義できます。

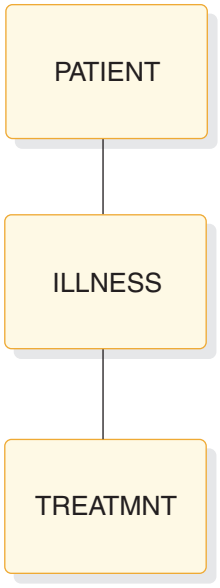


図 3. データベースの患者の病名プログラムのビュー

プログラムがデータベースのすべてのセグメントを処理することが必要な場合もあります。この場合には、DB PCB に定義されているデータベースのプログラムのビューは、DBD に定義されているデータベース階層と同じになります。

アプリケーション・プログラムは、データベース内の必要なセグメントだけを処理します。そのため、処理されていないセグメントの形式を変更しても、プログラムを変更する必要はありません。プログラムは、プログラムがアクセスするセグメン

トによってのみ影響を受けます。データベース内の特定のセグメントをセンシティブと見なすことに加え、プログラムは、セグメント内の特定フィールドだけを重要と見なすこともできます。プログラムがセンシティブと見なさないセグメントまたはフィールドを変更しても、そのプログラムには影響はありません。 PSBGEN 時に、セグメントとフィールド・レベル・センシティビティーを定義します。

定義: フィールド・レベル・センシティビティーとは、セグメント内の特定のフィールドのみを、プログラムがセンシティブと見なすことです。

関連資料 : さらに詳しくは、 *IMS V15* データベース管理 を参照してください。

データベース・レコードの処理

アプリケーション・プログラムは、データベースにある情報を処理するために、制御の受け渡し、処理要求の送信、および DL/I 呼び出しを使用した情報の交換という 3 とおりの方法で IMS と通信します。

- 制御の受け渡し。IMS は、プログラムのエントリー・ステートメントを使用して、制御をアプリケーション・プログラムに渡します。その処理を終えると、プログラムは IMS に制御を戻します。

  CICS オンライン・プログラムを実行する場合には、 CICS がアプリケーション・プログラムに制御を渡し、そのプログラムは IMS 要求を作成するために PSB をスケジュールします。プログラムは、CICS に制御を戻します。バッチまたは BMP プログラムを実行する場合には、 IMS は、既存のスケジュール済みの PSB とともに、制御をプログラムに渡します。
- 処理要求の送信。以下の 2 つの方法のいずれかで IMS に処理要求を送信します。
  - IMS では、 DL/I 呼び出しを出して、データベースを処理します。
  - CICS では、DL/I 呼び出しまたは EXEC DLI コマンドのいずれかを出すことができます。 EXEC DLI コマンドは、DL/I 呼び出しよりも高水準言語に近いものです。
- DL/I 呼び出しを使用する情報の交換。プログラムは次の 2 つのエリア内の情報を交換します。
  - DL/I 呼び出しは、AIB インターフェースのいずれかを使用した場合には、制御ブロックと AIB 通信ブロックに要求結果を報告します。 DL/I 呼び出しを使用して作成したプログラムの場合には、この制御ブロックは DB PCB になります。 EXEC DLI コマンドを使用して作成したプログラムの場合には、この制御ブロックは、DLI インターフェース・ブロック (DIB) になります。 DIB の内容は、プログラムで最後に実行された DL/I コマンドの状況を反映します。プログラムには適切な制御ブロックのマスクが含まれており、このマスクを使用して要求結果を検査します。
  - データベースからセグメントを要求すると、IMS は入出力域にセグメントを戻します。データベースのセグメントを更新する必要があるときには、入出力域に新しい値のセグメントを入れてください。

アプリケーション・プログラムは、データベースを読み取って、更新することができます。データベースを更新するときには、セグメントの置換、削除、または追加を行うことができます。 IMS では、処理したいセグメント、ならびにそのセグメ

ントを読み取るか更新するかを、DL/I 呼び出しで示します。 CICS では、DL/I 呼び出しまたは EXEC DLI コマンドを使用して、必要な処理を示すことができます。

アプリケーション開発の作業

IMS アプリケーション開発、およびアプリケーションの構成要素であるプログラムの開発には、以下の作業が関係しています。

アプリケーションの設計

アプリケーション・プログラムの設計は、場所によってもアプリケーションによっても異なります。

そのため、本書では、アプリケーション・プログラム設計の部分である初期段階の作業については取りあげていません。 その代わりに、以前にアプリケーションが開発した初期仕様と関係のある作業だけを取りあげています。 アプリケーション設計の作業は次のとおりです。

- アプリケーション・データ要件の分析

  アプリケーション設計の重要な 2 つの部分は、アプリケーションの各業務処理に必要なデータを定義し、各業務処理のローカル・ビューを設計することです。
- アプリケーション処理要件の分析

  アプリケーションの一部である業務処理を理解するためには、さまざまなタイプのアプリケーション・プログラムで利用できるという点から、各業務処理の要件を分析することができます。
- データベース・オプションの要件収集

  次に最も効果的に要件に合うデータベース・オプションを参照し、それぞれのオプションと関連のあるアプリケーションのデータ要件についての情報を収集する必要があります。
- メッセージ処理オプションの要件収集

  アプリケーションが端末ならびに他のアプリケーション・プログラムと通信する場合には、メッセージ処理オプションならびにそれを満たす要件を参照してください。

CICS アプリケーションの設計について詳しくは、「*CICS Transaction Server for z/OS CICS* アプリケーション・プログラミング・ガイド」を参照してください。

仕様の開発

仕様の開発には、アプリケーションがなにをどのように実行するかを定義することが含まれます。仕様開発の作業は、具体的なアプリケーションや使用する標準に全面的に依存するため、本書では説明していません。

設計の実装

アプリケーションの各プログラムの仕様を開発したら、それらの仕様に基づいてプログラムの構成およびコーディングを行うことができます。設計の実装作業は次のとおりです。

- プログラムのデータベース処理部分の作成

  プログラムの設計が完了すると、開発されたプログラミング仕様に基づいた要求とデータ域を構成およびコーディングすることができます。

- プログラムのメッセージ処理部分の作成

  端末および他のプログラムと通信するプログラムを作成している場合には、プログラムのメッセージ処理部分を構築し、コーディングする必要があります。

- **APPC/IMS** 要件の分析

  IMS TM の LU 6.2 機能により、アプリケーションがネットワークを介して分散されることになります。

- アプリケーション・プログラムのテスト

  プログラムのコーディングを終了したら、そのプログラムを単独でテストした後、システムの一部としてテストしてください。

- アプリケーション・プログラムの文書化

  プログラムの文書化は、プロジェクトの全期間を通して継続する作業であるため、漸進的に追加していくのが最も効果的です。プログラムのテストが完了したら、プログラムの使用および保守を行う要員に情報を提供する必要があります。

第 2 章 アプリケーションの設計: データ・ビューとローカル・ビュー

エンド・ユーザーの要件を満たすアプリケーションを設計することには、多様な作業と、通常は複数の部門の作業者が関係します。アプリケーション設計が開始されるのは、ある部門またはビジネス領域で、何らかのタイプの処理が必要であるとの声が上がるときです。アプリケーション設計が終了するのは、アプリケーション・システムの各部分 (例えば、プログラム、データベース、表示画面、およびメッセージ形式など) の設計が完了したときです。

アプリケーション設計の概要

アプリケーション設計プロセスは、環境やアプリケーションに応じて異なります。このセクションで説明している概要、およびアプリケーション設計の文書化や既存アプリケーションの変換に関する提案だけが、設計作業の唯一の実行方法というわけではありません。

この概要の目的は、参照用の枠組みを提示し、このセクションで説明している手法およびガイドラインが、プロセスのどの部分に適合するかを理解できるようにすることです。ここで説明する作業の実行順序、および各作業の重要度は、作業者が決めた設定内容に応じて異なります。また、各作業に関わる作業者とその肩書きは、作業サイトに応じて異なる場合があります。以下のような作業があります。

- 規格の確立

  設計プロセス全体を通じて、確立された規格に注意を払う必要があります。一般に規格が確立される分野には、次のものがあります。

  - 命名規則 (例えばデータベースや端末などの)
  - 画面およびメッセージの形式
  - データベースの制御およびデータベースへのアクセス
  - プログラミングおよび規則 (共通ルーチンおよびマクロに関するもの)

  これらの分野での規格のセットアップは、通常はデータベース管理者やシステム管理者が担当して継続的に実行される作業です。

- セキュリティー標準の順守

  セキュリティーは無許可のアクセスおよび使用からリソースを保護します。規格の定義と同様に、セキュリティー・システムを適切に設計することは、たいていは継続的に実行される作業です。アプリケーションの変更または拡張に伴い、たいていの場合セキュリティーも何らかの方法での変更が必要になります。セキュリティーは、アプリケーション設計の初期段階における重要な考慮事項です。

  セキュリティー標準および要件を確立することは、通常はシステム管理の担当範囲です。この標準は、アプリケーションの要件に基づくものになります。

  セキュリティーの考慮事項には、以下のものがあります。

- データベースへのアクセスならびにデータベースの使用
- 端末へのアクセス
- アプリケーション出力の配布
- プログラム修正の制御
- トランザクションおよびコマンド入力

- アプリケーション・データの定義

  アプリケーションが必要とするデータを識別することは、アプリケーション設計の重要な部分です。データ定義作業の 1 つとして、エンド・ユーザーを対象に調査して、必要な処理の実行にどのような情報が求められるかを把握します。

- データベース設計への入力を提供

  データベースを設計する際、そのデータベースを処理するすべてのアプリケーションの要件を満たせるようにするため、データベース管理者 (DBA) は、各アプリケーションのデータ要件に関する情報を必要とします。この情報を収集して提供するための 1 つの方法は、アプリケーションで扱う各ビジネス・プロセスのローカル・ビューを設計することです。ローカル・ビューとは、特定のビジネス・プロセスが必要とするデータの説明です。

- アプリケーション・プログラムの設計

  アプリケーション・フローおよびアプリケーション・システムの外部インターフェースの全体を定義した後、必要な処理を実行するプログラムを定義します。この作業における最重要の考慮事項としては、標準、セキュリティー要件、プライバシー要件、およびパフォーマンス要件などがあります。プログラム用に作成する仕様には、以下のものを含める必要があります。

  - セキュリティー要件
  - 入出力データ形式およびボリューム
  - データ検査と妥当性検査の必要性
  - ロジック仕様
  - パフォーマンス要件
  - リカバリー要件
  - リンク要件および規則
  - データの可用性に関する考慮事項

  さらに、ネットワークおよびユーザー・インターフェース設計の担当者用に、アプリケーションに関する情報提供が求められる場合もあります。

- アプリケーション設計プロセスの文書化

  アプリケーション設計プロセスに関する情報を記録することは、現在および将来そのアプリケーションを扱う他の作業者にとって価値があります。役立つ情報の 1 つは、なぜそのアプリケーションの設計方法を採用したかについての説明です。この情報は、データベース、IMS システム、およびアプリケーションのプログラムの担当者にとって役立つ可能性があります。特に将来、アプリケーションのいずれかの部分の変更が必要になる場合に役立ちます。アプリケーション設計の文書化は、設計プロセスの終了時に行うのではなく実行中に行うと、最も徹底したものになります。

- 既存アプリケーションの変換

  既存アプリケーションを IMS 向けに変換する際の主要な 1 つの面は、既に存在しているものを把握することです。既存システムの変換を開始する前に、そのシステムの現行の稼働方法に関する情報を可能な限り把握します。例えば、以下の情報は、変換を開始する際に役立つ場合があります。

  – アプリケーションによって使用されるすべてのレコードのレコード・レイアウト
  – 各データ・エレメントごとのデータ・エレメント・オカレンス数
  – 既存の関連データベースの構造

関連概念:

107 ページの『データ・セキュリティーの提供』
115 ページの『オンラインのセキュリティー要件の識別』
『アプリケーション・データの識別』
25 ページの『ローカル・ビューの設計』

アプリケーション・データの識別

アプリケーション設計の重要な 2 つの局面は、アプリケーション・データの識別と、特定の業務処理が必要とするデータの説明です。

アプリケーション・データを識別するステップの 1 つは、ユーザーが実行したい処理を完全に理解することです。アプリケーションのデータ要件を定義するためには、入力データおよび必要な出力データを理解することが必要です。ユーザーの処理ニーズに含まれている業務処理についても理解する必要があります。 アプリケーション・データを識別する際には、以下の 3 つの作業があります。

- 業務処理で必要なデータのリスト
- データの命名
- データの文書化

必要なアプリケーション・データを分析すると、それらのデータをエンティティーまたはデータ・エレメントとしてカテゴリー化することができます。

定義**:** エンティティー は、記憶できる情報の対象となるものです。データ・エレメント は、エンティティーに関連のある名前の付いた最小単位のデータです。 これは、エンティティーを説明する情報のことです。

例**:** 研修関連のアプリケーションにおける「生徒」と「講習」は両方ともエンティティーであり、これらは、データ収集および処理の 2 つの対象となります。次の表では、生徒と講習というエンティティーに関連のあるいくつかのデータ・エレメントを示しています。エンティティーは、エンティティーに関連のあるデータ・エレメントと共にリストされています。

表 *10.* エンティティーおよびデータ・エレメント :

エンティティー	データ・エレメント
生徒	生徒名
	生徒の番号

表 *10.* エンティティーおよびデータ・エレメント *(*続き*)* :

エンティティー	データ・エレメント
講習	講習の名前
	講習の番号
	講習の長さ

IMS データベースにこのデータを保管すると、データ・エレメントのグループは、階層のセグメントになる可能性があります。 各データ・エレメントは、そのセグメントのフィールドになる可能性があります。

関連概念:

17 ページの『アプリケーション設計の概要』

データ・エレメントのリスト

アプリケーション・データを識別するために、データ・エレメントをリストします。

例として、アプリケーション・データを識別するために、顧客に対して技術的な研修を提供する会社を考えてみます。この研修会社には、「Headquarters」という本社オフィスが 1 つあり、「Ed センター」という複数のローカル研修センターがあります。

クラスとは、特定の Ed センターで指定された日に行われる講習の 1 単位のことです。 ある講習がいくつかの Ed センターで複数行われる場合があり、それぞれは別個のクラスとなります。 Headquarters は、提供されるすべての講習内容を開発する責任があり、各 Ed センターは、クラスのスケジュールとそのクラスに生徒を登録する責任があります。

この研修会社の要件の 1 つが、Ed センターのすべてのクラスについての現行名簿を毎週印刷することであると想定してください。現行名簿には、クラスとそのクラスに登録されている生徒についての情報が書かれています。 Headquarters は、現行名簿を次の図に示す形式にする必要があります。

```
CHICAGO                                                    01/04/04

     TRANSISTOR THEORY                    41837
     10 DAYS
     INSTRUCTOR(S): BENSON, R.J.          DATE: 01/14/04

   STUDENT        CUST   LOCATION              STATUS   ABSENT   GRADE
 1.ADAMS, J.W.    XYZ    SOUTH BEND, IND       CONF
 2.BAKER, R.T.    ACME   BENTON HARBOR, MICH   WAIT
 3.DRAKE, R.A.    XYZ    SOUTH BEND, IND       CANC
   .
   .
   .
33.WILLIAMS, L.R. BEST   CHICAGO, ILL          CONF


  CONFIRMED = 30
  WAIT-LISTED = 1
  CANCELED = 2
```

図 *4.* 技術研修のための現行名簿の例

特定の業務処理のためにデータ・エレメントをリストする場合には、必要な出力を参照してください。 前の図に示す現行名簿は、Chicago Ed センターで 2004 年 1 月 14 日から 10 日間の予定で開かれる「Transistor Theory」というクラスの名簿です。各講習には、講習と関連付けられている講習コードがあります。この場合は 41837 です。特定の講習のコードは、常に同じです。例えば、Transistor Theory が New York でも行われる場合、その講習コードは同じく 41837 です。名簿には、講習を教える講師の名前も示されます。例では、1 人の講師だけを示していますが、2 人以上の講師が必要な講習もあります。

各生徒ごとに、その登録簿に以下の情報が保持されています。すなわち、各生徒別の順序番号、生徒名、生徒所属会社 (CUST)、その会社の住所、このクラスにおける生徒の登録現況、この生徒の欠席日数と評価です。講習と生徒に関する上記のすべての情報が入力情報となります。

現在の日付 (名簿が印刷された日付) は右上の隅に表示されます (01/04/04)。現在の日付は、出力専用データの一例です。このデータはオペレーティング・システムによって生成されたものであり、データベースに保管されているものではありません。

左下の隅には、クラスの登録状況の集計が書かれています。 このデータは、入力データに含まれません。 これらの値は、処理中のプログラムによって判別されます。

データ・エレメントをリストする場合には、ローカル・ビューを設計する際に頻繁にデータ・エレメントを参照することになるので、省略形にすると便利です。

現行名簿のデータ・エレメント・リストは以下の通りです。

EDCNTR
クラスを提供する Ed センター名

日付 クラスを開始した日付

CRSNAME
講習の名

CRSCODE
講習のコード

LENGTH
講習日数

INSTRS
クラスを教えるインストラクター名

STUSEQ#
生徒番号

STUNAME
生徒の名前

CUST 生徒の勤務している会社名

LOCTN
生徒の会社の住所

STATUS
クラスにおける生徒の登録状況 - 確認終了、待機リスト、または取り消し済み

ABSENCE
生徒の欠席日数

GRADE
講習における生徒の評価

データ・エレメントをリストしたら、これらのエレメントが説明する主要なエンティティーを選択してください。この場合には、主要なエンティティーはクラスになります。各生徒に関する多くの情報があり、一般的に研修コースに関する情報もいくらかありますが、この情報をすべて 1 つにすると、特定のクラスに関連があります。各生徒に関する情報 (例えば、登録状況、欠席率、評価など) が特定のクラスに関連付けられていない場合は、その情報は無意味です。このことはリストの前半にあるデータ・エレメントについても言えます。Ed センター、クラスを開始した日付、講師などのデータは、それがどのクラスについてのものかがわからなければ何の意味もありません。

データ・エレメントの命名

アプリケーションが使用するデータ・エレメントの中には、既に存在しており、名前が付けられているものもあります。データ・エレメントをリストしたら、データベース管理者 (DBA) に問い合わせて、存在しているデータ・エレメントがあるかどうか調べます。

データ・エレメントの命名を始める前に、従う必要のある命名標準に注意してください。データ・エレメントを命名するときには、できるだけ最もエレメントを説明している名前を使用するようにしてください。少なくともいくつかの同じデータを他のアプリケーションが使用することになるので、名前は、すべての人にとって同じことを意味するものでなければなりません。独自のアプリケーションに対してだけ意味を持つような名前にしないでください。

推奨事項: ローカル名ではなくグローバル名を使用することをお勧めします。グローバル名とは、名前の意味が特定のアプリケーション以外のアプリケーションにも明確な名前です。ローカル名とは、理解のために特定のアプリケーションの状況の中に出てくる名前です。

ローカル名を使用した場合の問題の 1 つとしては、同義語、すなわち同じデータ・エレメントに 2 つの名前が作成されるということがあります。

例えば、現行名簿の例では、生徒所属会社は「顧客」の代わりに、単に「会社」と呼ばれています。しかし、その研修関連会社の会計部門では、支払いアプリケーションの同じ部分のデータ、つまり生徒所属会社名を「顧客」として使用していると想定してください。これは、2 つの業務処理で、同じ部分のデータを 2 つの異なる名前で使用していることになります。 最悪の場合、「顧客」と「会社」に同じデータが含まれていること気付かなければ、冗長データになってしまいます。このことを解決するためには、このデータ・エレメントを使用する両方の部門で、承認されているグローバル名を使用することです。 この場合には、「顧客」のほうがより簡

単に認識され、選択しやすいでしょう。この名前はデータ・エレメントを固有に識別し、この研修関連会社における特別な意味を持ちます。

データ・エレメント名を選択するときには、修飾子を使用して、それぞれの名前が 1 つの意味だけを持つようにしてください。

例えば、Headquarters が、開発された順番に各研修コースに番号を割り当て、この数を「順序番号」と呼んでいると想定してください。Ed センターでは、特定のクラスの生徒登録を受け取ったので、クラス内で識別するために各生徒に番号を割り当てます。Ed センターは、この番号を「順序番号」と呼びます。Headquarters と Ed センターは、2 つの別々のデータ・エレメントに同じ名前を使用しています。これは同音異義語 と呼ばれます。 同音異義語の問題を解決するためには、名前を修飾してください。Headquarters が各講習に割り当てる数は、「講習コード」(CRSCODE) と呼び、Ed センターが生徒達に割り当てる番号は、「生徒の順序番号」(STUSEQ#) と呼ぶことができます。

同音異義語
: 2 つの異なったものに対する 1 つの語です。

エレメントを識別しそれを正確に説明するようなデータ・エレメント名を選択してください。データ・エレメント名は、以下のようなものにしてください。

固有なもの
: 他の名前と明らかに区別できる名前

(メッセージで説明されたとおりです**)**
: 容易に理解および認識できる名前

簡潔なもの
: 少ない語で説明できるような名前

汎用的なもの
: すべての人が同じ意味に理解してくれるような名前

アプリケーション・データの文書化

業務処理が必要とするデータ・エレメントがなにかを判別したら、各データ・エレメントについてできるだけ多くの情報を記録します。

この情報は、DBA にとって役立つものです。データの文書化について従うべき標準があれば、それに注意してください。多くの場所では、データについて記録すべき情報およびその情報を記録する方法と場所に関する標準があります。この情報の量とタイプは場所ごとに異なります。以下に、しばしば記録される情報のタイプをリストします。

データ・エレメントの記述名
: データ・エレメント名は正確で、かつそのアプリケーションに精通している人にとっても精通していない人にとっても意味のあるものでなければなりません。

データ・エレメントの長さ
: データ・エレメントの長さにより、セグメント・サイズとセグメント形式を判別します。

文字フォーマット
: プログラマーは、データが英数字か、16 進数か、パック 10 進数か、または 2 進数かを知る必要があります。

エレメントのために使用できる値の範囲
: エレメントのために使用できる値の範囲は、妥当性検査のために重要です。

デフォルト値
: プログラマーにもデフォルト値は必要です。

データ・エレメント・オカレンスの数
: データ・エレメント・オカレンスの数は、DBA がこのデータに必要なスペースを判別する際に役立ちます。また、この情報はパフォーマンスの考慮事項にも影響を与えます。

業務処理がデータ・エレメントに与える影響の程度
: データ・エレメントが読み取られるのか、または更新されるのかにより、アプリケーション・プログラムの PSB にコード化される処理オプションが決まります。

データに関する制御情報も記録する必要があります。このような情報は、次のような問題に関するものです。

- アクセスしようとしたデータが使用できない場合にプログラムがどのような処置を取るべきか。
- 特定のデータ・エレメントの形式が変わったときに、どの業務処理に影響があるか。例えば、研修データベースにはそのデータ・エレメントの 1 つとして各講習ごとに 5 桁のコードがあり、そのコードを 6 桁に変更した場合には、どの業務処理に影響があるか 。
- 現在、そのデータはどこにあるか。アプリケーションに必要な、データ・エレメントのソースを識別してください。
- どの業務処理が特定のデータ・エレメントに変更を行うか。
- アプリケーションのデータについてのセキュリティー要件があるか。例えば、従業員の給料などの情報は、だれでもアクセスできるようにするのは適切ではありません。
- どの部門がデータを所有し、制御しているか。

この情報を収集し、記録する方法の 1 つとしては、次の表に示されているような書式を使用します。 記録するデータの量とタイプは、従うべき標準により異なります。例として、次の表では、ID 番号、データ・エレメント名、長さ、文字形式、使用可能な値、ヌル値、デフォルト値、オカレンスの数をリストします。

表 *11.* データ・エレメント情報形式の例

ID #	データ・エレメント名	長さ	文字形式	使用可能な値	ヌル値	デフォルト値	オカレンスの数
5	講習のコード	5 バイト	16 進数	0010090000	00000	N/A	全カリキュラムには 200 の講習があります。 1 年に平均 10 の授業が新設されたり、改訂されたりします。 1 年に平均 5 の授業が廃止されます。

表 *11.* データ・エレメント情報形式の例 (続き)

ID #	データ・エレメント名	長さ	文字形式	使用可能な値	ヌル値	デフォルト値	オカレンスの数
25	Status	4 バイト	英数字	CONF WAIT CANC	ブランク	WAIT	生徒につき 1 つ
36	生徒名	20 バイト	英数字	英字のみ	ブランク	N/A	1 クラスにつき生徒は 3 人から 100 人で、平均すると、1 クラス 40 人になります。

データ・ディクショナリーは、アプリケーションのデータに関する事実を記録するのに最適な場所です。データを分析するときにディクショナリーを使用すると、特定のデータ・エレメントが既に存在しているかどうかを検出したり、存在している場合にはその特性を検出するのに役立ちます。IBM OS/VS DB/DC データ・ディクショナリーを使用すると、特定のデータベースに存在するセグメントや、そのセグメントに含まれるフィールドをオンラインで判別できます。どのツールを使用しても、同じ情報を含む報告書を作成することができます。

ローカル・ビューの設計

ローカル・ビューは、個々の業務処理に必要なデータを説明しています。

次の情報が含まれます。

- データ・エレメントのリスト
- データ・エレメントを説明するエンティティーによってデータ・エレメントがどのようにグループ分けされるかを示す概念上のデータ構造
- データ・エレメントの各グループ間の関係

定義**:** データ集合 とは、データ・エレメントのグループです。データ・エレメントを説明するエンティティーごとにデータ・エレメントをグループ分けすると、データ集合間の関係を判別することができます。これらの関係は、マッピング と呼ばれます。マッピングに基づいて、業務処理の概念上のデータ構造を設計することができます。 同様にこの処理を文書化してください。

関連概念:

17 ページの『アプリケーション設計の概要』

データ関係の分析

データ関係を分析すると、アプリケーションで扱うビジネス・プロセスの概念的なデータ構造を発展させることができます。

データの構造化 と呼ばれるこのプロセスは、ビジネス・プロセスが必要とするデータ・エレメント間の関係の分析方法のことであり、データベース設計方法のことではありません。セグメント形式および内容についての判断は、DBA に責任があります。ユーザーが発展させる情報こそが、データベースの設計における入力となります。

データの構造化は、多くの異なった方法で行うことができます。

データ・エレメントの階層へのグループ分け

データ集合、生徒を記述するデータ・エレメントは、記述名 STUSEQ#、STUNAME、CUST、LOCTN、STATUS、ABSENCE、および GRADE で表すことができます。このデータ・エレメントのグループを生徒データ集合と呼びます。

データ・エレメントには、名前だけでなく値もあります。生徒のデータ・エレメントを例にすると、値は特定の生徒番号、生徒の名、生徒所属会社、会社の住所、そのクラスにおける生徒の登録状況、この生徒の欠席回数、および評点などになります。データ集合の名前は固有なものではありません。つまり、クラス内のすべての生徒を同じ名前で表します。ただし、データ集合オカレンスを組み合わせた値は固有なものです。2 人の生徒が、これらの各フィールドに同じ値を持つことはありません。

データ・エレメントをデータ集合およびデータ構造にグループ分けする場合には、各グループを作成するデータ・エレメントを参照し、そのグループを固有に識別する 1 つまたは複数のデータ・エレメントを選択してください。 これがデータ集合の制御キー です。これは集合内のデータ・エレメントまたはデータ・エレメントのグループであって、集合を一意的に識別します。ときには、集合を固有に識別するために、キーについての複数のデータ・エレメントを使用しなければならない場合があります。

このセクションで説明するステップに従って、業務処理データの概念的なデータ構造を開発できます。ただし、業務処理を行うプログラムの論理データ構造を開発しているわけではありません。 ステップは次のとおりです。

1. データ集合の単一オカレンス内の繰り返されるデータ・エレメントを分離する。
2. データ集合の複数オカレンス内の重複値を分離する。
3. その制御キーを使用して各データ・エレメントをグループ分けする。

ステップ 1. 繰り返されるデータ・エレメントを分離する

データ集合の単一オカレンスを調べてください。次の表は、クラスの集合の場合に、どのように見えるかを示しています。データ・エレメントは、クラス集合オカレンスとともにリストされています。

表 *12.* クラスの集合の単一オカレンス

データ・エレメント	クラス集合オカレンス
EDCNTR	CHICAGO
DATE(START)	1/14/96
CRSNAME	TRANSISTOR THEORY
CRS CODE	41837
LENGTH	10 DAYS
INSTRS	複数
STUSEQ#	複数
STUNAME	複数
CUST	複数
LOCTN	複数
STATUS	複数

表 12. クラスの集合の単一オカレンス (続き)

データ・エレメント	クラス集合オカレンス
ABSENCE	複数
GRADE	複数

複数として定義された上記のデータ・エレメントは、繰り返されるエレメントのことです。 繰り返されるデータ・エレメントを低いレベルに移動して、それらを分離します。制御キーを使用してデータ・エレメントを保管してください。

単一クラスで繰り返されるデータ・エレメントは、 STUSEQ#、 STUNAME、 CUST、 LOCTN、 STATUS、 ABSENCE、および GRADE です。 INSTRS は、このクラスでは 1 人だけですが 2 人の講師が必要なクラスもあるので、繰り返されるデータ・エレメントになります。

繰り返されるデータ・エレメントを分離してグループ分けすると、次の図に示す構造になります。

図中の各ボックス内のデータ・エレメントは、1 つの集合を形成します。図全体は、データ構造を表しています。 データ・エレメントには、講習の集合、生徒の集合、講師の集合があります。

以下の図では、上述のキーに先行アスタリスク (*) を付けて、これらの集合を示しています。

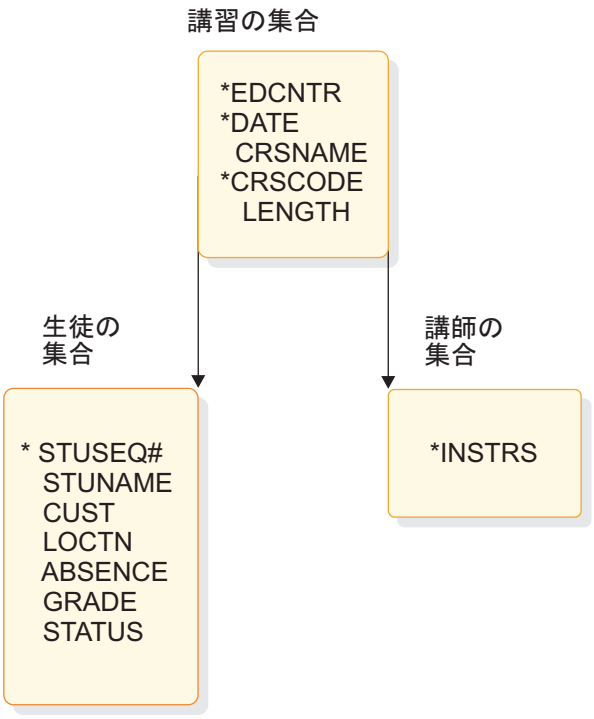


図 5. ステップ 1 の後の現行名簿

データ集合のキーは、以下の表に示されています。

表 *13.* ステップ *1* の後の現行名簿のデータ集合とキー

データ集合	キー
講習の集合	EDCNTR、DATE、CRSCODE
生徒の集合	EDCNTR、DATE、CRSCODE、STUSEQ#
講師の集合	EDCNTR、DATE、CRSCODE、INSTRS

前掲の図中のアスタリスクは、キー・データ・エレメントを示します。クラスの集合では、複数のデータ・エレメントを使用して講習を識別するので、キーを作成するためには複数のデータ・エレメントが必要です。クラスの集合を構成するデータ・エレメントは次のとおりです。

- 制御キー・エレメント STUSEQ#
- STUNAME
- CUST
- LOCTN
- STATUS
- ABSENCE
- GRADE

講師の集合を構成するデータ・エレメントは次のとおりです。

- キー・エレメント INSTRS

生徒の集合と講師の集合は、ルート・セグメントの講習の集合から以下のキーを継承します。

- EDCNTR
- 日付
- CRSCODE

繰り返されるデータ・エレメントを移動した後、各エレメントがその制御キーと同じグループ内にあることを確かめてください。 INSTRS は、講師についての情報が生徒に関する情報と関連付けられていないので、生徒を説明しているデータ・エレメントのグループとは分離されます。生徒の順序番号により、講師がだれであるかは制御されません。

前掲の図で示した例では、生徒の集合と講師の集合は、どちらも講習の集合に従属しています。従属集合のキーには、従属集合の上にあるすべての集合の連結キーが含まれます。 これは、より高位の集合のキーを知らなければ従属集合の制御キーにはなんら意味がないためです。例えば、受講者番号が 4 であることを知っているとすると、その番号に対応したその生徒の全情報を検出できることになります。しかし、この番号が特定の講習と関連付けられていなかった場合には、無意味なものになります。 ただし、生徒の集合のキーは、Ed センター、日付、および講習コードから成り立っているので、生徒が在籍していたクラスを知ることができます。

ステップ2. 重複する集合の値を分離する

集合の複数オカレンスを調べてください。この事例では、2 つのクラスに値を持つことになります。次の表は、同じデータ・エレメントの複数のオカレンス (2 つ) を

示しています。この表にあるように、重複値を確認してください。 両方のオカレンスは 1 つの講習について説明しているということを覚えておいてください。

表 *14.* クラスの集合の複数オカレンス

データ・エレメント・リスト	オカレンス **1**	オカレンス **2**
EDCNTR	CHICAGO	NEW YORK
DATE(START)	1/14/96	3/10/96
CRSNAME	TRANS THEORY	TRANS THEORY
CRSCODE	41837	41837
LENGTH	10 DAYS	10 DAYS
INSTRS	複数	複数
STUSEQ#	複数	複数
STUNAME	複数	複数
CUST	複数	複数
LOCTN	複数	複数
STATUS	複数	複数
ABSENCE	複数	複数
GRADE	複数	複数

複数と定義された上記のデータ・エレメントは、繰り返されるエレメントのことです。 これらのエレメントの値は、同じではありません。 集合は、常に特定なクラスに対して固有なものです。

このステップでは、2 つのオカレンスを比較して、重複値を持つフィールド (TRANS THEORY など) をより高いレベルに移動します。必要なら、まだキーを持っていない集合の制御キーを選択してください。

前掲の表では、CRSNAME、CRSCODE、および LENGTH は、重複値を持つフィールドです。この処理はだいたいにおいて直感的に理解できます。生徒の登録状況および評価には重複した値がありますが、単独では意味を持たないものなので、それらを分離してはなりません。 これらの値は、特別な生徒を識別するために使用するものではありません。 データ・エレメントはその制御キーと共に保管することを思い出せば、このことは明確になります。重複値を分離すると、次の図に示す構造になります。

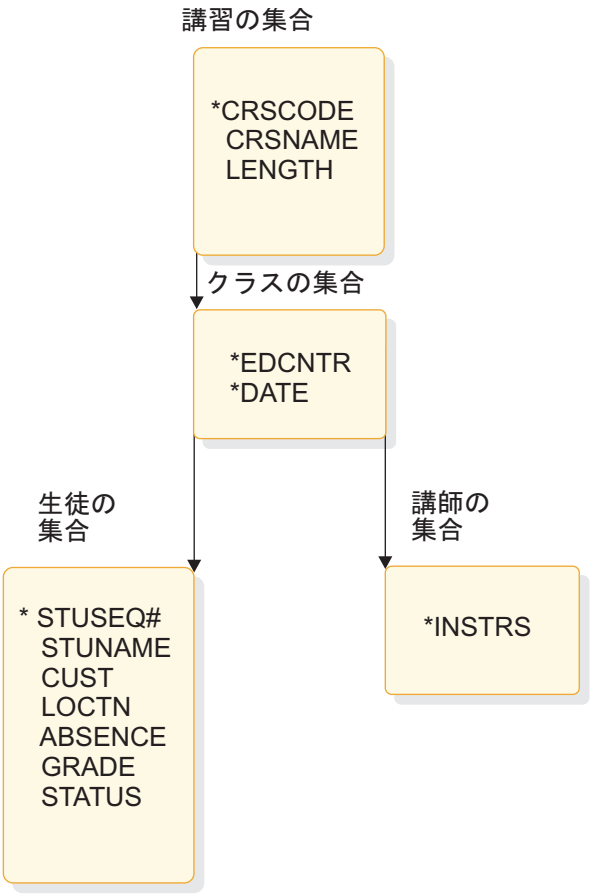


図 *6. ステップ 2 の後の現行名簿*

ステップ 3. 制御キーを使用してデータ・エレメントをグループ分けする

このステップは、最初の 2 つのステップのチェック・ステップとなります。 (このステップで指示されることが、前の 2 つのステップで既に実行済みの場合もあります。)

この段階で、各データ・エレメントが、その制御キーが入っているグループ内にあることを確認してください。データ・エレメントは、フル・キーによって決まらなければなりません。データ・エレメントがキーの部分のみに依存する場合は、データ・エレメントが依存する部分的な (制御) キーと共にデータ・エレメントを分離してください。

この例では、CUST と LOCTN は STUSEQ# に依存しません。これらは生徒に関連しますが、生徒によって決まるものではありません。生徒所属会社とその会社の住所を示すものです。

CUST と LOCTN は講習、Ed センター、または日付のいずれにも依存していません。 これらすべてのこととは分けられています。生徒は 1 つの CUST と LOCTN にだけ関連付けられていますが、CUST および LOCTN にはクラスに出席する多数の生徒を持ち、CUST および LOCTN 集合は、生徒の集合よりも上位になければなりません。

次の図では、これらの集合と、先行アスタリスク (*) で示されるキー、および CUST と LOCTN を分離した際の構造の様子を示しています。

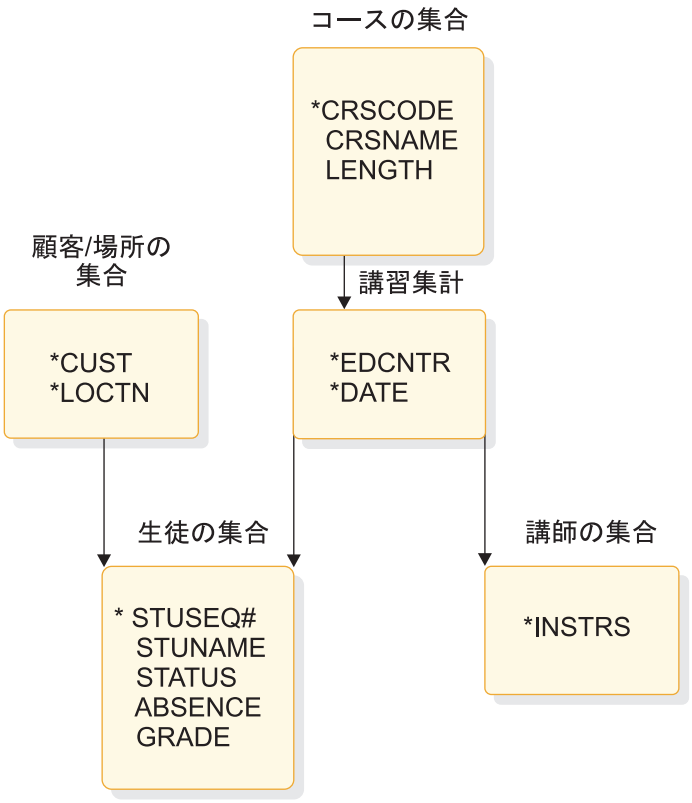


図 7. ステップ 3 の後の現行名簿

データ集合のキーは、以下の表に示されています。

表 15. ステップ 3 の後の現行名簿のデータ集合とキー

データ集合	キー
講習の集合	CRSCODE
クラスの集合	CRSCODE、EDCNTR、DATE
顧客の集合	CUST、LOCTN
生徒の集合	(『ステップ 2 の後の現行名簿』に記載した講習の集合からのビューに代わって、『ステップ 3 の後の現行名簿』に記載した顧客の集合からのビューの場合) CUST、 LOCTN、 STUSEQ、 CRSCODE、 EDCNTR、 DATE
講師の集合	CRSCODE、EDCNTR、DATE、INSTRS

顧客と住所の情報の配置を決めるのは、データベースの設計段階です。データの構造化は、残りのデータ・エレメントと矛盾しているデータ・エレメントを分離しなければなりません。

マッピングの判別

データ集合を概念上のデータ構造に配列した後、データ集合間の関係を検査することができます。 2 つのデータ集合間のマッピングは、2 つの間の量で計ることができる関係です。

マッピングを記録する理由は、開発したデータ構造のセグメント間の関係をマッピングが反映するからです。 IMS データベースにこの情報を保管すると、DBA は、マッピングに基づいて、すべてのローカル・ビューを満たすようなデータベース階層を構成することができます。マッピングを判別する場合には、収集したデータ・エレメントを使用するよりも、データ・エレメントのキーを使用してデータ集合を参照するほうが簡単です。

2 つのデータ集合の間には、以下の 2 つの関係が起こる可能性があります。

- 1 対多数

  セグメント A ごとに、1 つ以上のセグメント B のオカレンスがあります。例えば、クラスごとに1 人以上の生徒がマップされます。

  マッピング表記ではこれを次のように表します。

  クラス ◂——▸▸ 生徒
- 多対多

  セグメント B は、それと関連のある多くのセグメント A を持ちます。セグメント A にはそれと関連のある多数のセグメント B があります。 階層データ構造では、親は 1 つまたは複数の子を持つことができますが、子はそれぞれ 1 つの親とだけ関連付けることができます。 多対多の関連付けでは子はそれぞれ複数の親に関連付けることができるので、多対多の関連付けは階層に適合しません。

  関連資料: データ要件の分析について詳しくは、「*IMS V15* データベース管理」を参照してください。

  多対多の関係は、2 つの業務処理のセグメント間で生じます。多対多の関係は、2 つの業務処理がこれらのデータ集合を処理する必要がある点で、矛盾を示します。 IMS 全機能データベースを使用すると、副次索引または論理関係によりこの種の対立する集合の処理を解決することができます。

現行名簿のマッピングは以下のとおりです。

- 講習 ◂——▸▸ クラス

  各講習では、複数のクラスがスケジュールされることになりますが、1 つのクラスは、1 つの講習とだけ関連付けられます。
- クラス ◂——▸▸ 生徒

  クラスには、登録されている多数の生徒がいますが、1 人の生徒はこの講習を提供する 1 つのクラスだけに存在することができます。
- クラス ◂——▸▸ 講師

  クラスには、1 人または複数の講師がいますが、講師は一度に 1 つのクラスだけを教えます。

- 顧客 / 住所 ◀——▶▶ 生徒

  ある顧客から見ると、ある特定クラスに出席する生徒は複数いますが、各生徒から見ると、1 つの会社 (顧客) と住所に限定されて関連付けられます。

関連概念:

96 ページの『データ構造における矛盾の解決方法についての理解』

ローカル・ビューの例

以下の例では、講習のスケジュール、講師のスキル報告書、および講師のスケジュールを含むローカル・ビューを設計する方法を示しています。

それぞれの例には、ローカル・ビューを設計する以下の 3 つの部分を示しています。

1. データの収集。それぞれの例では、データ・エレメントがリストされていて、データ集合の 2 つのオカレンスが示されています。 2 つのオカレンスが示されるのは、繰り返されるフィールドと重複値を参照する際に両方のオカレンスを参照する必要があるからです。
2. データ関係の分析。 最初に、以下の 3 つのステップを使用して、データ・エレメントを概念上のデータ構造にグループ分けしてください。
   a. データ集合の単一オカレンス内で繰り返されるデータ・エレメントを低位レベルに移動して、これらのデータ・エレメントを分離します。データ・エレメントはそのキーと共に保持してください。
   b. データ集合の 2 つのオカレンスの重複値を、これらのデータ・エレメントを高位レベルに移動して、分離します。この場合も、データ・エレメントはそのキーと共に保持してください。
   c. データ・エレメントをそのキーによりグループ分けします。 1 つの集合内のすべてのデータ・エレメントが同じキーを持っていることを確認します。同じキーを持っていないものは分離します。
3. 作成したデータ構造内でのデータ集合間のマッピングを判別してください。

例 1: 講習のスケジュール

Headquarters では、四半期ごとに行われるすべての講習のスケジュールを管理し、それを毎月配布します。 Headquarters は講習コードでスケジュールを分類し、以下の図に示されているような形式で印刷します。

```
            COURSE SCHEDULE

COURSE:   TRANSISTOR THEORY       COURSE CODE:    418737
LENGTH:   10 DAYS                 PRICE:          $280

DATE                        LOCATION

APRIL 14                    BOSTON
APIRL 21                    CHICAGO
.
.
.
NOVEMBER 18                 LOS ANGELES
```

図 *8.* 講習のスケジュール

1. データの収集。以下の表は、データ集合のデータ・エレメントと 2 つのオカレンスをリストしたものです。

表 *16.* 講習スケジュール・データ・エレメント

データ・エレメント	オカレンス **1**	オカレンス **2**
CRSNAME	TRANS THEORY	MICRO PROG
CRSCODE	41837	41840
LENGTH	10 DAYS	5 DAYS
PRICE	$280	$150
日付	複数	複数
EDCNTR	複数	複数

2. データ関係の分析。 最初に、データ・エレメントを概念上のデータ構造にグループ分けしてください。
   a. このデータ集合の 1 つのオカレンスの中で繰り返しのあるデータ・エレメントを分離します。これを行うには、以下の表に示すように、この繰り返しデータ・エレメントを 1 つ低い階層レベルに移動させます。

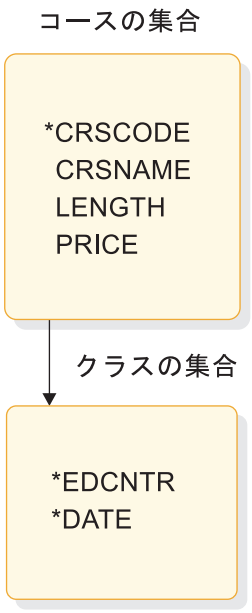


図 *9.* 研修コースのスケジュール *(*ステップ *1* 後*)*

   b. 次に、データ・エレメントを高位レベルに移動して、データ集合の 2 つのオカレンスの重複値を分離します。

      このデータ集合には重複値は含まれていません。
   c. その制御キーを使用してデータ・エレメントをグループ分けしてください。

      データ・エレメントは、現在の構造内ではそのキーによりグループ分けされます。このステップでは、変更は必要ありません。

      データ集合のキーは、以下の表に示されています。

表 *17.* 講習スケジュールのデータ集合とキー *(*ステップ *1* 後*)*

データ集合	キー
講習の集合	CRSCODE
クラスの集合	CRSCODE、EDCNTR、DATE

3. いったん概念上のデータ構造を作成したら、データ集合のマッピングを判別してください。

   このローカル・ビューのマッピングは次のとおりです。 講習 ◄——►► クラス

例 2: 講師のスキル報告書

各 Ed センターは、講師が教える資格を持っている講習を示す報告書を印刷する必要があります。報告書の形式を以下の図に示します。

```
                        INSTRUCTOR SKILLS REPORT

INSTRUCTOR              COURSE CODE             COURSE NAME

BENSON, R. J.           41837                   TRANS THEORY
MORRIS, S. R.           41837                   TRANS THEORY
                        41850                   CIRCUIT DESIGN
                        41852                   LOGIC THEORY
.
.
.
REYNOLDS, P. W.         41840                   MICRO PROG
                        41850                   CIRCUIT DESIGN
```

図 *10.* 講師のスキル報告書

1. データの収集。以下の表は、データ集合のデータ・エレメントと 2 つのオカレンスをリストしたものです。

表 *18.* 講師のスキル・データ・エレメント

データ・エレメント	オカレンス **1**	オカレンス **2**
INSTR	REYNOLDS, P.W.	MORRIS, S. R.
CRSCODE	複数	複数
CRSNAME	複数	複数

2. データ関係の分析。 最初に、データ・エレメントを概念上のデータ構造にグループ分けしてください。
   a. このデータ集合の 1 つのオカレンスの中で繰り返しのあるデータ・エレメントを分離します。これを行うには、以下の図に示すように、この繰り返しデータ・エレメントを 1 つ高い階層レベルに移動させます。

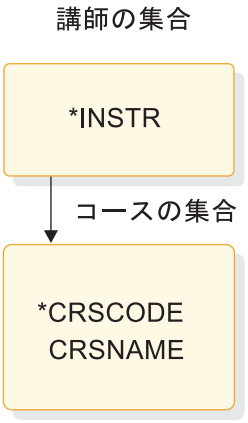


図 *11.* 講師のスキル *(*ステップ *1* 後*)*

b. データ集合の 2 つのオカレンス内の重複値を分離します。

このデータ集合には重複値はありません。

c. データ・エレメントをそのキーによりグループ分けします。

すべてのデータ・エレメントは、現在のデータ構造内ではそれらのキーを使用してグループ分けされます。このデータ構造に対する変更はありません。

3. データ集合のマッピングを判別してください。

このローカル・ビューのマッピングは次のとおりです。講師 ◄——►► 講習

例 3: 講師のスケジュール

Headquarters は、すべての講師のスケジュールを示す報告書を作成する必要があります。以下の図は報告書の形式を示したものです。

```
                        INSTRUCTOR SCHEDULES

INSTRUCTOR          COURSE           CODE     ED CENTER      DATE

BENSON, R. J.       TRANS THEORY     41837    CHICAGO        1/14/96
MORRIS, S. R.       TRANS THEORY     41837    NEW YORK       3/10/96
                    LOGIC THEORY     41852    BOSTON         3/27/96
                    CIRCUIT DES      41840    CHICAGO        4/21/96
REYNOLDS, B. H.     MICRO PROG       41850    NEW YORK       2/25/96
                    CIRCUIT DES      41850    LOS ANGELES    3/10.96
```

図 *12.* 講師のスケジュール

1. データの収集。以下の表は、データ集合のデータ・エレメントと 2 つのオカレンスをリストしたものです。

表 *19.* 講師のスケジュール・データ・エレメント

データ・エレメント	オカレンス **1**	オカレンス **2**
INSTR	BENSON, R. J.	MORRIS, S. R.
CRSNAME	複数	複数
CRSCODE	複数	複数
EDCNTR	複数	複数

表 *19.* 講師のスケジュール・データ・エレメント *(*続き*)*

データ・エレメント	オカレンス **1**	オカレンス **2**
DATE(START)	複数	複数

2. データ関係の分析。 最初に、データ・エレメントを概念上のデータ構造にグループ分けしてください。
   a. 以下の図に示すように、データ集合の 1 つのオカレンス内で繰り返されるデータ・エレメントを低位レベルに移動して、これらのデータ・エレメントを分離します。

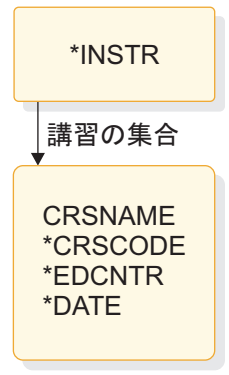


図 *13.* 講師のスケジュール *(*ステップ *1)*

   b. 以下の図に示すように、データ・エレメントを高位レベルに移動して、データ集合の 2 つのオカレンス内の重複値を分離します。

      この例では、CRSNAME および CRSCODE で、1 人または複数の講師が重複しています。例えば、Benson は 41837、Morris と Reynolds は 41850 になります。

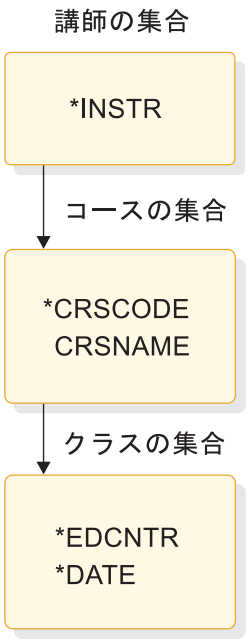


図 *14.* 講師のスケジュール *(*ステップ *2)*

c. データ・エレメントをそのキーによりグループ分けします。

すべてのデータ・エレメントは、現在のデータ構造内ではそれらの制御キーを使用してグループ分けされます。 現在のデータ構造に対する変更は必要ありません。

3. データ集合のマッピングを判別してください。

このローカル・ビューのマッピングは次のとおりです。 講師 ◄———►► 講習 講習 ◄———►► クラス

このセクションで示した 3 つの例の要件を組み合わせるために、またこれらの要件に基づいてデータベースの階層構造を設計するためには、データ要件の分析が必要です。

関連資料 : データ要件の分析の詳細は、「*IMS V15* データベース管理」を参照してください。

第 3 章 IMS アプリケーションの処理要件の分析

以下の情報は、IMS 環境向けのアプリケーション・プログラムの作成を計画するために使用します。

IMS アプリケーション要件の定義

アプリケーション設計の段階として、エンド・ユーザーが実行したい業務処理またはタスクを、どのようにして複数のプログラムに分割すると最も効率的に処理を実行できるのかを決定する必要があります。

処理要件を分析するには、以下のことを考慮してください。

- タスクをいつ実行するのか
  - タスクのスケジュールが予測できない (例えば、端末の要求による) か、あるいは定期的なスケジュール (例えば、毎週) か?
- そのタスクを実行するプログラムがどのように稼働するのか
  - プログラムがオンラインで実行される (応答時間が非常に重要) か、あるいはバッチ・ジョブとして実行依頼される (応答時間が遅くても許容される) か?
- 処理コンポーネントの一貫性
  - プログラムが実行する処理には、複数のプログラム論理タイプが含まれるのか。例えば、タスクの大部分がリトリーブであり、1 つまたは 2 つの更新だけが含まれるとします。その場合は、更新部分を分離して独立したプログラムにすることを考慮すべきです。
  - その処理には、複数の大きなデータ・グループが含まれるか。その場合は、プログラムがアクセスするデータごとにプログラムを分離すると、より効率的です。
- データまたは処理に関する特別要件

  セキュリティー
  : プログラムへのアクセスを制限するかどうか。

  リカバリー
  : プログラム処理において、リカバリーに関する特別な考慮事項があるかどうか。

  可用性
  : ユーザーのアプリケーションに、高いデータ可用性が必要かどうか。

  保全性
  : 他の部門が同一のデータを使用するかどうか。

これらの質問に対して答えることは、処理に必要なアプリケーション・プログラムの数、および最も効率的に処理を実行できるプログラムのタイプを決定することに役立ちます。プログラムの数がいくつであれば最も効率的に必要な処理を行うことができるかについての規則はありませんが、以下にいくつかの提案を示します。

- それぞれのプログラミング・タスクについて、含まれるデータおよび処理を調べてください。 1 つのタスクで、異なるタイプの処理が必要とされ、時間的制限が異なる場合 (例えば、毎週と毎月）、そのタスクは複数のプログラムで実行したほうが効率的です。
- プログラムを定義する際は、プログラムをできるだけ単純にしたほうが、保守およびリカバリーがしやすくなります。 プログラムは単純である (および実行することが少ない) ほど保守がしやすく、プログラムまたはシステムの障害の後の再始動も容易です。同じことは、データの可用性にも当てはまります。すなわち、アクセス対象のデータ量が少なければ、そのデータの可用性は向上します。要求されるアクセスを制限するほど、データは利用しやすくなります。

  同様に、アプリケーションに必要なデータが物理的に 1 つの場所にあると、1 つのプログラムで通常よりも多くの処理を行うことができ、より効率的になる場合があります。 これらは、各アプリケーションの処理およびデータに依存する考慮事項です。
- ユーザーの各作業を文書化しておくと、設計プロセスで役立つばかりでなく、将来他の要員がそのアプリケーションに関連して作業する場合にも役立ちます。この文書化の分野における標準について認識しておく必要があります。文書化して残しておく情報の種類としては、一般的には、作業を実施するタイミング、作業の機能説明、および次の要件 (保守、セキュリティー、リカバリー) です。

  例えば、前述の現行名簿の処理の場合、以下のフォームに示す情報を記録することになります。プログラムが実行される頻度は、Education Center が毎週必要とするクラスの数 (20) によって決まります。

ユーザー・タスク説明の文書化: 現行名簿の例

```
USER TASK DESCRIPTION

NAME: Current Roster
ENVIRONMENT: Batch                    FREQUENCY: 20 per week

INVOKING EVENT OR DOCUMENT: Time period (one week)

REQUIRED RESPONSE TIME: 24 hours

FUNCTION DESCRIPTION: Print weekly, a current student roster, in student
 number sequence for each class offered at the Education Center.

MAINTENANCE:Included in Education DB maintenance.

SECURITY: None.

RECOVERY:After a failure, the ability to start printing a particular
class roster starting from a particular sequential student number.
```

IMS アプリケーション・プログラムを使用したデータベースへのアクセス

プログラムを設計する際には、そのプログラムがアクセスするデータベースのタイプを考慮してください。 データベースのタイプは、操作環境によって異なります。

次の表では、実行可能なプログラム・タイプと、DB バッチ、TM バッチ、DB/DC 環境、DBCTL 環境、または DCCTL 環境でアクセス可能なさまざまなデータベースのタイプを記載しています。

表 20. *IMS* 環境におけるプログラムおよびデータベースのオプション

環境	実行可能なプログラムのタイプ	アクセス可能なデータベースのタイプ
DB/DC	BMP	Db2® for z/OS DEDB および MSDB 全機能 z/OS ファイル
	IFP	Db2 for z/OS DEDB 全機能
	JBP	Db2 for z/OS DEDB 全機能
	JMP	Db2 for z/OS DEDB 全機能
	MPP	Db2 for z/OS DEDB および MSDB 全機能
DB バッチ	DB バッチ	Db2 for z/OS 全機能 GSAM z/OS ファイル
DBCTL	BMP (バッチ指向)	Db2 for z/OS DEDB 全機能 GSAM z/OS ファイル
	JBP	Db2 for z/OS DEDB 全機能
DCCTL	BMP	Db2 for z/OS GSAM z/OS ファイル
	IFP	Db2 for z/OS
	JMP	Db2 for z/OS
	MPP	Db2 for z/OS

表 *20. IMS* 環境におけるプログラムおよびデータベースのオプション *(*続き*)*

環境	実行可能なプログラムのタイプ	アクセス可能なデータベースのタイプ
TM バッチ	TM バッチ	Db2 for z/OS GSAM z/OS ファイル

アクセス可能なデータベースのタイプは、以下のとおりです。

- **IMS** データベース

  IMS データベースには、全機能データベースと高速機能データベースの 2 種類があります。

  – 全機能データベース

    全機能データベースは階層データベースであり、データ言語/I (DL/I) 呼び出しインターフェースを通じてアクセスされ、IFP、JMP、JBP、MPP、BMP、および DB バッチのタイプのアプリケーション・プログラムで処理することができます。 DL/I 呼び出しを使用すると、IMS アプリケーション・プログラムで、全機能データベースのセグメントをリトリーブ、置換、削除、および追加することができます。

    JMP アプリケーションおよび JBP アプリケーションは、 DL/I のほかに JDBC を使用して全機能データベースにアクセスします。

    データ共用を使用する場合は、オンライン・プログラムとバッチ・プログラムが同一の全機能データベースに同時にアクセスできます。

    全機能データベースのタイプには、 HDAM、 HIDAM、 HSAM、 HISAM、 PHDAM、 PHIDAM、 SHSAM、および SHISAM があります。

  – 高速機能データベース

    高速機能データベースには、MSDB と DEDB の 2 つのタイプがあります。

    - 主記憶データベース (MSDB) は、ルート・セグメントのみのデータベースであり、実行時には仮想記憶域に常駐します。
    - 高速処理データベース (DEDB) は、階層データベースであり、多量の明細データについて、高レベルの可用性を提供し、効率的なアクセスを可能にします。

    MPP、BMP、および IFP のプログラムは、高速機能データベースにアクセスできます。 DBCTL 環境では、BMP プログラムは DEDB にアクセスできますが、MSDB にはアクセスできません。 JMP プログラムと JBP プログラムは DEDB にアクセスできますが、MSDB にはアクセスできません。

- **Db2 for z/OS** データベース

  Db2 for z/OS データベースは、IMS バッチ、BMP、IFP、JBP、JMP、および MPP のプログラムで処理可能なリレーショナル・データベースです。 IMS アプリケーション・プログラムは、DL/I データベースのみ、Db2 for z/OS データベースのみ、あるいは DL/I と Db2 for z/OS の両方のデータベースにアクセ

スすることもあります。リレーショナル・データベースは、アプリケーション・プログラムおよびユーザーに対しては、表の形式で表され、構造化照会言語 (SQL) と呼ばれる関係データ言語を使用して処理されます。

注: JMP プログラムと JBP プログラムは Db2 for z/OS データベースにアクセスできません。

関連資料: Db2 for z/OS データベースの処理については、「DB2® for z/OS アプリケーション・プログラミングおよび SQL 解説書」を参照してください。

- **z/OS** ファイル

  BMP (DB/DC 環境、DBCTL 環境、および DCCTL 環境において) は、入力または出力によって z/OS ファイルにアクセスできる、唯一のタイプのオンライン・アプリケーション・プログラムです。バッチ・プログラムでも、z/OS ファイルにアクセスすることができます。

- **GSAM** データベース (汎用順次アクセス方式)

  汎用順次アクセス方式 (GSAM) というアクセス方式を使用すると、BMP およびバッチ・プログラムで、単純データベースとして順次 z/OS データ・セットにアクセスすることが可能になります。 GSAM データベースには、z/OS または IMS によってアクセスすることができます。

データへのアクセス: IMS アプリケーション用に作成できるプログラムのタイプ

使用するプログラムのタイプを決める必要があります。バッチ・プログラム、メッセージ処理プログラム (MPP)、IMS 高速機能 (IFP) アプリケーション、バッチ・メッセージ処理 (BMP) アプリケーション、Java メッセージ処理 (JMP) アプリケーション、または Java バッチ処理 (JBP) アプリケーションのいずれかです。使用可能なプログラムのタイプは、稼働環境がバッチ、DB/DC、または DBCTL のどれであるかによって異なります。

DB バッチ処理

DB バッチ処理に関する以下の説明は、このバッチ・プログラムがご使用のアプリケーションにとって適切であるかどうかを判断するのに役立ちます。

DB バッチ・プログラムがアクセスできるデータ

バッチ・プログラムがアクセスできるのは、全機能データベース、Db2 for z/OS データベース、GSAM データベース、および z/OS ファイルです。 DB バッチ・プログラムは、DEDB や MSDB にはアクセスできません。

DB バッチ処理の使用

バッチ・プログラムは通常、オンライン・プログラムよりも長時間稼働するプログラムです。多量のデータベースを更新する場合や、報告書を印刷する場合にバッチ・プログラムを使用します。バッチ・プログラムは、単独で稼働する (データベースなどのリソースを他のプログラムと競合することがない) ため、制御領域から独立して実行できます。データ共用を使用する場合は、DB バッチ・プログラムと

オンライン・プログラムは全機能データベースに同時にアクセスできます。バッチ・プログラムには、以下のような特徴があります。

- 通常、報告書などの多量の出力を作成します。
- 別のプログラムやユーザーによって実行されることはありません。 バッチ・プログラムは、通常、特定の時間間隔でスケジュールされ（例えば、週ごとに)、JCL を使用して始動します。
- 早急には必要ない出力を作成します。 バッチ出力のターンアラウンド・タイムは、オンライン・プログラムのように重要ではありません。

DB バッチ・プログラムのリカバリー

バッチ・プログラムにチェックポイントを組み込み、障害が起こった場合にプログラムを再起動します。

チェックポイントの発行

バッチ・プログラム内でチェックポイントを出して、データベース変更をコミットし、プログラムを再始動する場所を設けてください。バッチ・プログラムにおいては、MPP、トランザクション指向の BMP、および IFP の場合とは異なり、コミット・ポイントは自動的には起こらないため、バッチ・プログラム内でチェックポイントを出すことは重要です。

チェックポイントを出すことは、オンライン・システムとデータを共用しているバッチ・プログラムでは特に重要です。 チェックポイントによってリソースが開放され、オンライン・プログラムがリソースを使用できるようになります。 作成するすべてのバッチ・プログラムに最初からチェックポイントを組み込んでおく必要があります。 その時点ではチェックポイントのサポートが必要ないかもしれませんが、後でチェックポイントを組み込むよりも、あらかじめ組み込んでおいたほうが簡単です。 また、バッチ・プログラムから BMP に変換したり、データを共用したい場合があるかもしれません。

チェックポイント (あるいはその他のシステム・サービス呼び出し) を出す場合、プログラムに I/O PCB を指定する必要があります。 I/O PCB を得るには、プログラムの PSB 内で、 PSBGEN ステートメントに CMPAT=YES を指定して、互換性オプションを使用します。

GSAM DB はバックアウトされませんが、XRST 呼び出しを介した BMP の再始動プロセス中に再配置されます。XRST 呼び出しは、データ・セット・ポインターを呼び出しで指定されたチェックポイント ID に再配置します。アプリケーションは、起動されると、そのポイントから開始されて先へ進みます。XRST 呼び出しで指定されたチェックポイント ID は、動的バックアウトまたはバッチ・バックアウトのいずれかによる非 GSAM DB のバックアウト先と同じ ID であることが必要です。

推奨事項: DB バッチ・プログラムが使用する PSB では、必ず CMPAT=YES を指定してください。

データベース変更のバックアウト

DB バッチ・プログラムが異常終了した場合にどのような事態が起こるかは、システム・ログのためのストレージ・メディアのタイプによって異なります。 DASD (直接アクセス・ストレージ・デバイス) またはテープのいずれかにシステム・ログを保管するように指定できます。

DASD へのシステム・ログ

システム・ログを DASD に記録する場合は、BKO 実行パラメーターを使用して、プログラムが最後のコミット・ポイント以降にデータベースに行った変更を、IMS が動的にバックアウトするように指定できます。

関連資料 **:** BKO 実行パラメーターの使用については、「*IMS V15* システム定義」を参照してください。

データベース変更を動的にバックアウトすると、次のような利点があります。

- 障害が起きたプログラムによってアクセスされたデータを、すぐに他のプログラムで使用することができます。バッチ・バックアウトを使用した場合は、IMS バッチ・バックアウト・ユーティリティーが実行されてデータベース変更がバックアウトされるまで、他のプログラムはデータにアクセスできません。
- データを共用している 2 つのプログラムがデッドロックされた場合、そのうちの 1 つは処理を続行することができます。 動的バックアウトではなく、バッチ・バックアウトを使用した場合は、プログラムは両方とも失敗します。

例えば、デッドロックが検出された場合のように、IMS が障害を検出した場合、IMS はバッチ・プログラムに対して動的バックアウトを行います。DASD へのロギングを行うことにより、バッチ・プログラムが SETS、 ROLB、および ROLS システム・サービス呼び出しを出すことが可能になります。これらの呼び出しが行われると、IMS はプログラムが行った変更を動的にバックアウトします。

関連資料**:** SETS、ROLB、および ROLS の呼び出しについては、「*IMS V15* データベース管理」にある、データベースのリカバリーおよびデータベース保全性の維持に関する情報を参照してください。

テープへのシステム・ログ

バッチ・アプリケーション・プログラムが異常終了し、バッチ・システム・ログがテープに保管されている場合には、IMS バッチ・バックアウト・ユーティリティーを使用して、プログラムによるデータベースへの変更をバックアウトする必要があります。

関連概念:

58 ページの『チェックポイント呼び出しの使用が重要な場合』

TM バッチ処理

TM バッチ・プログラムは、DB バッチ・プログラムと同様に機能しますが、次のような相違点があります。

- TM バッチ・プログラムは全機能データベースにはアクセスできませんが、Db2 for z/OS データベース、GSAM データベース、および z/OS ファイルにはアクセス可能です。

- リカバリーのためのチェックポイントを設定する際に、プログラムの PSB に CMPAT=YES を指定する必要はありません。 (TM バッチ・プログラムでは、CMPAT パラメーターは無視されます。) I/O PCB は常にリスト内で最初の PCB になります。
- IMS はデータベースを所有しないため、データベースを動的にバックアウトすることはできません。

DB2 for z/OS などの他の外部サブシステムとのログ同期を有効にするには、IEFRDER ログ DD ステートメントが必要です。

メッセージ処理: メッセージ処理プログラム

メッセージ処理プログラム (MPP) はオンライン・プログラムであり、全機能データベース、DEDB、MSDB、および Db2 for z/OS データベースにアクセスすることができます。 BMP やバッチ・プログラムとは異なり、MPP は GSAM データベースにはアクセスできません。 MPP は、DB/DC 環境および DCCTL 環境でのみ実行することができます。

MPP の使用

MPP の主要な目的は、端末のユーザーや他のアプリケーション・プログラムからの要求を処理することです。 MPP はきわめて小さく、素早く要求を処理し、応答するように調整されているのが理想的です。入力としてメッセージを処理し、応答としてメッセージを送信します。

メッセージ
: 任意の 2 つの端末装置、アプリケーション・プログラム、または IMS システム間で伝送されるデータ。各メッセージには、1 つまたは複数のセグメントがあります。

MPP はトランザクション・コードを通して実行されます。 MPP を定義する際は、MPP を 1 つまたは複数のトランザクション・コードと組み合わせてください。各トランザクション・コードはそれぞれ、MPP が処理するトランザクションを表します。トランザクションを処理する場合、端末のユーザーがそのトランザクションに対するコード入力します。そうすると IMS は、入力されたコードに関連する MPP をスケジュールし、 MPP はそのトランザクションを処理します。その MPP は処理を行うために、データベースへのアクセスが必要になる場合があります。通常は、MPP は以下の 5 つのステップを行ってトランザクションを処理します。

1. IMS からのメッセージをリトリーブします。
2. メッセージを処理し、必要であればデータベースにアクセスします。
3. メッセージに応答します。
4. メッセージが無くなるまで処理を繰り返します。
5. 終了します。

MPP を定義する場合、システム管理者はプログラムのスケジューリングと処理に関する決定を行います。 それぞれの MPP に対して、システム管理者は以下のことを指定します。

- トランザクションの優先度

- 特定のトランザクション・コードが入力された場合に、単一のスケジューリングで MPP が処理できるメッセージの数
- MPP が 1 つのトランザクションを処理するために与えられた時間の総計 (秒単位)

優先度および処理の限界を定義することによって、システム管理において、ロード・バランシングおよび処理を制御することができます。

MPP の主要な目的は、メッセージを迅速に処理し応答することですが、 MPP は、トランザクションの処理のしかた、および出力メッセージの送信先に関して柔軟性があります。 例えば、MPP は出力メッセージを他の端末やアプリケーション・プログラムに送信することができます。

関連概念:

87 ページの『第 5 章 データベース・オプションの要件収集』

メッセージ処理: IMS 高速機能プログラム

IMS 高速機能プログラム (IFP) は、MPP と似ています。その主な目的は、端末からのメッセージを迅速に処理して応答することです。 MPP と同様に、IFP は全機能データベース、DEDB、MSDB、および Db2 for z/OS データベースにアクセスすることができます。 IFP は、DB/DC 環境および DCCTL 環境でのみ稼働することができます。

IFP の使用

早急な処理が必要で、IFP に関連する特性および制約が許容できる場合は、 IFP を使用してください。

IFP と MPP の主な違いは次のとおりです。

- IFP で処理するメッセージは 1 つのセグメントだけで構成されていなければなりません。 MPP で処理するメッセージは、複数のセグメントで構成されていてもかまいません。
- IFP は、処理をより効率的にするために、IMS キューイングをバイパスします。高速機能の EMH (急送メッセージ・ハンドラー) によって処理されるトランザクションは、先入れ先出し法で処理されます。

IFP には次のような特性もあります。

- トランザクション応答モードで稼働します。 このため、メッセージを送信した端末がそれ以上要求を出す前に、その端末に応答しなければなりません。
- 入力待ちトランザクションだけを処理します。 入力待ちトランザクションを処理するようにプログラムを定義した場合は、処理するメッセージがなくなっても、プログラムは仮想記憶装置内に存在します。

制約事項:

- IMS プログラムが IFP トランザクションにメッセージを送信するためには、その IFP トランザクションが、システム間連絡 (ISC) を使用して接続された他の IMS システム内になければなりません。
- MPP は、IFP トランザクションに会話を渡すことができません。

IFP のリカバリー

IFP は、単一モードとして定義しなければなりません。 そうすると、プログラムがメッセージをリトリーブするたびにコミット・ポイントが発生します。 そのため、チェックポイント呼び出しを出す必要はありません。

バッチ・メッセージ処理: BMP

BMP はアプリケーション・プログラムであり、バッチ・タイプ処理をオンラインで実行したり、入力用および出力用 IMS メッセージ・キューにアクセスすることができます。この点においても、使用できるデータにおいても、 BMP は IMS アプリケーション・プログラムのなかで、最も柔軟性があります。 BMP には、バッチ指向とトランザクション指向の 2 つのタイプがあります。

オンラインでのバッチ処理: バッチ指向 BMP

バッチ指向 BMP は、どのようなオンライン環境でもバッチ・タイプ処理を実行します。 DB/DC または DCCTL 環境で稼働する場合、バッチ指向 BMP は、後で別のアプリケーション・プログラムが処理するように、その出力を IMS メッセージ・キューに送信することができます。トランザクション指向 BMP とは異なり、バッチ指向 BMP は、入力用 IMS メッセージ・キューにアクセスすることはできません。

バッチ指向 BMP がアクセスできるデータ

DBCTL 環境では、バッチ指向 BMP は全機能データベース、Db2 for z/OS データベース、DEDB、z/OS ファイル、および GSAM データベースにアクセスすることができます。 DB/DC 環境では、バッチ指向 BMP は、これらすべてのタイプのデータベースと、同様に高速機能 MSDB にアクセスすることができます。 DCCTL 環境では、このプログラムは Db2 for z/OS データベース、z/OS ファイル、および GSAM データベースにアクセスすることができます。

バッチ指向 BMP の使用

バッチ指向 BMP は、オンラインで稼働するバッチ・プログラムとしても使用できます。 (オンライン要求は、バッチ・システムではなく、IMS DB/DC、DBCTL、または DCCTL システムによって処理されます。) 同じプログラムを、BMPとしても、バッチ・プログラムとしても実行することができます。

推奨事項: プログラムがチェックポイントを指定せず大量のデータベース更新を実行する場合は、オンライン・システムのパフォーマンスを低下させないようにバッチ・プログラムとして実行することを考慮します。

バッチ指向 BMP を最も効率的に使用するため、オンラインでの多量のバッチ・タイプ処理は避けてください。 BMP で、報告書作成やデータベース走査などの時間のかかる処理を行う場合は、処理のピーク時を避けてスケジュールしてください。そうすることで、MPP の応答時間の遅延を防ぐことができます。

BMP は応答時間を遅らせることがあるので、バッチ・メッセージ処理を使用する範囲を決定する際には、応答時間の要件が主な考慮事項となります。 BMP の使用は、その結果に応じて決定してください。

バッチ指向 BMP のリカバリー

バッチ指向 BMP で処理を行う場合は、MPP、トランザクション指向 BMP および IFP の場合とは異なり、コミット・ポイントは自動的に起こらないため、チェックポイント呼び出しを出すことが重要になります。 大部分のバッチ・プログラムとは異なり、BMP は MPP とリソースを共用します。 （バッチ・プログラムの場合と同様に）データベース変更をコミットしたり、再始動する位置を設けたりする以外に、チェックポイントは、プログラムに対してロックされたリソースを解放します。

バッチ指向 BMP に障害が起こった場合、IMS および Db2 for z/OS は、最後のコミット・ポイント以降にプログラムが行ったデータベース更新をバックアウトします。その後で、JCL を使用してプログラムを再始動してください。 BMP で z/OS ファイルを処理する場合は、チェックポイント設定および再始動の方式をユーザーが作成しなければなりません。

バッチ・プログラムのバッチ指向 BMP への変換

IMS TM を使用しているか DBCTL 環境で稼働している場合、バッチ・プログラムをバッチ指向 BMP に変換することができます。

- IMS TM を使用する場合、バッチ・プログラムを変換すると以下のような利点があります。
  - BMP は、メッセージ・キューに出力を送信することができます。
  - BMP は、DEDB および MSDB にアクセスすることができます。
  - ロギングは単一システム・ログに記録されるため、BMP はプログラム・リカバリーを単純化できます。 システム・ログのための DASD をバッチで使用する場合は、プログラムに動的バックアウトを指定できます。その場合は、バッチ・リカバリーは BMP リカバリーに似ていますが、バッチではユーザーは複数のログを管理する必要があります。
  - JCL を変更しなくても、最後のチェックポイントから自動的に再始動することができます。
- DBCTL を使用している場合、プログラムを以下の理由によって変換したいことがあります。
  - BMP は、DEDB にアクセスすることができます。
  - ロギングは単一システム・ログに記録されるため、BMP はプログラム・リカバリーを単純化できます。 システム・ログのための DASD をバッチで使用する場合は、プログラムに動的バックアウトを指定できます。その場合は、バッチ・リカバリーは BMP リカバリーに似ていますが、バッチではユーザーは複数のログを管理する必要があります。
- シスプレックス・データ共用を実行中で、IMS TM を使用するか DBCTL を使用中の場合には、プログラムを変換したいことがあります。これは、バッチ指向 BMP を使用すると、それぞれの OSAM または VSAM 構造について、32 個の接続を限度とするシスプレックス・データ共用を続けることが可能になるためです。

データ共用を使用する場合は、バッチ・プログラムをオンライン・プログラムと並行して実行できます。データ共用を使用しない場合は、バッチ・プログラムを BMP に変換すると、そのバッチ・プログラムを BMP およびその他のオンライン・プログラムと共に実行できます。

また、バッチ・プログラムをオフラインで実行することを計画している場合には、バッチ・プログラムを BMP に変換することによって、オンライン・システムが稼働しなくなるまで待機せずに、オンライン・システムでプログラムを実行することが可能になります。バッチ・プログラムを BMP として実行すると、データをより最新の状態に保つことができます。

- IMS TM を使用する場合、または DBCTL を使用している場合には、プログラムをバッチ・プログラムとしても BMP としても稼働することができます。

  推奨事項: チェックポイントのコーディングは、修正が容易にできるような方法で行ってください。バッチ・プログラムを BMP に変更したり、データを共用するように変換するには、より頻繁なチェックポイントが必要です。また、バッチ領域で稼働中にプログラムに障害が起こった場合は、バッチ領域で再始動しなければなりません。 BMP 領域でプログラムに障害が起きた場合は、BMP 領域で再始動しなければなりません。

バッチ・プログラムを BMP に変換するための要件は、以下のとおりです。

- プログラムに I/O PCB がなければなりません。 プログラムのプログラム仕様ブロック (PSB) 内に互換性 (CMPAT) オプションを指定すると、バッチ・プログラムで I/O PCB を獲得することができます。

  関連資料: PSB の CMPAT オプションの詳細については、「*IMS V15* システム・ユーティリティー」を参照してください。
- BMP では、バッチ・プログラムよりも頻繁にチェックポイント呼び出しを出さなければなりません。

関連概念:

58 ページの『チェックポイント呼び出しの使用が重要な場合』

バッチ・メッセージ処理: トランザクション指向 BMP

トランザクション指向 BMP は、z/OS ファイル、GSAM データベース、 Db2 for z/OS データベース、全機能データベース、DEDB および MSDB にアクセスすることができます。

トランザクション指向 BMP がアクセスできるデータ

バッチ指向 BMP とは異なり、トランザクション指向 BMP は、入力、出力の両方で IMS メッセージ・キューにアクセスでき、DB/DC と DCCTL 環境でのみ実行可能です。

トランザクション指向 BMP の使用

MPP とは異なり、トランザクション指向 BMP は IMS によってスケジュールされません。必要に応じてスケジュールし、JCL を使用して始動してください。例えば、MPP はメッセージを処理する際に、トランザクションの詳細を含む出力メッセ

ージをメッセージ・キューに送信することができます。 その場合、トランザクション指向 BMP は、メッセージ・キューにアクセスして、毎日のアクティビティー報告書を作成することができます。

トランザクション指向 BMP の典型的な使用法は、オンラインでの直接更新をシミュレートすることです。トランザクションの処理中にデータベースを更新する代わりに、MPP はメッセージ・キューに更新を送ります。すると、トランザクション指向 BMP が、MPP の代わりにその更新を実行します。 ユーザーは、更新の数により、必要に応じて BMP を実行することができます。 それによって、MPP が要する応答時間が短縮され、データを最新に保つことができます。 MPP の応答時間が重要な場合には、MPP でトランザクションを処理するよりも効率的です。この方法の欠点は、トランザクションを分割する必要がない場合でも 2 つの部分に分割してしまうことです。

BMP に MPP の更新を実行させる場合は、BMP が異常終了した場合に再始動するときに、最後のメッセージを BMP への入力としてユーザーが再入力できるように、BMP を設計してください。 例えば、MPP がデータベース更新を 3 つの BMP に処理させる場合に、BMP の中の 1 つが異常終了したとします。ユーザーは、異常終了した BMP が処理していたメッセージを、他の BMP に再入力して処理する必要があります。

BMP は、入力待ち (WFI) として定義されたトランザクションを処理することができます。 つまり、IMS は、有効な入力メッセージを処理してしまった後も、BMP を仮想記憶装置内に残します。 以下のいずれかが発生した場合、IMS は QC 状況コードを戻して、プログラムを終了すべきことを示します。

- プログラムが制限時間に達した場合
- マスター端末オペレーターが、コマンドを入力して処理を停止した場合
- IMS がチェックポイント・シャットダウンで終了した場合

トランザクションについて WFI を、 IMS システム定義の際に、TRANSACT マクロの WFI パラメーターで指定します。

WFI トランザクションに対してスケジュールされたバッチ・メッセージ処理領域 (BMP) は、/PSTOP REGION、/DBD、/DBR、または /STA コマンドの場合にのみ QC 状況コードを (メッセージなしで) 戻します。

MPP の場合と同様に、BMP は、他のアプリケーション・プログラムを含む複数の宛先に出力メッセージを送信することができます。

トランザクション指向 BMP のリカバリー

MPP の場合と同様に、トランザクション指向 BMP を使用すると、プログラムのコミット・ポイントが起こる地点を選択することができます。 MPP の場合と同様に、トランザクション指向 BMP を単一モードまたは複数モードのどちらに指定することもができます。 BMP が単一モードの場合は、チェックポイント呼び出しは、複数モード BMP の場合ほど重大ではありません。単一モード BMP では、プログラムがメッセージをリトリーブするたびに、コミット・ポイントが起こります。

関連概念:

124 ページの『出力メッセージの宛先の識別』
58 ページの『チェックポイント呼び出しの使用が重要な場合』

Java メッセージ処理: JMP

JMP アプリケーション・プログラムは MPP によく似ていますが、JMP アプリケーションは Java またはオブジェクト指向 COBOL で作成しなければならないという相違があります。 JMP アプリケーションは、MPP と同様に、JMP アプリケーション用のメッセージ・キューがあって、IMS がメッセージを処理のためにスケジュールに入れる場合に開始します。

JMP アプリケーションは JDBC を使用して IMS データまたは Db2 for z/OS データにアクセスできます。JMP アプリケーションは、JVM (Java 仮想マシン) を含む JMP 領域で実行されます。

関連概念:

825 ページの『IMS Java 従属領域の概要』

Java バッチ処理: JBP

JBP アプリケーション・プログラムは非メッセージ・ドリブン BMP アプリケーション・プログラムによく似ていますが、JBP アプリケーションは Java、オブジェクト指向 COBOL、またはオブジェクト指向 PL/I で作成しなければならないという相違があります。

JBP アプリケーションは JDBC を使用して IMS データまたは Db2 for z/OS データにアクセスできます。JBP アプリケーションは、JBP 領域で実行され、その領域には JVM が存在します。

関連概念:

825 ページの『IMS Java 従属領域の概要』

IMS プログラミングの保全性およびリカバリーの考慮事項

IMS では、アプリケーション・プログラムに対するデータ保全性の保護がサポートされています。

IMS がデータの保全性を保護する方法: コミット・ポイント

オンライン・プログラムがデータベースにアクセスする場合は、必ずしもそのプログラムだけがアクセスしているとは限りません。 IMS および Db2 for z/OS を使用すると、同時に複数のアプリケーション・プログラムがデータにアクセスすることができ、データの保全性を損なう危険もありません。

データ保全性を保護しながら並行してデータにアクセスするために、IMS と Db2 for z/OS は、プログラムがコミット・ポイント に達するまでに削除、置換、または挿入するセグメントへ、他のアプリケーション・プログラムがアクセスするのを防ぎます。コミット・ポイントとは、プログラムの処理の中でプログラムが 1 つの作業単位を完了する個所です。 1 つの作業単位が完了すると、IMS と Db2 for z/OS は、プログラムがデータベースに行った変更をコミットします。これで、これらの変更が永続的となり、変更データが他のアプリケーション・プログラムに使用可能になります。

コミット・ポイントで起きる事柄

アプリケーション・プログラムが 1 つの作業単位の処理を終了すると、IMS および Db2 for z/OS は、後でプログラムに問題が発生したとしても、その処理を有効なものと見なします。例えば、ある端末からの 1 つのメッセージに対する、アプリケーション・プログラムによる一連のリトリーブ、処理、および応答は、1 つの作業単位を構成します。 プログラムが次の入力メッセージを処理している間に問題が起きても、前の入力メッセージによって行われた処理には影響はありません。これらの入力メッセージはそれぞれ、独立した処理単位です。

コミット・ポイントによって、プログラムが 1 つの作業単位を終了したこと、およびその処理が正確に行われたことが、IMS に通知されます。 その際、以下のことが行われます。

- IMS は、最後のコミット・ポイント以降にプログラムに対してロックされたセグメントを解放します。その後、それらのセグメントが他のアプリケーション・プログラムで利用できるようになります。
- IMS および Db2 for z/OS は、データベースに対してプログラムが行った変更を永続的にします。
- GSAM を除くすべてのデータベース内での現在位置は、データベースの最初にリセットされます。

プログラムがコミット・ポイントに到達する前に異常終了した場合は、以下のことが行われます。

- IMS および Db2 for z/OS は、最後のコミット・ポイント以降にプログラムがデータベースに加えた変更をバックアウトします。 (これは、テープにログを行うバッチ・プログラムには適用されません。)
- IMS は、最後のコミット・ポイント以降にプログラムが作成した出力メッセージをすべて廃棄します。

  プログラムがコミット・ポイントに到達するまで、IMS はプログラムの出力メッセージを保留するので、プログラムが異常終了した場合でも、端末のオペレーターや他のアプリケーション・プログラムが、異常終了したプログラムからの不正確なメッセージを受け取ることはありません。

  プログラムが入力メッセージの処理中に異常終了しても、次の 2 つの条件が存在する場合は、入力メッセージは廃棄されません。

  1. 廃棄不能メッセージ (NDM) 出口ルーチンを使用していない。
  2. IMS が異常終了コード U0777、U2478、U2479、U3303 のいずれかを出して、プログラムを終了させる。入力メッセージは保管され、後で処理されます。

     例外: プログラムが上記の異常終了コードで終了しなかった場合は、入力メッセージは廃棄されます。プログラムを再始動すると、IMS はプログラムに次のメッセージを送ります。

  プログラムが入力メッセージの処理中に異常終了しても、NDM 出口ルーチンが使用されている場合は、異常終了したかどうかにかかわらず、入力メッセージは

システムから廃棄される可能性があります。入力メッセージがシステムから廃棄されるかどうかは、NDM 出口ルーチンがどのように書かれているかで決まります。

関連資料: NDM 出口ルーチンについて詳しくは、「*IMS V15* 出口ルーチン」を参照してください。

- IMS は、プログラムが異常終了したことを MTO に通知します。
- IMS および Db2 for z/OS は、プログラムが最後のコミット・ポイント以降に更新したデータに行っていたロックをすべて解除します。それによって、他のアプリケーション・プログラムおよびユーザーがそのデータを使用できるようになります。

コミット・ポイントが発生する地点

次のいずれかの理由により、プログラム内にコミット・ポイントが発生することがあります。

- プログラムが正常に終了した場合。高速機能リソースにアクセスするプログラムの場合を除き、通常のプログラム終了はすべてコミット・ポイントになります。高速機能リソースをアクセスするプログラムは、終了の前にコミット・ポイントに到達しなければなりません。
- プログラムがチェックポイント呼び出しを出した場合。チェックポイント呼び出しは、プログラムが処理中にコミット・ポイントに到達したことを、プログラムから IMS に対して明示的に指示する手段です。
- プログラムがメッセージを入力として処理する場合、プログラムが新しいメッセージを読み込むと、コミット・ポイントが起こる場合があります。 IMS は、このコミット・ポイントをプログラムの新しい作業単位の始まりと見なします。新しいメッセージをリトリーブすると必ずしもコミット・ポイントが起こるとは限りません。コミット・ポイントが起こるかどうかは、プログラムが単一モードに定義されているか、複数モードに定義されているかによって異なります。
  - 単一モードを指定している場合は、プログラムが新しいメッセージをリトリーブするための呼び出しを出す度に、コミット・ポイントが起こります。単一モードを指定すると、プログラムが異常終了した場合に、最後の新規メッセージ呼び出しの位置からプログラムを再始動できるので、リカバリーを単純化することができます。 IMS がプログラムを再始動する場合、次のメッセージを処理することによってプログラムが始動します。
  - 複数モードを指定している場合、プログラムがチェックポイント呼び出しを出した場合、およびプログラムが正常に終了した場合にコミット・ポイントが起こります。このような場合、 IMS はプログラムの出力メッセージをその宛先へ送信します。複数モード・プログラムには、含まれているコミット・ポイントが単一モード・プログラムの場合より少ないので、複数モード・プログラムは単一モード・プログラムよりパフォーマンスが多少よくなる場合があります。複数モード・プログラムが異常終了した場合、 IMS は、チェックポイントからのみプログラムを再始動できます。いつプログラムが最後のチェックポイント呼び出しを出したかによって、プログラムは、最後に出されたメッセージだけではなく、複数のメッセージを再処理することがあります。

次の表では、プログラムを実行できるモードをリストしています。処理モードは、バッチ・プログラムおよびバッチ指向 BMP には不適当なので、表にはリストしていません。この表では、プログラム・タイプをリストし、どのモードがサポートされているかを示しています。

表 *21.* 処理モード

プログラム・タイプ	単一モードのみ	複数モードのみ	両モード
MPP			X
IFP	X		
トランザクション指向 BMP			X

TRANSACT マクロの MODE パラメーターに、単一モードまたは複数モードを指定してください。

関連資料 :TRANSACT マクロの詳細については、「*IMS V15* システム定義」を参照してください。

単一モード・プログラムと複数モード・プログラムにおける相違点の例については、次の図を参照してください。単一モード・プログラムは、メッセージを取得して処理した後、出力を送信し、さらにメッセージを検索して、それ以上なければ終了します。複数モード・プログラムは、メッセージを取得して処理し、出力を送信しますが、次に、チェックポイントを設けてから、さらにメッセージを検索して終了します。単一モード・プログラムの場合、コミット・ポイントは、メッセージが取得される時点およびプログラムが終了する時点です。複数モードの場合、コミット・ポイントは、チェックポイントとプログラムが終了する時点です。

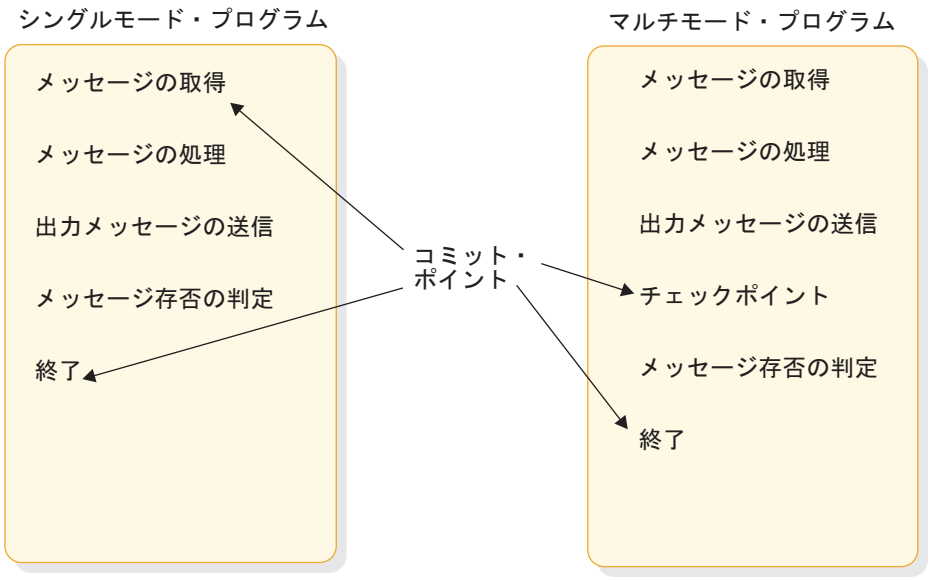


図 *15.* 単一モードと複数モード

Db2 for z/OS は、複数モード・プログラムおよび単一モード・プログラムで、IMS が行わない処理をいくつか行います。複数モード・プログラムが新しいメッセージをリトリーブするための呼び出しを出すと、Db2 for z/OS は許可検査を行い

ます。許可検査が成功すると、Db2 for z/OS は、オープンしているすべての SQL カーソルをクローズします。これは、ユーザーのプログラムの設計に影響を与えます。

Db2 for z/OS SQL COMMIT ステートメントによって、Db2 for z/OS はデータベースに対して永続的な変更を行います。しかし、このステートメントは、TSO アプリケーション・プログラムでのみ有効です。 IMS アプリケーション・プログラムがこのステートメントを出した場合は、否定の SQL 戻りコードを受け取ります。

プログラム・リカバリーの計画: チェックポイントおよび再始動

Db2 for z/OS データにアクセスする IMS アプリケーション・プログラムでは、リカバリーは IMS および Db2 for z/OS によって処理されます。 IMS は処理を調整し、Db2 for z/OS は Db2 for z/OS データのリカバリーを実施します。

関連概念:

『チェックポイント呼び出しの概要』

58 ページの『チェックポイント呼び出しの使用が重要な場合』

61 ページの『チェックポイントの頻度の指定』

チェックポイント呼び出しの概要

チェックポイント呼び出しは、プログラムがコミット・ポイントに到達したことを IMS に通知します。チェックポイント呼び出しによって、プログラム内にプログラムが再始動できる地点を設けることができます。 IMS には、シンボリック・チェックポイント呼び出しと基本チェックポイント呼び出しがあります。

チェックポイント呼び出しは、1 つのプログラムにつき 1 種類だけ出すことができます。

- MPP および IFP では、基本チェックポイント呼び出しを使用しなければなりません。
- BMP、JMP、およびバッチ・プログラムでは、シンボリック・チェックポイント呼び出しまたは基本チェックポイント呼び出しのどちらでも使用することができます。

シンボリック・チェックポイント呼び出しを出すプログラムは、プログラム内に最大 7 つのデータ域にチェックポイントを設定するように指定することができます。IMS がプログラムを再始動する場合、再始動呼び出しによって、これらの領域は、プログラムがシンボリック・チェックポイント呼び出しを出した時点の状態に復元されます。シンボリック・チェックポイント呼び出しは z/OS ファイルをサポートしないので、プログラムが z/OS ファイルをアクセスする場合には、ユーザーが自分でチェックポイントの設定方法を指定しなければなりません。

シンボリック・チェックポイントは、通常始動または拡張再始動 (XRST) のいずれの場合でも使用することができます。

例えば、通常始動の場合の標準的な呼び出しは、以下のようになります。

- XRST (入出力域はブランク)
- CHKP (入出力域にはチェックポイント ID がある)
- データベース呼び出し (チェックポイントを含む)

- CHKP (最終チェックポイント)

例えば、拡張再始動 (XRST) の場合の標準的な呼び出しは、以下のようになります。

- XRST (入出力域にはチェックポイント ID がある)
- CHKP (入出力域には新規のチェックポイント ID がある)
- データベース呼び出し (チェックポイントを含む)
- CHKP (最終チェックポイント)

再始動呼び出しは、シンボリック・チェックポイント呼び出しの場合に使用しなければなりません。再始動呼び出しによって、異常終了後にプログラムを再始動することができます。再始動呼び出しは、プログラムのデータ領域をプログラムがシンボリック・チェックポイント呼び出しを出した時点の状態に復元します。また、異常終了する前にプログラムが設定した最後のチェックポイントからプログラムを再始動します。

基本チェックポイント呼び出しは、あらゆるプログラムで使用することができます。 基本チェックポイント呼び出しとともに再始動呼び出しを使用することはできないので、ユーザーがプログラムを再始動しなければなりません。 基本チェックポイント呼び出しは、z/OS ファイルおよび GSAM ファイルをサポートしません。IMS プログラムは、z/OS チェックポイントを使用して再始動することはできません。z/OS ファイルにアクセスする場合は、ユーザーは自分でチェックポイント設定および再始動の方法を作成しなければなりません。

コミット・ポイントでの処置に加えて、チェックポイント呼び出しを出すと IMS は以下のことを行います。

- ユーザーのプログラムがデータベースに加えた変更を永続的にすることができることを、Db2 for z/OS に通知します。 Db2 for z/OS は Db2 for z/OS データに加えられた変更を永続的なものにし、 IMS は IMS データに加えられた変更を永続的なものにします。
- PSB が DB PCB を含む場合にだけ、チェックポイント呼び出しで与えられたチェックポイント識別を含むログ・レコードを、システム・ログに書き込みます。IMS のファイル選択および印刷フォーマット設定プログラム (DFSERA10) を使用すると、チェックポイント・ログ・レコードを印刷できます。このユーティリティーを使用すると、タイプ、含まれるデータ、またはデータ・セット内の順次位置に基づいて、ログ・レコードを選択および印刷することができます。 チェックポイント・レコードは、X'18' のログ・レコードです。

  関連資料: DFSERA10 プログラムについて詳しくは、「*IMS V15* システム・ユーティリティー」を参照してください。
- チェックポイント呼び出しで与えられたチェックポイント識別を含むメッセージを、システム・コンソールのオペレーターや IMS マスター端末オペレーターに送信します。
- プログラムが入力メッセージを処理する場合は、プログラム入出力領域に次の入力メッセージを返します。 MPP およびトランザクション指向 BMP では、チェックポイント呼び出しは、新しいメッセージの呼び出しと似た働きをします。

制約事項**:** 結果が予測できないので、IMS のチェックポイントをとるために、DD ステートメントに CHKPT=EOV を指定しないでください。

関連概念:

56 ページの『プログラム・リカバリーの計画: チェックポイントおよび再始動』

チェックポイント呼び出しの使用が重要な場合

チェックポイント呼び出しを出すことは、組み込みのコミット・ポイントがないプログラムの場合に最も重要です。

プログラムがチェックポイントを出すべきかどうか、また、出すべきであればどのくらいの頻度で出すかは、ユーザーのプログラムによって決定されます。一般的に、以下のプログラムはチェックポイント呼び出しを出す必要があります。

- 複数モード・プログラム
- バッチ指向 BMP (BMP は `SYNC` 呼び出しまたは `CHKP` 呼び出しのどちらを出すこともできます。)
- 大部分のバッチ・プログラム
- データ共用環境で稼働するプログラム
- JMP アプリケーション

以下のプログラムでは、チェックポイント呼び出しを出す必要はありません。

- 単一モード BMP または MPP プログラム
- データベース・ロード・プログラム
- PROCOPT=GO オプションで定義された (PSBGEN の間に) 読み取り専用モードでデータベースにアクセスするプログラムで、最初から再始動できる短さのもの
- データベースを排他使用するプログラム

MPP およびトランザクション指向 BMP のチェックポイント

プログラムのモードのタイプは、IMS システム生成中に、TRANSACT マクロの MODE キーワードで指定します。 モードには単一モードと複数モードがあります。

- 単一モード・プログラムの場合

  単一モード・プログラム (IMS システム定義の間に TRANSACT マクロに MODE=SNGL が指定されている) の場合、メッセージ・キューに対する GET UNIQUE によって、暗黙コミットが行われ、処理されます。
- 複数モード・プログラムの場合

  複数モード BMP および MPP では、コミット・ポイントは、プログラムによって出されるチェックポイント呼び出しからのものと、プログラムの正常終了からのものとがあるだけです。チェックポイント呼び出しを出していない場合にプログラムが異常終了すると、IMS はプログラムが行ったデータベース更新をバックアウトし、プログラムの開始以後に作成されたメッセージを取り消します。プログラムが、チェックポイント呼び出しを出していれば、 IMS は、最後のチェックポイント以降にプログラムが行った変更をバックアウトし、プログラムが作成した出力メッセージを取り消します。

複数モード・プログラムでチェックポイント呼び出しを出す場合には、次のことを考慮してください。

- 1 つの処理単位をバックアウトし、リカバリーするのにどのくらい時間がかかるか。プログラムでは、バックアウトおよびリカバリーが容易にできる頻度でチェックポイントを設定する必要があります。
- 出力メッセージをどのようにグループ分けしたいか。 チェックポイント呼び出しは、複数モード・プログラムの出力メッセージのグループ分けのしかたを確立します。プログラムでは、出力メッセージを作成しすぎるのを防ぐように、チェックポイント呼び出しは頻繁に設定する必要があります。

データベース編成に応じて、チェックポイント呼び出しを出すと、データベース内でのユーザーの位置をリセットすることができます。

関連資料 : チェックポイントが出された際にユーザーの位置を失った場合の詳細については、「*IMS V15* データベース管理」を参照してください。

バッチ指向 BMP のチェックポイント

バッチ指向 BMP でチェックポイント呼び出しを出すことは、いくつかの理由により重要です。

- データベースへの変更をコミットしたり、プログラムが再始動できる地点を確立することに加えて、チェックポイント呼び出しによって、IMS がプログラムに対してロックしていたリソースを開放することができます。
- DEDB または MSDB を使用しているバッチ指向 BMP は、アプリケーション・プログラムが終了する前に、SYNC 呼び出しまたは CHKP 呼び出しを出さないと、異常終了コード U1008 で終了することがあります。
- バッチ指向 BMP が十分な頻度でチェックポイント呼び出しを出さない場合、バッチ指向 BMP が異常終了するか、他のアプリケーション・プログラムが、以下の理由のため IMS によって異常終了することがあります。
  - BMP が、チェックポイント呼び出しとチェックポイント呼び出しの間に多量のデータベース・レコードを更新すると、その更新の間はデータベースの大きな部分が使用できなくなるため、これらのセグメントを必要としている他のプログラムを長時間待たせることになります。

    例外: GO 処理オプションを使用した BMP、または排他使用の BMP の場合は、IMS はプログラムのセグメントをロックしません。 チェックポイント呼び出しを出すと、BMP によってロックされたセグメントが解放され、これらのセグメントが他のプログラムにより使用可能になります。
  - プログラムが読み取って更新したセグメントに関するロック情報を保持するのに必要なスペースは、IMS システムに対して定義されていたスペースより大きくなります。 BMP が多くのセグメントをロックしすぎた場合、ロックされたセグメントに必要なストレージの量が、使用可能なストレージの量を超えてしまう場合があります。その場合、IMS はプログラムを異常終了します。プログラム設計者は、プログラムを再び実行する前に、プログラムのチェックポイントの頻度を増やさなければなりません。 使用可能なストレージは、IMS システム定義の際に指定します。

関連資料 : ストレージの指定の詳細については、「*IMS V15* システム定義」を参照してください。

PSBGEN ステートメントの LOCKMAX=*n* パラメーターを使用して、 BMP が行うロックの回数を制限することができます。例えば、LOCKMAX=5 を指定すると、どのような場合でもアプリケーションが 5000 回を超えるロックを行うことは許可されません。*n* の値は、0 から 255 にしなければなりません。最大ロック数が指定されていない場合には、0 がデフォルトになります。 BMP が指定された数より多くのロックを行おうとすると、IMS は異常終了コード U3301 を使用してアプリケーションを終了します。

関連資料: この異常終了について詳しくは、「*IMS V15* メッセージおよびコード 第 *3* 巻: *IMS* 異常終了コード」を参照してください。

バッチ・プログラムのチェックポイント

データベースを更新するバッチ・プログラムでは、チェックポイント呼び出しを出す必要があります。バッチ・プログラム内で、どのくらいの頻度でチェックポイントを取るかを決定する際は、障害が起きた場合に、バックアウトおよび再処理にかかる時間を主に考慮してください。一般的に、10 分または 15 分に一度チェックポイント呼び出しを行うことをお勧めします。

バッチ・プログラム全体をバックアウトする必要がある場合には、プログラムは一番初めの段階で、チェックポイント呼び出しを出さなければなりません。 IMS は、指定されたチェックポイントまでプログラムをバックアウトします。チェックポイントが指定されていない場合は、最後のチェックポイントまでプログラムをバックアウトします。プログラムの開始後の最初のチェックポイントの前にデータベースが更新された場合は、IMS はこれらのデータベース更新をバックアウトできません。

バッチ・プログラムがチェックポイント呼び出しを出すには、PSB 内に互換性オプション (CMPAT=YES) が指定されていなければなりません。この互換性オプションによってプログラムに I/O PCB が生成され、チェックポイント呼び出しの際に、IMS が I/O PCB としてそれを使用します。

それ以外にも、バッチ・プログラムにチェックポイント呼び出しを出すことが重要な理由は、バッチ・プログラムが現在 IMS バッチ領域で稼働していても、後にオンライン・データベースにアクセスすることが必要になる可能性があるためです。これを行うためには、バッチ・プログラムを BMP に変換する必要があります。BMP でチェックポイント呼び出しを出すことは、リカバリー以外の理由、例えばデータベース・リソースを他のプログラムに解放するためにも重要です。このため、作成するすべてのバッチ・プログラムに最初からチェックポイントを組み込んでおく必要があります。 その時点ではチェックポイントのサポートが必要ないかもしれませんが、後でチェックポイント呼び出しを組み込むよりも、あらかじめ組み込んでおいたほうが簡単です。

他のプログラムのためにデータベース・リソースを解放する場合、データ共用環境で稼働するバッチ・プログラムは、この環境で稼働していないバッチ・プログラムより、より頻繁にチェックポイント呼び出しを出す必要があります。

関連概念:

43 ページの『DB バッチ処理』
48 ページの『オンラインでのバッチ処理: バッチ指向 BMP』
50 ページの『バッチ・メッセージ処理: トランザクション指向 BMP』
56 ページの『プログラム・リカバリーの計画: チェックポイントおよび再始動』

チェックポイントの頻度の指定

プログラム設計者は、プログラム内のチェックポイントの頻度を指定する際に、頻度の調整が必要な場合に容易に変更できるように指定しておく必要があります。

このことは、以下のことによって行うことができます。

- プログラム内でカウンターを使用して、経過時間を記録し、特定の時間間隔でチェックポイント呼び出しを出す。
- カウンターを使用して、プログラムがアクセスしたルート・セグメントの数を記録し、決められたルート・セグメントへのアクセス数の後でチェックポイント呼び出しを出す。
- カウンターを使用して、プログラムが実行した更新の数を記録し、決められた更新数の後で、チェックポイント呼び出しを出す。

関連概念:

56 ページの『プログラム・リカバリーの計画: チェックポイントおよび再始動』

データの可用性に関する考慮事項

ここでは、全機能データベースでデータが使用不可になる状況と、そのような状況においてプログラムによるデータ管理を可能にするプログラム呼び出しについて説明します。

使用不能データの処理

以下の条件下では、データベースは読み取りおよび更新のどちらも実行できません。

- データベースに対する /LOCK コマンドが出されている。
- データベースに対する /STOP コマンドが出されている。
- /DBRECOVERY コマンドが出されている。
- データベースに対する許可が失敗した。

データベースが読み取りはできるが更新ができなくなる条件は、以下のとおりです。

- /DBDUMP コマンドが出されている。
- データベース ACCESS 値が RD（読み取り）になっている。

データベースが完全に使用不能な状況だけでなく、データの一部が使用不能な状況においても、プログラムによるアクセスは禁止されます。 そのような例としては、データ共用にかかわる障害があります。その場合、データを共用する IMS システムに障害が起こったときにその IMS システムがどのロックを保持していたのかを、アクティブ IMS システムが認識しています。アクティブ IMS システムは、データベースの使用を続けますが、障害が起きた IMS システムが障害時にロックし

たデータへのアクセスは拒否する必要があります。この状況は、全機能データベースおよび DEDB データベースの両方で起こります。

以下の 2 つの状況において、プログラムが使用不能なデータに出会うことがあります。

- プログラムが呼び出しを行って、プログラムがスケジュールされた時点に使用不能だったデータベースへのアクセスを要求した場合
- プログラムがスケジュールされたときにはデータベースが使用可能であったが、データの一部が使用不能であり、現行の呼び出しが、その使用不能データへのアクセスを試みた場合

データが使用不能になった状況には関係なく、プログラムが使用不能データを処理するには、以下の 2 つの方法が可能です。プログラムは、データが使用不能であることに対し、センシティブと見なす場合とセンシティブと見なさない場合があります。

- プログラムがセンシティブと見なさない場合、プログラムが使用不能なデータへアクセスしようとすると、IMS は適切な処置を取ります。
- プログラムがセンシティブと見なす場合、IMS は、プログラムに対して、アクセスしようとしているデータは使用不能であることを通知します。

プログラムが、データが使用不能であることをセンシティブと見なさない場合に、使用不能データへのアクセスを試みると、 IMS はプログラムを中断し (3303 疑似異常終了)、プログラムが行った更新をすべてバックアウトします。 プログラムが処理していた入力メッセージは延期され、プログラムは、データが使用可能になったときに入力メッセージを処理するようにスケジュールされます。しかし、動的割り振りが失敗したためにデータベースが使用不能になった場合は、呼び出しの結果、AI (オープン不可) 状況コードが返されます。

データが使用不能であることをプログラムがセンシティブと見なす場合に、使用不能データへのアクセスを試みると、IMS は状況コードを返して、その呼び出しを処理できないことを通知します。プログラムは、その後、適当な処置を取ります。プログラムが、使用不能データをセンシティブと見なさない場合、プログラムには、IMS が取るのと同じ処置を開始する機能が存在します。

バッチ・プログラムがアクセスできるデータが使用不能な場合には、 IMS はそのバッチ・プログラムをスケジュールしません。バッチ・プログラムが、ブロック・レベルのデータ共用を使用している場合に、共用システムに障害が起こり、その障害システムによって更新されたが、コミットされていないデータをバッチ・システムがアクセスしようとすると、使用不能データに出会うことがあります。

以下の条件のみでは、バッチ・プログラムは初期設定時に障害を起こしません。

- PCB が HALDB を参照する。
- DBRC の使用が抑制される。

ただし、DBRC を使用しない場合、HALDB に PCB を使用するデータベース呼び出しは許可されません。このプログラムが使用不能データをセンシティブと見なす場合は、このような呼び出しにより状況コード BA が出されます。そうでない場合は、このような呼び出しによりメッセージ DFS3303I とその後に ABENDU3303 が出されます。

使用不能データベースのスケジューリングおよびアクセス

INIT、INQY、SETS、SETU 、および ROLS 呼び出しを使用することにより、プログラムは、使用不能なデータベースでスケジュールされたデータ環境を管理することができます。

INIT 呼び出しは、プログラムが使用不能データをセンシティブと見なすことを、IMS に対して通知し、プログラムが使用不能データへのアクセスを試みたときに出された状況コードを受け入れることができます。 INIT 呼び出しは、各 PCB についてデータ可用性を判別するために使用することもできます。

INQY 呼び出しは、バッチおよびオンラインのいずれの IMS 環境でも作動可能です。 IMS アプリケーション・プログラムは INQY 呼び出しを使用して、出力宛先、セッション状況、現行の実行環境、データベースの可用性、および PCBNAME に基づく PCB アドレスに関する情報を要求することができます。 INQY 呼び出しは、AIB インターフェース (PCB アドレスではなく AIB を使用した AIBTDLI または CEETDLI) を通じてのみサポートされます。

SETS、SETU、および ROLS 呼び出しを使用すると、アプリケーションで複数の位置を定義して、全機能データベース (HSAM を除く) の状態、およびメッセージのアクティビティーを保持することができます。アプリケーションは、後でこれらの位置に戻ることができます。 DL/I 呼び出しを開始して機能を実行する前に SETS 呼び出しまたは SETU 呼び出しを出すと、データ使用不能のために機能を完了できない場合に、プログラムは ROLS 呼び出しを出すことができます。

ROLS 呼び出しを使用すると、プログラムは、IMS 全機能データベース・アクティビティーを、 SETS 呼び出しまたは SETU 呼び出しが出される前の状態にロールバックすることができます。 PSB 内に MSDB または DEDB が存在する場合、SETS および ROLS (トークン使用) 呼び出しは無効になります。 PSB 内に DEDB、MSDB、または GSAM PCB が存在する場合、 SETS 呼び出しではなく、SETU 呼び出しを使用してください。

また、ROLS 呼び出しは、最後のコミット・ポイント以降のすべての更新アクティビティー (データベースおよびメッセージ) を取り消す場合や、現行の入力メッセージを後で処理するために延期キューに入れる場合にも使用できます。この処置は、トークンまたは入出力域を使用せずに ROLS 呼び出しを出すと開始することができます。

制約事項: Db2 for z/OS では、ROLS (トークン使用) および SETS は使用できません。

関連情報:

3303 (メッセージおよびコード)

IMS プログラムでの STAE または ESTAE、および SPIE の使用

制御領域、従属 (MPP、IFP、BMP) 領域、およびバッチ領域では、 IMS は STAE または ESTAE ルーチンを使用します。制御領域では、STAE または ESTAE ルーチンによって、データベース・ロギングおよび各種のリソース終結機能が確実に完了します。

従属領域では、 STAE または ESTAE ルーチンを使用して、アプリケーション・プログラムまたは従属領域の異常終了を制御領域に通知します。制御領域に従属領域終了が通知されない場合、リソースは適切に解放されず、正常なチェックポイント・シャットダウンが妨げられる可能性があります。

バッチ領域では、STAE または ESTAE ルーチンによって、データベース・ロギングおよび各種のリソース終結機能が確実に完了します。 バッチ領域にアプリケーション・プログラムの終了が通知されない場合、リソースは適切に解放されない可能性があります。

STAE または ESTAE 機能には、以下の 2 つの重要な特性があります。

- IMS は、データベース保全性およびリソースの制御を完全に STAE または ESTAE 機能に頼っています。
- STAE または ESTAE 機能はアプリケーション・プログラムでも使用可能です。

この 2 つの要因があるため、プログラムと STAE または ESTAE 機能との間の関係を明確に理解しておかなければなりません。

通常、アプリケーション・プログラムでは STAE または ESTAE 機能を使用しないでください。ただし、STAE または ESTAE 機能の使用が必要であると思われる場合には、次の基本規則に従ってください。

- 環境が STAE または ESTAE 処理をサポートしている場合、アプリケーション・プログラムの STAE または ESTAE ルーチンは、常に IMS STAE または ESTAE ルーチンよりも優先的に制御権を持っています。したがって、アプリケーション・プログラム内で次の手順を守って、IMS STAE または ESTAE 出口ルーチンが制御を受け取るようにしてください。
  - STAE または ESTAE ルーチンは一度だけ、必ず最初の DL/I 呼び出しの前に確立してください。
  - STAE または ESTAE 機能を使用する場合は、アプリケーション・プログラムは IMS 異常終了コードを変更してはなりません。
  - STAE または ESTAE ルーチンから出るときには、RETRY オプションを使用しないでください。代わりに、STAE または ESTAE 処理の終了時に CONTINUE-WITH-TERMINATION 標識を返してください。 アプリケーション・プログラムが RETRY オプションを指定している場合は、 IMS STAE または ESTAE 出口ルーチンは終結処置を行う制御権を持たないことに注意してください。したがって、システムおよびデータベースの保全性に危険が伴います。
- アプリケーション・プログラムの STAE または ESTAE 出口ルーチンは、DL/I 呼び出し (DB または TM) を出してはなりません。これは、異常終了がもともとアプリケーションと IMS の間の問題によって発生した可能性があるためです。アプリケーションと IMS の間の問題は、結果として、データベース保全性を損失する恐れがある STAE または ESTAE への再帰入力が起こるか、またはチェックポイントを取る際に問題が発生することがあります。また、この結果として停止状態になったり、終了時に ABENDU0069 が出されることがあります。

関連概念:

188 ページの『IMS プログラムの異常終了時の処置』

IMS データベースの動的割り振り

バッチ・ジョブまたはオンライン・ジョブの JCL の代わりに、ライブラリー内の IMS データベースについての JCL 情報を指定するには、動的割り振り機能を使用してください。

動的割り振りを使用している場合、動的割り振りに定義されたデータベース・データ・セットについては JCL DD ステートメントを組み込まないでください。 DBA または同等の技能のある専門家に相談して、動的割り振りに定義されているデータベースを判別してください。

関連資料: 動的割り振りの定義に関する追加情報については、「*IMS V15* システム定義」の DFSMDA マクロの項を参照してください。

第 4 章 CICS アプリケーションの処理要件の分析

IMS は、CICS 環境で実行するアプリケーション・プログラムをサポートします。

CICS アプリケーション要件の定義

アプリケーション設計の段階として、エンド・ユーザーが実行したい業務処理またはタスクを、どのようにして複数のプログラムに分割すると最も効率的に必要とされる処理を実行できるのかを決定する必要があります。

処理要件の分析における考慮事項は、以下のとおりです。

- タスクをいつ実行するのか
  - タスクを (例えば端末の要求次第で実行するなど) 予測できないようなスケジュールをするのか、あるいは (例えば週ごとに実行するなど) 定期的にスケジュールするのか。
- そのタスクを実行するプログラムがどのように稼働するのか
  - タスクをオンライン実行するか (応答時間が非常に重要)、あるいはバッチ・ジョブとして実行依頼して実行するか (応答時間が遅くても可)。
- 処理コンポーネントの一貫性
  - その処理には、複数のプログラム論理タイプが含まれるのか。 例えば、タスクの大部分がリトリーブであり、1 つまたは 2 つの更新だけが含まれるとします。 その場合は、更新部分を分離して独立したプログラムにすることを考慮すべきです。
  - その処理には、複数の大きなデータ・グループが含まれるか。その場合は、プログラムがアクセスするデータごとにプログラムを分離すると、より効率的です。
- データまたは処理に関する特別要件

  セキュリティー
  : プログラムへのアクセスを制限するかどうか。

  リカバリー
  : プログラム処理において、リカバリーに関する特別な考慮事項があるかどうか。

  保全性
  : 他の部門が同一のデータを使用するかどうか。

これらの質問に対して答えることは、処理に必要なアプリケーション・プログラムの数、および最も効率的に処理を実行できるプログラムのタイプを決定することに役立ちます。プログラムの数がいくつであれば最も効率的に必要な処理を行うことができるかについての規則はありませんが、以下にいくつかの提案を示します。

- それぞれのプログラミング・タスクについて、含まれるデータおよび処理を調べてください。 1 つのタスクで、異なるタイプの処理が必要とされ、時間的制限が異なる（例えば、毎週と毎月）場合、そのタスクは、複数のプログラムで実行したほうが効率的です。

- プログラムを定義する際は、プログラムをできるだけ単純にしたほうが、保守およびリカバリーが容易になります。 プログラムは単純である (および実行することが少ない) ほど保守がしやすく、プログラムまたはシステムの障害の後の再始動も容易です。データ可用性についても同じことがいえます。データがアクセスされる回数が少ないほど、また、データへのアクセスを制限するほど、データは利用しやすくなります。

  同様に、アプリケーションに必要なデータが物理的に 1 つの場所にあると、1 つのプログラムで通常よりも多くの処理を行うことができ、より効率的になる場合があります。 これらは、それぞれのアプリケーションの処理およびデータに基づいた考慮事項です。
- ユーザーの各作業を文書化しておくと、設計プロセスで役立つばかりでなく、将来他の要員がそのアプリケーションに関連して作業する場合にも役立ちます。この領域における標準を知っている必要があります。通常保持される情報は、いつタスクが実行されるのか、機能の説明、保守、セキュリティー、およびリカバリーの要件です。

  例えば、前述の現行名簿の処理の場合、以下のフォームに示す情報を記録することになります。プログラムが実行される頻度は、クラスの数 (20) によって決定されます。 1 週間につきこの回数だけ、Ed センターが現行名簿を印刷します。

例: 現行名簿タスクの説明

```
                    USER TASK DESCRIPTION

NAME: Current Roster
ENVIRONMENT: Batch                    FREQUENCY: 20 per week

INVOKING EVENT OR DOCUMENT: Time period (one week)

REQUIRED RESPONSE TIME: 24 hours

FUNCTION DESCRIPTION: Print weekly, a current student roster, in student
 number sequence for each class offered at the Education Center.

MAINTENANCE: Included in Education DB maintenance.

SECURITY: None.

RECOVERY: After a failure, the ability to start printing a particular
class roster starting from a particular sequential student number.
```

CICS アプリケーション・プログラムを使用したデータベースへのアクセス

プログラムを設計する際には、そのプログラムがアクセスするデータのタイプを考慮してください。 データのタイプは操作環境によって決まります。

次の表は、IMS データベース、Db2 for z/OS データベース、および z/OS ファイルのデータで、CICS オンライン・プログラムと IMS バッチ・プログラムが使用可能なものを示しています。

表 *22. CICS* プログラムがアクセスできるデータ

プログラムのタイプ	**IMS** データベース	**Db2 for z/OS** データベース	**z/OS** ファイル
CICS オンライン	可[1]	可[2]	可[3]

表 22. *CICS* プログラムがアクセスできるデータ (続き)

プログラムのタイプ	**IMS** データベース	**Db2 for z/OS** データベース	**z/OS** ファイル
DB バッチ	あり	可[3]	あり

注:

1. 汎用順次アクセス方式 (GSAM) データベースは除きます。 GSAM を使用すると、バッチ・プログラムで、単純データベースとして順次 z/OS データ・セットにアクセスすることが可能になります。
2. IMS はこの呼び出し処理には使用されません。
3. CICS ファイル制御または一時データ・サービスを通じてアクセスします。

また、ユーザーのプログラムがアクセスする必要のあるデータベースのタイプについても考慮してください。次の表に示すように、作成可能なプログラムのタイプ、およびアクセス可能なデータベースのタイプは、操作環境に応じて異なります。

表 23. *CICS* 環境におけるプログラムおよびデータベースのオプション

環境[1]	書き込み可能なプログラムのタイプ	アクセス可能なデータベースのタイプ
DB バッチ	DB バッチ	Db2 for z/OS[2] DL/I 全機能 GSAM z/OS ファイル
DBCTL	BMP	Db2 for z/OS DEDB 全機能 GSAM z/OS ファイル
	CICS オンライン	Db2 for z/OS[2] DEDB 全機能 z/OS ファイル (CICS ファイル制御または一時データ・サービスを通じてのアクセス)

注:

1. CICS 環境または CICS リモート DL/I 環境も存在し、機能シップとしても参照されます。この環境では、CICS システムは DL/I 呼び出しを出すアプリケーションをサポートしますが、CICS システムは要求そのもののサービスは行いません。この CICS 環境は、DL/I 呼び出しを、DBCTL を使用している別の CICS システムに「機能シップ」します。リモート DL/I について詳しくは、「*CICS Transaction Server for z/OS IMS Database Control Guide*」を参照してください。
2. IMS はこの呼び出し処理には使用されません。

アクセス可能なデータベースのタイプは、以下のとおりです。

- 全機能データベース

全機能データベースは階層データベースであり、データ言語 I (DL/I) を使用してアクセスできます。 DL/I 呼び出しを行うと、アプリケーション・プログラムで全機能データベースに対して、セグメントのリトリーブ、置換、削除、および追加を行うことができます。 CICS オンライン・プログラムおよび BMP プログラムは、同時に同じデータベースをアクセスする場合があります (IMS データ共用を行っている場合)。 IMS バッチ・プログラムはデータベースに対して排他的アクセスを持っていなければなりません (IMS データ共用を行っていない場合)。

全機能データベースには、すべてのタイプ (バッチ、BMP、およびオンライン) のプログラムがアクセスできます。

- 高速機能 **DEDB**

  高速処理データベース (DEDB) は、多量の明細データを効率的にアクセスするための階層データベースです。DBCTL 環境では、CICS オンライン・プログラムおよび BMP プログラムが、 DEDB にアクセスすることができます。

- **Db2 for z/OS** データベース

  Db2 for z/OS データベースはリレーショナル・データベースです。リレーショナル・データベースは、アプリケーション・プログラムおよびユーザーに対しては、表の形式で表され、構造化照会言語 (SQL) と呼ばれる関係データ言語を使用して処理されます。 Db2 for z/OS データベースは、IMS バッチ・プログラムおよび BMP プログラムと同様に、 CICS オンライン・トランザクションでも処理することができます。

  関連資料: Db2 for z/OS データベースの処理については、「*DB2 for z/OS* アプリケーション・プログラミングおよび *SQL* 解説書」を参照してください。

- **GSAM** データベース

  汎用順次アクセス方式 (GSAM) というアクセス方式を使用すると、BMP およびバッチ・プログラムで、単純データベースとして「フラットな」順次 z/OS データ・セットにアクセスすることが可能になります。 GSAM データベースには、z/OS または CICS によってアクセスすることができます。

- **z/OS** ファイル

  CICS オンライン・プログラムおよび IMS バッチ・プログラムは、 z/OS ファイルに対し、入力、処理、出力でのアクセスが可能です。バッチ・プログラムは、z/OS ファイルに直接アクセスすることができますが、オンライン・プログラムでは、CICS ファイル制御または一時データ・サービスを通じてファイルにアクセスする必要があります。

関連概念:

72 ページの『CICS プログラムのデータ共用の使用』

IMS データベースにアクセスする CICS プログラムの作成

使用可能なプログラムのタイプは、稼働環境が DBCTL 環境であるかどうかによって異なります。異なる複数の環境で使用する場合は、ユーザーが作成するプログラムのタイプは、アプリケーションに必要な処理によって異なります。それぞれのタイプのプログラムは、異なるアプリケーション要件に対応します。

関連概念:

693 ページの『第 38 章 Java 開発用の IMS ソリューションの概要』

CICS オンライン・プログラムの作成

以下の情報は、オンライン・プログラムがアプリケーションに適しているかどうかを判断するために使用します。

CICS オンライン・プログラムがアクセスできるデータ

CICS オンライン・プログラムは DBCTL 環境で稼働し、IMS 全機能データベース、高速機能 DEDB、Db2 for z/OS データベース、および z/OS ファイルにアクセスすることができます。

IMS データベースにアクセスするオンライン・プログラムは、他の CICS プログラムと同様に実行されます。

CICS オンライン・プログラムの使用

オンライン・プログラムは、CICS の制御下で稼働し、他のオンライン・プログラムと同時にリソースにアクセスします。 オンライン・プログラムが満たすことができるアプリケーション要件は、以下のとおりです。

- データベース内の情報は、多くのユーザーが使用できなければならない。
- プログラムは、端末および他のプログラムと通信する必要がある。
- プログラムは、リモート端末のユーザーにも使用可能でなければならない。
- 応答時間は重要である。

オンライン・プログラムの構造、および状況情報の受信のしかたは、そのプログラムが呼び出しレベルのプログラムかコマンド・レベルのプログラムかによって異なります。 しかし、コマンド・レベルのプログラム、呼び出しレベルのオンライン・プログラムは、ともに以下のことを行います。

- PSB をスケジュールします (CICS オンライン・プログラムの場合)。 PSB は、バッチ・プログラムまたは BMP プログラムの場合は自動的にスケジュールされます。
- コマンドまたは呼び出しを出してデータベースにアクセスします。オンライン・プログラムでは、1 つの作業論理単位 (LUW) でコマンドおよび呼び出しを混合することはできません。
- オプションで、CICS オンライン・プログラムの PSB を終了します。
- 処理を終えると、`EXEC CICS RETURN` ステートメントを出します。このステートメントによって、リンク・プログラムに制御が戻されます。 最もレベルが高いプログラムが RETURN ステートメントを出すと、CICS に制御が戻り、 CICS は、PSB が終了していなければそれを終了します。

オンライン・アプリケーション・プログラムは、複数のタスクで同時に使用する場合があるため、準再入可能でなければなりません。

DBCTL 環境のオンライン・プログラムは、多数の IMS システム・サービス要求を出すことができます。

DL/I データベース要求またはシステム・サービス要求は、IMS によってプログラムに渡されるプログラム連絡ブロック (PCB) のリストの中の 1 つの PCB に基づいて作成されなければなりません。 システム・サービス要求を作成するのに使用しなければならない PCB を、I/O PCB といいます。 I/O PCB が存在する場合、それは PCB リスト内の最初の PCB です。

DBCTL 環境内のオンライン・プログラムの場合、I/O PCB はオプションです。 I/O PCB を使用する場合、アプリケーション・プログラムは PSB をスケジュールするときにプログラム内でこれを指示しなければなりません。

プログラムを実行する前に、プログラムが使用するプログラム仕様ブロック (PSB) およびデータベース記述 (DBD) を、IMS ACBGEN ユーティリティーを使用して、内部制御ブロック・フォーマットに変換する必要があります。 PSB では、アプリケーション・プログラムの特性を指定します。 DBD では、IMS データベースの物理特性および論理特性を指定します。

関連資料 **:** ACBGEN および PSBGEN の実行に関する詳細については、「*IMS V15* システム・ユーティリティー」を参照してください。

オンライン・プログラムは他のオンライン・プログラムとデータベースを共用しているため、ユーザーのオンライン・システムのパフォーマンスに影響を与えます。

関連概念:

75 ページの『CICS システムのパフォーマンスの最大化』

702 ページの『IMS Universal ドライバーを使用した分散およびローカル・コネクティビティー』

CICS プログラムのデータ共用の使用

データ共用を使用する場合は、ユーザーのプログラムは IMS データ共用に加わることができます。データ共用を使用すると、CICS オンライン・プログラムと BMP プログラムは、同時に同じ DL/I データベースをアクセスすることができます。

データ共用環境において、バッチ・プログラムは、他のバッチ・プログラム、 CICS オンライン・プログラム、および IMS オンライン・プログラムによって使用されているデータベースにアクセスすることができます。データ共用を使用すると、直接データを共用することができ、プログラムの要求はミラー・トランザクションを通じて行う必要はありません。

関連資料**:** IMS システムによるデータベースの共用について詳しくは、「*IMS V15* システム管理」を参照してください。

関連概念:

68 ページの『CICS アプリケーション・プログラムを使用したデータベースへのアクセス』

PSB のスケジュールおよび終了 (CICS オンライン・プログラムのみ)

ユーザーのオンライン・プログラムが DL/I 呼び出しを出す前に、 PCB 呼び出しまたは SCHD コマンドを出して、特定の PSB を使用する意図を IMS に通知しなければなりません。ユーザーのプログラムがどの PSB を使用するのかを指示するだけではなく、 PCB 呼び出しは PSB 内での PCB のアドレスを獲得することもできます。 PSB が必要でなくなれば、TERM 要求を使用して終了することができます。

CICS オンライン・プログラムでは、 PCB 呼び出しまたは SCHD コマンド (コマンド・レベル・プログラムの場合) を使用して、プログラムに PSB を獲得します。トランザクションの終了時に、CICS はプログラムが使用している PSB を解放するので、ユーザーのプログラムが明示的に PSB を終了する必要はありません。 ユーザーは、以下の場合のみ終了要求を使用してください。

- 別の PSB を使用したい場合
- データベース更新をすべてコミットし、更新をバックアウトするための作業論理単位を確立したい場合
- 他の CICS タスクで使用できるように、IMS リソースを解放したい場合

終了要求によって CICS 同期点が発生し、CICS 同期点によって PSB が終了します。 CICS リカバリーの概念に関する詳細については、適当な CICS 資料を参照してください。

その他の理由で終了要求を使用してはならない理由は以下のとおりです。

- 終了要求を出すと、必ず CICS 同期点が出されます。 この同期点は、すべてのリカバリー可能リソース、およびこのタスクのためにエンキューされた IMS データベース・リソースを解放します。

  終了要求が出された後もプログラムが他の CICS リソースの更新を続け、異常終了した場合、終了要求が出された後で更新されたリソースだけがバックアウトされます。プログラムが行った IMS 変更はバックアウトされません。
- IMS ロック管理によって、デッドロックが検出されます。このデッドロックは、2 つのトランザクションが、他のトランザクションによって保留されているセグメントを待っている場合に起こります。

  デッドロックが検出されると、一方のトランザクションは異常終了します。 データベース変更は、最後の TERM 要求の地点までバックアウトされます。 デッドロックよりも前に TERM 要求または CICS 同期点が出された場合、CICS はトランザクションを再始動しません。

  関連資料: トランザクションの再始動の考慮事項に関する詳細な説明については、「*CICS Transaction Server for z/OS Recovery and Restart Guide*」を参照してください。
- 終了要求を出すと、追加ロギングが起こります。
- 終了要求の後で端末出力要求が出され、トランザクションがその地点で失敗した場合、端末オペレーターはメッセージを受信しません。

端末オペレーターはすべてトランザクションが失敗したと見なし、再入力を行う可能性があります。そうすると、終了要求が出される以前に行われた更新を繰り返すことになります。終了要求より以前に行われた更新はバックアウトされません。

他のプログラムとのリンクおよび制御の受け渡し (CICS オンライン・プログラムのみ)

CICS を使用すると、プログラムを他のプログラムにリンクする際に、リンク元のプログラムで獲得した機能へのアクセスを失うことがありません。

以下に例を示します。

- PSB をスケジュールしてから、 LINK コマンドを使用して他のプログラムにリンクしたとします。 そのプログラムから戻る際も、PSB はスケジュールされたままです。
- 同様に、XCTL コマンドを使用して他のプログラムに制御を渡したとすると、そのプログラムが EXEC CICS RETURN ステートメントを出すまで、PSB はスケジューリングされたままです。 しかし、XCTL を使用して他のプログラムに制御を渡す場合、制御を渡す側のプログラムの作業用ストレージは失われます。その作業用ストレージを、リンク先のプログラムでも使用できるように保存したい場合は、COMMAREA でその情報を渡してください。

推奨事項: 作業を単純化するため、別のプログラムとリンクさせる代わりに、1 つのプログラム・モジュールですべての DL/I 要求を発行できます。そうすると、プログラミングを単純にすることができ、保守が容易になります。

PSB を終了したり、同期点を出すことは、リンク元のプログラムに影響を与えます。例えば、リンク先のプログラムで出された終了要求または同期点によって、リンク元のプログラム内でエンキューされた CICS リソースが解放されます。

CICS 分散トランザクションが IMS にアクセスする方法

CICS によって、単一の論理作業単位を別個の CICS トランザクションに分割し、グローバルに同期点を調整することができます。このような CICS トランザクションが DBCTL にアクセスした場合、ロッキングおよびバッファー管理に関する問題が発生する可能性があります。

IMS に対しては、このトランザクションは異なる DBCTL スレッド上の分離した作業単位であり、ロックおよびバッファーを共用しません。例えば、グローバル・トランザクションが稼働し、データベース・ロックを取得し、コミット・ポイントに達した場合、CICS が同期点処理を行うのは、CICS リカバリー単位 (UOR) 内の他のトランザクションがコミット可能になった後です。同じ CICS UOR 内の 2 番目のトランザクションが、最初のトランザクションが保持するものと同じロックを要求した場合、その 2 番目のトランザクションはロック待機状態のままになります。最初のトランザクションは、2 番目のトランザクションもコミット・ポイントに達してから同期点の完了およびロックの解放を行います。ただしこの状態は、2 番目のトランザクションがロック待機状態にあるため、起こりえません。このタイプの衝突が、IMS にアクセスする CICS 分散トランザクションに発生しないことを確認してください。

CICS システムのパフォーマンスの最大化

他のプログラムとデータを共用するプログラム (例えば、IMS データ共用に加わるプログラム、BMP プログラムなど) を作成する場合は、そのプログラムがオンライン・システムのパフォーマンスに及ぼす影響に注意してください。

特に、BMP プログラムは CICS オンライン・トランザクションのパフォーマンスに影響を与えることがあります。 これは、BMP プログラムは通常は、 CICS オンライン・トランザクションより多量のデータベース更新を行うため、 BMP プログラムのほうが CICS オンライン・プログラムが必要としているよりより多くのセグメントを保留しがちなためです。共用データベース・プログラムまたは BMP プログラムが保留するセグメントの数を制限し、CICS オンライン・プログラムがそれらのセグメントを待たずに獲得できるようにしてください。

BMP プログラム、および IMS データ共用に加わるバッチ・プログラムによって保留されるセグメントの数を制限する方法として、プログラム内でチェックポイント要求を出してデータベースの変更をコミットし、そのプログラムにより保留されたセグメントを解放する方法があります。チェックポイント要求を出す頻度を決定する際は、以下の手法の 1 つまたは両方を使用できます。

- プログラムを小さな論理作業単位に分割し、それぞれの単位の終わりにチェックポイント呼び出しを出す。
- DL/I 要求が一定の数だけ出されるか、またはトランザクションが一定の数だけ処理されるとチェックポイント呼び出しを出す。

CICS オンライン・プログラムでは、オンライン・システムのパフォーマンスを最大化するために、他のトランザクションが使用できるようにセグメントを解放してください。 (通常、CICS に制御が戻されたときにのみ、データベース変更はコミットされ、セグメントは解放されます。) より早急に、リソースを他のトランザクションで使用できるように解放するために、TERM 要求を出して PSB を終了することができます。 しかし、制御が CICS に戻された時に PSB が終了すると通常、処理のオーバーヘッドはより少なくなります。

関連概念:

71 ページの『CICS オンライン・プログラムの作成』

77 ページの『バッチ・プログラムおよび BMP プログラムでのチェックポイントの設定』

CICS プログラムのプログラミングの保全性およびデータベース・リカバリーの考慮事項

IMS では、CICS オンライン・プログラムのデータ保全性の保護がサポートされています。

IMS が CICS オンライン・プログラムのデータ保全性を保護する方法

IMS は、CICS オンライン・プログラムのデータ保全性を保護できます。

IMS は以下のことを行って、データを共用するプログラムのためにデータベースの保全性を保護します。

- ユーザーのプログラムが、あるデータベース・レコードでの処理を終えて、同じデータベース内の新しいデータベース・レコードに移るまで、他のアプリケーション・プログラムが更新機能を使用して処理中のデータベース・レコード内のセグメントにアクセスするのを防ぎます。
- ユーザーのプログラムが同期点に到達するまで、ユーザーのプログラムが削除、置換、または挿入するセグメントに他のプログラムがアクセスするのを防ぎます。プログラムが同期点に到達すると、プログラムがデータベースに対して行った変更が永続的なものになり、変更データが他のアプリケーション・プログラムで使用可能になります。

  例外: ユーザーのプログラムで、PSBGEN 中に PROCOPT=GO が定義されると、ユーザーのプログラムは、他のプログラムによって更新されたがコミットされていないセグメントにアクセスできます。
- 異常終了したアプリケーション・プログラムによって更新されたデータベースをバックアウトします。

セグメントを更新しなくても、プログラムが同期点に到達するまでは、そのプログラムだけが使用されるようにセグメントを保持することにより、そのプログラムがアクセスするデータを保護することもできます。 (通常、セグメントを更新しない場合は、プログラムが新しいデータベース・レコードに移動する際に、IMS はそのセグメントを解放します。) Q コマンド・コードを使用して、プログラムが排他使用できるようにセグメントを予約することができます。データが他のプログラムで使用できなくなったり、パフォーマンスに影響したりすることがあるので、このオプションは、必要な場合だけに使用するようにしてください。

バッチ・プログラムおよび BMP プログラムがアクセスするデータベースのリカバリー

バッチ・プログラムまたは BMP プログラムがアクセスするデータベースについて、リカバリー計画を立てることができます。

CICS は、CICS オンライン・プログラムがアクセスするデータベースを、他のリカバリー可能な CICS リソースの場合を扱うのと同じ方法でリカバリーします。例えば、IMS トランザクションが異常終了した場合、CICS および IMS は、最後の同期点までデータベース更新をバックアウトします。

バッチ・プログラムまたは BMP プログラムの場合は、次のことを行います。

- プログラム内にチェックポイントを取ってデータベース変更をコミットし、プログラムが再始動できる位置を与えてください。
- コードを作成するか、または要求を出してプログラムを再始動してください。

バッチ・プログラムが行ったデータベース変更をコミットしていない場合、その変更をバックアウトしたい場合もあります。

これらのタスクを実行するには、システム・サービス呼び出しを使用します。システム・サービス呼び出しについては、ユーザーの環境に合ったアプリケーション・プログラミング情報に詳細に記述されています。

バッチ・プログラムでの I/O PCB の要求

プログラムが正常にシステム・サービス要求を出すようにするには、あらかじめ I/O PCB を要求しておかなければなりません。

関連概念:

837 ページの『IMS Java 従属領域リソース・アダプターを使用した JBP アプリケーションの開発』

バッチ・プログラムおよび BMP プログラムでのチェックポイントの設定

バッチ・プログラムおよび BMP プログラムでは、チェックポイントを取ることができます。チェックポイントは、リカバリーに使用したり保全性を確保するために重要です。

以下の 2 つの理由のため、バッチ・プログラムおよび BMP プログラムではチェックポイントを取ることが重要です。

- リカバリー: チェックポイントを取ると、プログラムまたはシステムに障害が起こった場合に、プログラムを再始動できる位置をプログラム内に設けることができます。チェックポイント要求を出した後でプログラムが異常終了した場合、チェックポイントが出された位置まで、データベース変更がバックアウトされます。
- 保全性: また、チェックポイントは、プログラムがデータベースに行った変更をコミットします。

プログラムを再始動する地点を設けたり、データベース変更をコミットする以外にも、 BMP プログラムや IMS データ共用を行っているプログラムでは、チェックポイント呼び出しを出すと、他のプログラムで使用できるようにデータベース・セグメントが解放されます。

バッチ・プログラムまたは BMP プログラムがチェックポイント要求を出すと、IMSIMS は、チェックポイント ID を含むレコードを IMS システム・ログに書き込みます。

実行中にアプリケーション・プログラムがある位置に到達し、そこで、ユーザーがその位置までに行った変更がすべて物理的にデータベースに入っているか確かめたい場合は、チェックポイント要求を出してください。なんらかの状況が起こって、プログラムが実行を終える前に障害が発生した場合、データベースは元の状態に復元されなければなりません。 そのデータベースが部分的に更新された状態のままにならないように、データベースに対して行われた変更をバックアウトして、他のアプリケーション・プログラムがアクセスできるようにしなければなりません。

プログラムが長時間稼働する場合、プログラム内にチェックポイントを取ると、バックアウトしなければならない変更の量を削減することができます。チェックポイントを取った場合、プログラムが異常終了しても、チェックポイント以降に行われたデータベース更新だけをバックアウトすれば済みます。 また、最初からプログラムを再始動しなくても、チェックポイント要求が出された地点から再始動することができます。

チェックポイント呼び出しを出すと、データベース内でのユーザーの位置が取り消されます。

Get Unique 呼び出しを出す直前にチェックポイント呼び出しを出してください。これによって、チェックポイントが取られた後のデータベース・レコード内でのユーザー位置が再設定されます。

チェックポイントのタイプ

チェックポイント呼び出しには、基本とシンボリックの 2 タイプがあります。どちらのタイプも、プログラムによるデータベースへの変更をコミットし、プログラムを再始動できる位置を確立します。

バッチ・プログラムおよび BMP プログラムは、CHKP 呼び出しを使用して、基本チェックポイント呼び出しを出すことができます。 基本チェックポイント呼び出しを出す場合は、プログラムが異常終了した後に再始動するようにコードを作成しなければなりません。

バッチ・プログラムおよび BMP プログラムの場合のみ、シンボリック・チェックポイント呼び出しを出すことができます。 CHKP 呼び出しを使用して、シンボリック・チェックポイント呼び出しを出すことができます。基本チェックポイント呼び出しと同様に、シンボリック・チェックポイント呼び出しでも、データベースに対する変更をコミットし、プログラムが再始動できる地点を設けることができます。加えて、シンボリック・チェックポイント呼び出しでは以下のことを行うことができます。

- 拡張再始動呼び出しを行って、プログラムの再始動およびリカバリーを単純化する。
- プログラム内に、最大 7 つのデータ域を指定してチェックポイントを設けることを可能にする。プログラムを再始動する場合、再始動呼び出しはプログラムが異常終了したときと同じ方法で上記の領域を復元します。

チェックポイント ID の指定

プログラムが出すチェックポイント呼び出しにはそれぞれ識別、すなわち ID が必要です。チェックポイント ID は、必ず長さを 8 バイトにし、印刷可能な EBCDIC 文字を含めます。

プログラムを再始動したい場合、プログラムを再始動したいチェックポイントの ID を指定することができます。 プログラムが再始動すると、IMS は、ユーザーが指定したものと一致する ID を持つチェックポイント情報を検索するのでこの ID は重要です。 IMS が検出した最初の一致する ID が、プログラムがリスタート・ポイントになります。 したがって、それぞれのアプリケーション・プログラムの内部でも、異なるアプリケーション・プログラム間でも、チェックポイント ID は固有でなければなりません。 チェックポイント ID が固有でないと、ユーザーが指定したチェックポイントから IMS がプログラムを再始動するとは限りません。

プログラムの内部および異なるプログラム間で、チェックポイント ID が固有かどうか確認する方法の 1 つは、次の順に ID を構成することです。

- ユーザーのプログラムを固有に識別する 3 バイトの情報

- プログラム内部での ID となる 5 バイトの情報。例えば、チェックポイント・コマンドまたは呼び出しごとに 1 ずつ増える値、または、プログラムを始動する際に TIME マクロを出すと獲得できるシステム時刻の一部などがあります。

チェックポイントの頻度の指定

チェックポイント要求の頻度を決定するには、プログラムのタイプ、およびパフォーマンスの特性を考慮しなければなりません。

バッチ・プログラムの場合

バッチ・プログラムにおいてチェックポイント要求を出す頻度を決定する際は、障害が起こった場合にプログラムをバックアウトし、再処理するために必要な時間を考慮してください。例えば、プログラムが行う処理をバックアウトするのに時間がかかると思われる場合は、それだけ頻繁にチェックポイントを設定する必要があります。

プログラム全体をバックアウトしなければならない場合は、プログラムの一番初めで、チェックポイント要求を出してください。IMS は、ユーザーが指定したチェックポイントまで、データベース更新をバックアウトします。プログラムの開始後の最初のチェックポイントの前にデータベースが更新された場合は、IMS はこれらのデータベース更新をバックアウトできません。

データ共用環境の場合は、オンライン・システムで他のプログラムとリソースを共用することによって、ユーザーのオンライン・システムが受ける影響について考慮してください。オンライン・プログラムとデータを共用しているバッチ・プログラムの場合は、より頻繁にチェックポイント呼び出しを出して、リソースに対する競合を少なくしてください。

バッチ・プログラムの設計は常に、チェックポイントおよび再始動を考慮して行ってください。 そのときはチェックポイントのサポートが必要なくても、あらかじめチェックポイント呼び出しを組み込んだほうが、後で組み込むよりも簡単です。 チェックポイント呼び出しが組み込まれている場合、バッチ・プログラムを BMP プログラムやデータ共用を行うバッチ・プログラムに変換することが容易になります。

BMP プログラムの場合

BMP プログラムでチェックポイント要求を出す頻度を決定する際は、 CICS オンライン・システムのパフォーマンスを考慮してください。これらのプログラムは CICS オンライン・トランザクションとリソースを共用しているため、CICS オンライン・プログラムがリソースを獲得するために待機する必要がないように、チェックポイント要求を出してセグメントを解放してください。

チェックポイント・ログ・レコードの印刷

IMS のファイル選択および印刷フォーマット設定プログラム (DFSERA10) を使用すると、チェックポイント・ログ・レコードを印刷できます。このユーティリティーを使用すると、タイプ、含まれるデータ、またはデータ・セット内の順次位置に基づいて、ログ・レコードを選択および印刷することができます。 チェックポイント・レコードは、タイプ 18 のログ・レコードです。「*IMS V15* システム・ユーテ

ィリティー」で、このプログラムについて説明しています。

関連概念:

75 ページの『CICS システムのパフォーマンスの最大化』

データベース変更のバックアウト

ユーザーのプログラムが異常終了した場合、データベースは以前の状態に復元され、コミットされていない変更はバックアウトされなければなりません。BMP プログラムまたは CICS オンライン・プログラムによって行われた変更は、自動的にバックアウトされます。 バッチ・プログラムによって行われたデータベース変更は、システム・ログが DASD に記録されているかどうかによって、バックアウトされる場合とされない場合があります。

バッチ・プログラムの場合

バッチ・プログラムが異常終了した場合に起こる状態、および、データベースのリカバリーのしかたは、システム・ログのストレージ・メディアによって異なります。 システム・ログを DASD に保管するか、テープに保管するかを指定できます。

- **DASD** にシステム・ログを保管する場合

  JCL で BKO=Y をコーディングすると、最後のコミット・ポイント以降にバッチ・プログラムがデータベースに行った変更を、IMS が動的にバックアウトするように指定することができます。 (データを共用するバッチ・プログラムで) デッドロックが検出された場合のように、IMS が検出できる障害が起こった場合、IMS はバッチ・プログラムを動的にバックアウトします。

  DASD ロギングを行うことにより、バッチ・プログラムが、 ROLL に加えて、ロールバック (ROLB) システム・サービス要求を出すことも可能になります。 ROLB 要求によって、IMS は、最後のコミット・ポイント以降にプログラムがデータベースに行った変更を動的にバックアウトし、アプリケーション・プログラムに制御を返します。

  データベース変更を動的にバックアウトすると、次のような利点があります。

  – 障害が起きたプログラムによってアクセスされたデータを、すぐに他のプログラムで使用することができます。 バッチ・バックアウトを使用されない場合は、IMS バッチ・バックアウト・ユーティリティーが実行されてデータベース変更をバックアウトするまで、他のプログラムはそのデータにアクセスできません。
  – 2 つのプログラムがデッドロックされた場合、そのうちの 1 つは処理を続けることができます。バッチ・バックアウトを使用しない場合は、プログラムは両方とも失敗します。 （これは、データを共用しているバッチ・プログラムのみに適用されます。）

  動的バックアウトを使用せずに、IMS バッチ・バックアウト・ユーティリティーを実行して変更をバックアウトすることもできます。
- テープにシステム・ログを保管する場合

システム・ログがテープに記録されている場合、バッチ・アプリケーション・プログラムが異常終了したときには、IMS バッチ・バックアウト・ユーティリティーを使用して、プログラムがデータベースに対して行った変更をバックアウトしなければなりません。

関連資料 : さらに詳しくは、 *IMS V15* データベース・ユーティリティー を参照してください。

BMP プログラムの場合

プログラムが異常終了した場合、最後のコミット・ポイント以降にプログラムが行った変更はバックアウトされます。 システムに障害が起こった場合、あるいは CICS 制御領域または DBCTL が異常終了した場合には、DBCTL 緊急時再始動によって、最後のコミット・ポイント以降にプログラムが行った変更はすべてバックアウトされます。 DBCTL が変更をバックアウトするので、IMS バッチ・バックアウト・ユーティリティーを使用する必要はありません。 変更をすべてバックアウトする必要がある場合、ROLL システム・サービス呼び出しを使用して、データベース変更を動的にバックアウトすることができます。

プログラムの再始動

シンボリック・チェックポイント呼び出しを出した場合 (バッチ・プログラムおよび BMP プログラムの場合)、拡張再始動システム・サービス要求 (XRST) を使用して、異常終了したプログラムを再始動することができます。

プログラムが異常終了すると、XRST 呼び出しは、そのプログラムのデータ域をそのときの状態に復元し、プログラムが異常終了する前に出した最後のチェックポイント要求からプログラムを再始動します。

基本チェックポイント呼び出しを使用した場合 (バッチ・プログラムおよび BMP プログラムの場合)、プログラムが異常終了した場合に、最後のチェックポイントから再始動できるように、必要なコードを作成してください。

最後のチェックポイントからプログラムを再始動する方法として、 HDAM データベース内に位置変更データを格納することもできます。ユーザーのプログラムは、位置変更情報を含むデータベース・レコードを HDAM データベースに書き込みます。ユーザーのプログラムは、定期的にこのレコードを更新します。プログラムが異常終了した場合、データベース・レコードは削除されます。 XRST 呼び出しが完了すると、入出力域には再始動で使用したチェックポイント ID が必ず入っています。 通常 XRST は、8 バイトのシンボリック・チェックポイント ID と、それに続けて 4 個のブランクを返します。 8 バイトの ID がすべてブランクの場合には、XRST は 14 バイトのタイム・スタンプ ID を戻します。 PCB の状況コードも調べてください。 XRST 呼び出しが正常に終了すれば、状況コードはブランクの行になります。

関連概念:

837 ページの『IMS Java 従属領域リソース・アダプターを使用した JBP アプリケーションの開発』

CICS プログラムのデータの可用性に関する考慮事項

プログラムでアクセスする必要があるデータが、利用不可という場合があります。データが利用不可の場合は、以下の機能を使用します。

データベースの使用不可状況

データベース全体が読み取りと更新の両方で使用不可になってしまう条件は次のとおりです。

- データベースに STOP コマンドが出された場合
- データベースに DBRECOVERY (DBR) コマンドが出された場合
- データベースに対する DBRC 許可が失敗した場合

データベースが、読み取りはできるが更新ができなくなる条件は、以下のとおりです。

- データベースに DBDUMP コマンドが出された場合
- データベースのアクセス値が RD (読み取り) である場合

データ共用環境では、これらのいずれかの状態の原因になったコマンドまたはエラーは、データを共用している他のシステムで生じた場合もあります。

データベースが使用不能な場合に、プログラムがスケジュールされているか、実行中のプログラムが PSB をスケジュールすることができるかは、プログラムおよび環境のタイプによって異なります。

- バッチ・プログラムの場合

  バッチ・プログラムがアクセスできるデータベースのいずれかが使用不能な場合、IMS はそのバッチ・プログラムをスケジュールしません。

  データ共用以外の環境では、データベースは現在 DB/DC 環境に対して許可されているため、データベースへの DBRC 許可が失敗する場合があります。 データ共用環境では、データベースをリカバリーまたはダンプするための CICS または DBCTL マスター端末グローバル・コマンドによって、データベースがバッチ・プログラムで使用不能になることがあります。

  以下の条件のみでは、バッチ・プログラムは初期設定時に障害を起こしません。

  – PCB が HALDB を参照する。
  – DBRC の使用が抑制される。

  ただし、DBRC を使用しない場合、HALDB に PCB を使用するデータベース呼び出しは許可されません。このプログラムが使用不能データをセンシティブと見なす場合は、このような呼び出しにより状況コード BA が出されます。そうでない場合は、このような呼び出しによりメッセージ DFS3303I とその後に ABENDU3303 が出されます。
- DBCTL 環境でのオンライン・プログラムまたは BMP プログラムの場合

  この環境で実行しているプログラムが、1 つまたは複数の使用不能な全機能データベースを含む PSB でスケジュールしようとすると、許可されます。 プログラムが使用不能データベースにアクセスしようとしなければ、正常に機能します。

プログラムが使用不能なデータベースにアクセスしようとした場合は、データベースが使用可能ではあるけれどもその一部が使用できない場合と同じ結果になります。

データベース内の一部のデータが使用不可である状況

データベース全体が使用不能な状況に加えてさらに、データの一部が使用不能な状況もあります。一例としては、データ共用状態で発生した障害があります。このケースでは、データを共用する 1 つの IMS システムに障害発生時にその IMS システムがどのロックを保持していたのかを、IMS システムが認識します。この IMS システムは、データベースの使用を続けますが、障害が起こった IMS がその際にロックしたままのデータへのアクセスは拒否します。

バッチ・プログラム、オンライン・プログラム、あるいは BMP プログラムは、DBCTL 環境で作動できます。オンライン・プログラムまたは BMP プログラムは、データベース全体が使用不可なときにスケジュールされている場合があります。 以下のオプションは、これらのプログラムがデータにアクセスしようとしたときに、データベース全体が使用不可な場合にも、データベースの一部だけが使用不可な場合にも適用されます。

これらの環境で稼働しているプログラムが、データが使用不能であることをセンシティブと見なすか、センシティブと見なさないかを任意で選択することができます。

- プログラムが、データが使用不能であることをセンシティブと見なさない場合に、使用不能データにアクセスしようとすると、プログラムは 3303 異常終了によって中断されます。オンライン・プログラムの場合は、疑似異常終了します。バッチ・プログラムの場合は、真に異常終了します。しかし、動的割り振りが失敗したためにデータベースが使用不能になった場合は、呼び出しの結果、AI (オープン不可) 状況コードが返されます。
- プログラムが、データを使用できないことをセンシティブと見なす場合、使用不能データへのアクセスを試みると、IMS は状況コードを返して、その要求を処理することができないことを通知します。プログラムはその後、適当な処置を取ります。プログラムが、使用不能データをセンシティブと見なさない場合、プログラムには、 IMS が取るのと同じ処置を開始する機能が存在します。

プログラムは、INIT 呼び出しまたは ACCEPT STATUS GROUP A コマンドを出して、プログラムが、使用不能データをセンシティブと見なしており、使用不能データにアクセスしようとしたときに出された状況コードを、受け入れ可能であることを、IMS に通知します。また、INIT 要求を使用して、 PSB 内のそれぞれの PCB のデータ可用性を判別することができます。

SETS または SETU 機能および ROLS 機能

SETS または SETU 要求および ROLS 要求を使用して、アプリケーションで、全機能データベースの状態を保存する複数の位置を定義することができます。

アプリケーションは、後でこれらの位置に戻ることができます。 DL/I 要求を開始して機能を実行する前に SETS または SETU 要求を出すと、データ使用不能のためにプログラムが機能を完了できない場合、プログラムは後で ROLS 要求を出すことができます。

ROLS 要求を出すと、プログラムは IMS アクティビティー状態を、 SETS または SETU 呼び出しが出される前の状態にロールバックすることができます。

制約事項: SETS または SETU、および ROLS のみが、IMS の更新をロールバックします。ただし、CICS ファイル制御または一時データを使用して行われた更新はロールバックしません。

また、ROLS 呼び出しまたはコマンドを使用して、最後のチェックポイント以降のデータベース更新アクティビティーをすべて取り消すこともできます。

IMS バッチ・プログラムでの STAE または ESTAE、および SPIE の使用

IMS では、IMS バッチ領域で STAE または ESTAE ルーチンを使用して、データベース・ロギングおよび各種のリソース終結処理機能が確実に完了するようにします。

STAE または ESTAE 機能には、以下の 2 つの重要な特性があります。

- IMS は、データベース保全性およびリソースの制御を完全に STAE または ESTAE 機能に頼っています。
- STAE または ESTAE 機能はアプリケーション・プログラムでも使用可能です。

この 2 つの要因があるため、プログラムと STAE または ESTAE 機能との間の関係を明確に理解しておかなければなりません。

通常、バッチ・アプリケーション・プログラムでは STAE または ESTAE 機能を使用しないでください。ただし、STAE または ESTAE 機能の使用が必要であると思われる場合には、次の基本規則に従ってください。

- 環境が STAE または ESTAE 処理をサポートしている場合、アプリケーション・プログラムの STAE または ESTAE ルーチンは、常に IMS STAE または ESTAE ルーチンよりも優先的に制御権を持っています。したがって、アプリケーション・プログラム内で次の手順を守って、IMS STAE または ESTAE 出口ルーチンが制御を受け取るようにしてください。
  - STAE または ESTAE ルーチンは一度だけ、必ず最初の DL/I 呼び出しの前に確立してください。
  - STAE または ESTAE 機能を使用する場合は、アプリケーション・プログラムは IMS 異常終了コードを変更しないでください。
  - STAE または ESTAE ルーチンから出るときには、RETRY オプションを使用しないでください。代わりに、STAE または ESTAE 処理の終了時に CONTINUE-WITH-TERMINATION 標識を返してください。 アプリケーション・プログラムが RETRY オプションを指定している場合は、 IMS STAE または ESTAE 出口ルーチンは終結処置を行う制御権を持たないことに注意してください。したがって、システムおよびデータベース保全性に危険が伴います。
- アプリケーション・プログラムの STAE/ESTAE 出口ルーチンは、DL/I 呼び出しを出してはなりません。これは、元の異常終了が、アプリケーションと IMS の間の問題によって発生した可能性があるためです。 これを行うと、データベース保全性の損失の恐れがある STAE/ESTAE への再帰的入力が行われるか、またはチェックポイントを取る際に問題が発生することがあります。

IMS データベースの動的割り振り

ライブラリー内の IMS データベースについての JCL 情報を指定するには、バッチ・ジョブの JCL や DBCTL の JCL の代わりに、動的割り振り機能を使用します。

動的割り振りを使用している場合、動的割り振りに定義されたデータベース・データ・セットについては JCL DD ステートメントを組み込まないでください。データベース管理者 (DBA) または同等の技能のある専門家に相談して、動的割り振りに定義されているデータベースを判別してください。

関連資料: 動的割り振りの定義の詳細については、「*IMS V15* システム定義」の DFSMDA マクロの項を参照してください。

第 5 章 データベース・オプションの要件収集

アプリケーションがアクセスするデータベースの階層を設計した後で、 DBA はどのデータベース・オプションがアプリケーション要件に最も適しているかを評価します。それらのオプションを使用するかどうかは、収集されたアプリケーション要件によって異なります。有効なデータベースを設計するために、DBA は個々のアプリケーションに関する情報を必要とします。

関連概念:

46 ページの『メッセージ処理: メッセージ処理プログラム』

データ・アクセスの分析

DBA は、データベースを使用する大半のプログラムが取るデータ・アクセス方式に基づいて、データベースのタイプを選択します。

IMS データベースは、使用するアクセス方式に従って分類されます。以下に示すのは、定義可能なデータベース・タイプのリストです。

HDAM (階層直接アクセス方式)

PHDAM (区分階層直接アクセス方式)

HIDAM (階層索引直接アクセス方式)

PHIDAM (区分階層索引直接アクセス方式)

MSDB (主記憶データベース)

DEDB (高速処理データベース)

HSAM (階層順次アクセス方式)

HISAM (階層索引順次アクセス方式)

GSAM (汎用順次アクセス方式)

SHSAM (単純階層順次アクセス方式)

SHISAM (単純階層索引順次アクセス方式)

重要: PHDAM および PHIDAM は、データベース・タイプ HDAM および HIDAM の区分化バージョンです。したがって、HDAM および HIDAM データベース・タイプの対応する説明は、PHDAM および PHIDAM にも適用されます。

DBA による決定に役立ついくつかの収集可能な情報から、以下のような質問に答えを出すことができます。

- データベース・レコードにアクセスするには、プログラムはまずレコードのルートにアクセスしなければなりません。 各プログラムはルート・セグメントにどのようにアクセスしますか。

  直接

  順次

  その両方

- データベース・レコード内のセグメントは、ルート・セグメントの従属セグメントです。 各プログラムは各データベース・レコード内のセグメントにどのようにアクセスしますか。

  直接

  順次

  その両方

  データベース・レコードへのアクセスと、レコード内のセグメントへのアクセスとの区別を念願におくことは重要なことです。 プログラムは、データベース・レコードには順次にアクセスしますが、レコード内ではセグメントには直接アクセスできます。これらには相違があるため、アクセス方式の選択に影響を与える場合があります。
- プログラムはどの程度までデータベースを更新しますか。

  新しいデータベース・レコードを追加しますか。

  既存のデータベース・レコードに新しいセグメントを追加しますか。

  セグメントまたはデータベース・レコードを削除しますか。

ここでも、データベース・レコードの更新と、データベース・レコード内のセグメントの更新の違いに注意してください。

直接アクセス

直接アクセス処理を行うことの利点は、直接処理でも順次処理でも良い結果を得られることです。 直接アクセスとは、ランダム化ルーチンまたは索引を使用することです。これによって、IMS はデータベース内でのデータベース・レコードの順序に関係なく、ユーザーが必要とするあらゆるデータベース・レコードを見つけることができます。

IMS 全機能には、4 つの直接アクセス方式があります。

- HDAM および PHDAM は、ルート・セグメントを保管したり探し出すのに、ランダム化ルーチンを使用して、データを直接に処理します。
- HIDAM および PHIDAM は、ルート・セグメントの直接処理を可能にする索引を使用します。

直接アクセス方式では、ポインターを使用して、データベース・レコードのセグメント間の階層的な関係を保ちます。 ポインターを追うことによって、IMS はそれより前のパス内のすべてのセグメントを通過しなくてもセグメントのパスにアクセスすることができます。

直接アクセスによって満たされる要件の一部は、以下のとおりです。

- 索引またはランダム化ルーチンを使用したルートの高速直接処理
- 索引を使用する、HIDAM および PHIDAM によるデータベース・レコードの順次処理
- ポインター使用によるセグメントのパスへの高速アクセス

さらに、直接アクセス・データベースからデータを削除すると、すぐに新しいスペースが使用可能になります。 これによって、効率的なスペース使用が可能になり、

データベースを頻繁に再編成する必要がなくなります。 直接アクセス方式は、内部でポインターおよびアドレスの保守を行います。

直接アクセスの欠点は、ポインターのために IMS オーバーヘッドが大きくなることです。 しかし、ユーザーのデータ・アクセス要件をすべて満たした場合は、直接アクセスは順次アクセス方式よりも効率的です。

基本直接処理: HDAM

HDAM は、通常は直接にアクセスされ、時々順次にアクセスされるデータベースの場合に効率的です。 HDAM は、ランダム化ルーチンを使用してルート・セグメントを探しだし、選択されたポインター・オプションに応じて、従属セグメントを互いにチェーニングさせます。 HDAM が使用できる z/OS アクセス方式は、仮想記憶アクセス方式 (VSAM) およびオーバーフロー順次アクセス方式 (OSAM) です。

重要: PHDAM は、HDAM データベース・タイプの区分化バージョンです。したがって、HDAM データベース・タイプの対応する説明は、PHDAM にも適用されます。

HDAM によって満たされる要件は、以下のとおりです。

- HDAM は、ランダム化ルーチンを使用してルート・セグメントを探しだすため、ルート・キーを使用して、ルートに直接アクセスが可能になります。
- 従属セグメントのパスに直接アクセスができます。
- 新しいデータは最も近くにある使用可能なスペースに入るので、新しいデータベース・レコードおよびセグメントを追加できます。
- 削除によって生じたスペースは、新しいセグメント用に利用できるので、データベース・レコードおよびセグメントを削除できます。

HDAM の特性

HDAM データベースには以下のような特性があります。

- どこにでもルート・セグメントを保管できます。ランダム化ルーチンがルート・セグメントを探しだすため、ルート・セグメントは、順番になっていなくてもかまいません。
- ランダム化ルーチンを使用して、相対ブロック番号と、そのブロック内でルート・セグメントを指し示す、ルート・アンカー・ポイント (RAP) を探しだします。
- ルートが物理的順序でチェーニングされている、RAP にアクセスします。すると、ルート・アンカー・ポイントからチェーニングされたルート・セグメントが戻されます。したがって、HDAM からのルート・セグメントの順次リトリーブは、ランダム化ルーチンの結果に基づくものではなく、しかもランダム化ルーチンが、ルート・セグメントをキー・シーケンスにランダム化しない限り、キー・シーケンスになりません。
- ランダム化モジュールがルート・セグメントの物理順序をキー・シーケンスにしないと、呼び出しに対して意図した結果が得られない場合があります。例えば、ルート・キー値以下となるように修飾されたルート・セグメントへの GU 呼び出

しでは物理順序でスキャンされ、最初のブロックの最初の RAP を探します。この結果、修飾と一致するセグメントが存在するのに、検出できないという状態が起こることがあります。

従属セグメントの場合は、HDAM データベースには、以下のような特性があります。

- どこにでも保管できます。
- 1 つのデータベース・レコードのセグメントをすべて、ポインターを使用してチェーニングします。

HDAM の稼働の概要

このトピックには診断、変更、およびチューニングに関する情報が含まれています。

データベース・レコードが HDAM データベースに保管される場合、 HDAM はそれぞれの物理ブロックの最初に 1 つまたは複数の RAP を保持します。 RAP はルート・セグメントを指し示します。 HDAM はまた、各物理ブロックの最初にポインターを保持します。そのポインターは、ブロック内のすべてのフリー・スペースを指し示します。あるセグメントを挿入するときに、HDAM はポインターを使用して、物理ブロック内のフリー・スペースを指し示します。 HDAM データベース内でルート・セグメントを探し出すには、 HDAM にルート・キーを渡してください。ランダム化ルーチンによって、相対物理ブロック番号、およびルート・セグメントを指し示す RAP が与えられます。指定された RAP 番号によって、 HDAM には物理ブロック内のルートの位置が与えられます。

HDAM は、ルート・セグメントおよび従属セグメントをデータベース内のどこにでも配置することができますが、ルートと従属を近くにしておく HDAM オプションを選択することをお勧めします。

HDAM のパフォーマンスは、ユーザーが使用するランダム化ルーチンに大きく影響されます。 非常に良好なパフォーマンスを実現することもできますが、それは以下のような要因によっても左右されます。

- ユーザーが使用するブロックのサイズ
- ブロックごとの RAP の数
- 異なるセグメントをつなぐパターン。以下の 2 つの方法で、データベース・レコードのセグメントを互いにチェーニングすることができます。
  - ルートで始まる階層順
  - 親から従属への順 (親は従属のパスのそれぞれに対してポインターを持つ)

HDAM を使用して、データベース・レコードへのルート・キーによる順次アクセスを行う場合、ユーザーは、副次索引または物理キー・シーケンスにルートを保管するランダム化ルーチンを使用する必要があります。

直接処理および順次処理: HIDAM

HIDAM は、直接処理と順次処理の割合がほとんど等しい場合に最も効率的なアクセス方式です。

重要: PHIDAM は、HIDAM データベース・タイプの区分化バージョンです。したがって、HIDAM データベース・タイプの対応する説明は、PHIDAM にも適用されます。

使用できる z/OS アクセス方式は、VSAM および OSAM です。 HIDAM によって満たされる特定要件は、以下のとおりです。

- ルート・キーによるレコードへの直接または順次アクセスが可能です。
- 従属セグメントのパスに直接アクセスができます。
- 新しいデータは最も近くにある使用可能なスペースに入るので、新しいデータベース・レコードおよびセグメントを追加できます。
- 削除によって生じたスペースは、新しいセグメント用に利用できるので、データベース・レコードおよびセグメントを削除できます。

HIDAM は、直接処理および順次処理の混合を含む、ほとんどの処理要件を満たすことができます。 しかし、HIDAM は、従属セグメントへの順次アクセスにはそれほど効率的ではありません。

HIDAM の特性

ルート・セグメントの場合は、HIDAM データベースには以下の特性があります。

- あらかじめキー・シーケンスにルート・セグメントをロードします。
- スペースがある所ならどこにでも新しいルート・セグメントを保管することができます。
- 索引を使用してルートを見つけます。ルートはルートのキー値を指定して要求し識別します。

従属セグメントの場合は、HIDAM データベースには以下のような特性があります。

- セグメントをどこにでも保管することができ、セグメントをなるべく互いに近づけておきます。
- ポインターを使用して、データベース・レコードのすべてのセグメントを互いにチェーニングします。

HIDAM の稼働の概要

このトピックには診断、変更、およびチューニングに関する情報が含まれています。

HIDAM は 2 つのデータベースを使用します。 基本データベースにはデータが保管されます。 索引データベースには、キー・フィールドの順ですべてのルート・セグメントの項目が含まれます。 それぞれのキー項目について、索引データベースは、基本データベース内でのルート・セグメントのアドレスを含んでいます。

ルートにアクセスする場合は、そのルートに対するキーを指定してください。 HIDAM は索引内のキーを探してそのルートのアドレスを突きとめ、それから基本データベースでそのルートを見つけます。

HIDAM は、従属セグメントを互いにチェーニングするので、従属セグメントにアクセスする場合、 HIDAM は 1 つのセグメント内のポインターを使用して、階層内の次のセグメントを探し出します。

データベース・レコードを直接処理する場合は、HIDAM は索引を検索してルートを探し、そのルートからセグメントを探し出します。 HIDAM は、ポインターを通じて従属セグメントを探し出します。

データベース・レコードを順次処理する場合には、そのデータベースについて DBD 内の特定のポインターを指定することができ、これによって、IMS は索引を使用して次のルート・セグメントを探し出す必要がなくなります。 これらのポインターは、ルートを互いにチェーニングします。ルートを互いにチェーニングしない場合は、HIDAM は必ず索引を探してルート・セグメントを探し出します。 データベース・レコードを順次に処理する場合、HIDAM は索引内のキー・シーケンスにルートにアクセスします。 これは順次処理だけに適用されます。ルート・セグメントを直接処理したい場合、 HIDAM は他のルート・セグメント内のポインターは使用せずに索引を使用し、ユーザーが要求したルート・セグメントを探し出します。

主記憶データベース: MSDB

最も頻繁にアクセスされるデータを保管するには、MSDB を使用します。 MSDB は、金融業における、総勘定元帳アプリケーションなどのアプリケーションに適しています。

推奨事項: 新規の高速機能データベースを開発する場合は、MSDB の代わりに DEDB を使用します。端末関連 MSDB および端末関連鍵を使用する非端末関連 MSDB は、サポートされなくなりました。非端末関連鍵を使用する非端末関連 MSDB は引き続きサポートされていますが、既存の MSDB はすべて DEDB に変換することを検討してください。 MSDB-to-DEDB 変換ユーティリティーを使用することができます。

MSDB の特性

MSDB は仮想記憶域に常駐し、これによって、アプリケーション・プログラムがアクセスするのに必要な入出力アクティビティーを避けることができます。 MSDB には、端末関連と非端末関連の 2 種類があります。

端末関連 MSDB では、各セグメントを 1 つの端末が所有します。各端末は 1 つのセグメントだけを所有します。このタイプの MSDB の使用目的としては、各セグメントが論理端末に関連するデータを含むアプリケーションなどがあります。このタイプのアプリケーションでは、 (おそらく報告のために) プログラムはデータを読むことができますが、そのデータを更新することはできません。非端末関連 MSDB は、同じ時間枠の間に多くのユーザーが必要とするデータを保管します。このデータは、すべての端末から更新および読み取りが可能です。 (例えば、リアルタイム在庫管理アプリケーションでは、在庫の減少を多くのキャッシュ・レジスターから記録することができます。)

MSDB の稼働の概要

このトピックには診断、変更、およびチューニングに関する情報が含まれています。

MSDB セグメントは、ルート・セグメントとしてのみ保管されます。 ポインターのタイプは、順方向チェーン・ポインターのみが使用されます。 このポインターは、データベース内のセグメント・レコードを接続します。

高速処理データベース: DEDB

DEDB は、多量のデータへのアクセスを可能にし、効率的にストレージを利用できるように設計されています。 DEDB によって満たされる基本的な要件は、高レベルのデータ可用性です。

DEDB の特性

DEDB は階層データベースであり、最大 15 の階層レベル、および最大 127 のセグメントのタイプを持つことができます。直接従属セグメントおよび順次従属セグメントの両方を含むことができます。順次従属セグメントは、データベースにコミットされた日時順に保管されるため、アプリケーションをジャーナル処理するのに便利です。

DEDB は、HIDAM または HDAM データベースで使用可能な機能およびオプションのサブセットをサポートします。例えば、DEDB は、論理的に関連したセグメントまたは 1 次索引を使用したアクセスをサポートしません。副次索引を使用したアクセスはサポートしています。

DEDB の稼働の概要

このトピックには診断、変更、およびチューニングに関する情報が含まれています。

DEDB を複数の区域に分割し、それぞれの区域に異なるデータベース・レコードの集合を入れることができます。 DEDB エリア内のデータは、VSAM データ・セットに保管されます。 ルート・セグメントは、エリアのルート・アドレス可能部分に保管され、素早くアクセスするために、ルートの近くに直接従属セグメントが保管されます。ルートの近くに保管できない直接従属セグメントは、その区域の独立オーバーフロー部分に保管されます。 順次従属セグメントは、素早く挿入できるように、区域の終わりの順次従属部分に保管されます。 各エリア・データ・セットは、最大 7 つのコピーを持つことができ、データを容易に各アプリケーション・プログラムで使用できるようにします。

順次アクセス

順次アクセス方式を使用する場合、データベース内のセグメントは階層順に、連続的に保管され、ポインターは使用されません。

IMS 全機能には、2 つの順次アクセス方式があります。直接アクセス方式の場合と同様に、一方には索引があり、他方にはありません。

- HSAM は、ルート・セグメントおよび従属セグメントを順次に処理するだけです。
- HISAM はデータを順次に処理しますが、索引を持っているのでレコードに直接アクセスできます。 HISAM は主に、従属セグメントを順次に処理し、データベース・レコードを直接処理するためのものです。

順次アクセスによって満たされる一般要件は以下のとおりです。

- 高速順次処理が可能です。
- HISAM を使用したデータベース・レコードの直接処理が可能です。
- 順次アクセス方式は、ポインターによってではなく、隣接性によってセグメントを関連付けるため、ストレージでの IMS オーバーヘッドが少なくなります。

順次アクセス方式の使用には、以下の 3 つの欠点があります。

- 順次アクセス方式では、階層の右端のセグメントへのアクセスが遅くなります。これは、HSAM および HISAM が、それらのセグメントに到達するのに他のすべてのセグメントを読み取らなければならないためです。
- HISAM は削除されたセグメントのスペースを再利用するため、また、データベース・レコードの論理レコードを物理的に隣接させておくために、頻繁に再編成しなければなりません。
- HSAM データベースを更新することはできません。 データを変更するためには、新しいデータベースを作成しなければなりません。

順次処理のみ: HSAM

HSAM は、階層アクセス方式であり、順次処理しか行えません。 HSAM データベースのデータをリトリーブすることはできますが、更新することはできません。 HSAM が使用できる z/OS アクセス方式は、QSAM および BSAM です。

以下の状態には HSAM が適しています。

- データまたは統計を収集するためにデータベースを使用しているが、データベースを更新する必要がない場合
- データを順次に処理するだけの場合

HSAM の特性

HSAM は、ユーザーが要求した順にデータベース・レコードを保管します。ユーザーは、レコードおよび従属セグメントを順次に処理することしかできません。それは、ユーザーがロードした順にということです。 HSAM は、従属セグメントを階層順に保管します。

HSAM の稼働の概要

このトピックには診断、変更、およびチューニングに関する情報が含まれています。

HSAM データベースは、非常に単純なデータベースです。 データは 1 セグメントずつ順番に階層順に保管され、ポインターや索引は使用されません。

基本順次処理: HISAM

HISAM は、階層順にセグメントを保管するアクセス方式であり、ルート・セグメントを位置付けるために索引を使用します。オーバーフロー・データ・セットもあります。論理レコードの最後に到達するまで、論理レコード内にセグメントを保管してください。論理レコードのスペースを使い果たしたが、そのデータベース・レコードに属するセグメントが残っている場合には、残りのセグメントをオーバーフロー・データ・セットに保管してください。 HISAM が使用できるアクセス方式は、VSAM および OSAM です。

HISAM は以下のことに適しています。

- ルート・キーによるレコードの直接アクセス
- レコードの順次アクセス
- 従属セグメントの順次アクセス

ユーザーの行う処理に上記のいくつかの特性はあっても、次のように、HISAM を選択することが必ずしも良いとはいえない場合があります。

- 従属セグメントに直接アクセスしなければならない場合
- 多数の挿入および削除を行う場合
- データベース・レコードの多くが、平均サイズを超えており、オーバーフロー・データ・セットを使用しなければならない場合。 オーバーフロー・データ・セットにオーバーフローするセグメントには、追加の入出力が必要です。

HISAM の特性

データベース・レコードの場合、HISAM データベースには以下のような特性があります。

- レコードをキー・シーケンスに保管します。
- 索引を使用して、キー値によって特定のレコードを見つけることができます。

従属セグメントの場合は、HISAM データベースには以下のような特性があります。

- 各 HISAM データベース・レコードを、1 次データ・セット内の新しい論理レコード内で開始します。
- データベース・レコードが 1 次データ・セットに入りきらない場合、残りのセグメントを、オーバーフロー・データ・セット内の 1 つまたは複数の論理レコードに保管します。

HISAM の稼働の概要

このトピックには診断、変更、およびチューニングに関する情報が含まれています。

HISAM は、スペースをすぐには再使用しません。新しいセグメントを挿入する場合、HISAM データベースはデータをシフトして、新しいセグメントを保管するスペースをつくります。これにより、削除の後の未使用スペースがそのまま残ります。 HISAM スペースは、HISAM データベースを再編成する際に再利用されます。

IMS を介した z/OS ファイルへのアクセス: GSAM

GSAM を使用すると、IMS バッチ・アプリケーション・プログラムおよび BMP で、単純データベースとして順次 z/OS データ・セットにアクセスすることが可能になります。 GSAM が使用できる z/OS アクセス方式は、BSAM および VSAM です。GSAM データベースは、データベース・レコードとして定義された z/OS データ・セット・レコードです。レコードは 1 つの単位として処理されます。レコードにはセグメントまたはフィールドは含まれず、レコードの構造は階層的ではありません。 GSAM データベースには、z/OS、IMS、および CICS によってアクセスすることができます。

CICS 環境では、アプリケーション・プログラムは、呼び出し DL/I (または EXEC DL/I) バッチ・プログラム、あるいはバッチ指向 BMP プログラムから、 GSAM データベースにアクセスすることができます。しかし、CICS アプリケーションは、EXEC DLI を使用して GSAM データベースを処理することはできません。その場合は、IMS 呼び出しを使用しなければなりません。

バッチ指向 BMP あるいはバッチ・プログラムへ入力を送信し、出力を受信するには、通常は GSAM を使用します。 GSAM データベースを処理するには、アプリケーション・プログラムは、全機能データベースを処理する場合と似た呼び出しを出します。プログラムは、GSAM データベースから順次にデータを読んだり、GSAM データベースへ出力を送信することができます。

GSAM は順次アクセス方式です。 ユーザーは、出力データベースに順次にレコードを追加することだけができます。

z/OS を介した IMS データへのアクセス: SHSAM および SHISAM

2 つのデータベース・アクセス方式 (SHSAM および SHISAM) によって、 z/OS がデータ・セットとして使用できる単純階層データベースを得ることができます。

これらのアクセス方式は、z/OS ファイルから IMS データベースにデータを変換する場合に特に役立ちます。 SHISAM は索引付きですが、SHSAM には索引は付いていません。

これらのアクセス方式を使用する場合は、データベース・レコード全体を 1 つのセグメントとして定義します。 このセグメントは、IMS 制御情報またはポインターをまったく含んでいません。データ・フォーマットは、z/OS データ・セット内にある場合と同じです。 SHSAM が使用できる z/OS アクセス方式は、BSAM および QSAM です。 SHISAM は VSAM を使用します。

SHSAM データベースおよび SHISAM データベースには、IMS を使用しない z/OS アクセス方式でアクセスすることができます。この方法は変換の際に便利です。

データ構造における矛盾の解決方法についての理解

アプリケーション・プログラムが、階層内でフィールドおよびセグメントを処理する順番は、アプリケーションによってしばしば異なります。 DBA が 2 つ以上のプログラムがデータにアクセスしなければならないという矛盾を検出した場合、この問題を解決するための 3 つのオプションがあります。 以下のオプションは、それぞれ異なる種類の矛盾を解決します。

- アプリケーション・プログラムが、セグメント内のすべてのフィールドにアクセスする必要はない場合、または異なる順番でアクセスしなければならない場合、DBA は、そのプログラムにフィールド・レベル・センシティビティーを使用することができます。フィールド・レベル・センシティビティー によって、アプリケーション・プログラムは、セグメントに含まれるフィールドのサブセットだけにアクセスすることが可能になります。あるいは、アプリケーション・プログラムが、セグメントのフィールドをセグメント内の順序とは異なる順序で処理することが可能になります。

- アプリケーション・プログラムが、セグメントのキー・フィールド以外のあるフィールドによって、特定のセグメントにアクセスする必要がある場合、 DBA は、そのデータベースに対して副次索引を使用することができます。
- アプリケーション・プログラムが、異なる階層のセグメントを結び付ける必要がある場合、DBA は、論理関係を使用することができます。 論理関係を使用すると、アプリケーション・プログラムは、複数の階層のセグメントを含む論理階層を得ることができます。

関連概念:

32 ページの『マッピングの判別』

さまざまなフィールドの使用: フィールド・レベル・センシティビティー

フィールド・レベル・センシティビティーは、セグメント・センシティビティーが階層内のセグメントに適用するセキュリティーと同種のものを、セグメント内のフィールドに対して適用します。すなわち、アプリケーション・プログラムは、セグメント内のこれらのフィールド、および階層内のセンシティブと見なすセグメントにだけアクセスすることができます。

フィールド・レベル・センシティビティーによって、アプリケーション・プログラムは、セグメントを構成するフィールドのサブセットを使用したり、セグメント内のすべてのフィールドを異なる順序で使用したりすることもできます。アプリケーション・プログラムが処理を必要としないフィールドがセグメントに含まれている場合、フィールド・レベル・センシティビティーを使用すると、プログラムはフィールドを処理しないですむようになります。

フィールド・レベル・センシティビティーの例

従業員に関するデータを含むセグメントに、次の表に示すフィールドが含まれているとします。これらのフィールドは、以下のとおりです。

- 従業員番号: EMPNO
- 従業員名: EMPNAME
- 生年月日: BIRTHDAY
- 給与: SALARY
- 住所: ADDRESS

表 24. 従業員の物理セグメント

EMPNO	EMPNAME	BIRTHDAY	SALARY	ADDRESS

週ごとの従業員の小切手の郵送宛先ラベルを印刷するプログラムの場合、このセグメント内のデータがすべて必要というわけではありません。DBA がそのアプリケーションにフィールド・レベル・センシティビティーを使用する場合は、プログラムは必要なフィールドだけを入出力域で受け取るようになります。入出力域は、EMPNAME および ADDRESS フィールドを含むと考えられます。次の表では、プログラムの入出力域の内容を示しています。

表 25. フィールド・レベル・センシティビティーを使用した場合の従業員セグメント

EMPNAME	ADDRESS

フィールド・レベル・センシティビティーによって、アプリケーション・プログラムは、セグメントを構成するフィールドのサブセットを受け取ったり、または同一のフィールドを異なる順序で受け取ったり、あるいはその両方を行うことができます。

他に、データベースを新しく使用する際に、既存のセグメントに新しいデータ・フィールドを追加する場合にも、フィールド・レベル・センシティビティーを使用すると非常に便利です。この場合には、現行のセグメントを使用しているプログラムをコーディングし直すことを避けたいからです。 フィールド・レベル・センシティビティーを使用すると、以前のプログラムには、元のセグメント内にあったフィールドだけが見えます。新しいプログラムには、元のフィールドと新しいフィールドの両方が見えます。

フィールド・レベル・センシティビティーの指定

フィールド・レベル・センシティビティーは、アプリケーション・プログラムの PSB 内に指定します。指定は、アプリケーション・プログラムが重要と見なすフィールドごとに、センシティブ・フィールド (SENFLD) ステートメントを使用して行います。

関連資料:

SENFLD ステートメント (システム・ユーティリティー)

階層内での処理上の矛盾の解決: 副次索引

データベース階層が、それを処理するアプリケーション・プログラムの処理要件のすべてを満たしてはいない場合があります。

副次索引を使用して、以下の 2 種類の処理の矛盾を解決することができます。

- あるセグメントをアプリケーション・プログラムがリトリーブする際に、そのセグメントのキー・フィールドによって指定された順番とは異なる順番でリトリーブしたい場合
- あるセグメントをアプリケーション・プログラムがリトリーブする際に、そのセグメントの従属セグメント内に見られる状況に基づいてリトリーブしたい場合

上記の矛盾、ならびに副次索引がこの矛盾をどのように解決するかを理解するために、次の図の患者階層を処理する 2 つのアプリケーション・プログラムの例について考えてみましょう。この階層における 3 つのセグメントは、以下のとおりです。

- PATIENT セグメントには、患者の識別番号、名前、および住所の 3 つのフィールドが含まれます。 患者の識別番号フィールドがキー・フィールドになります。
- ILLNESS セグメントには、病気にかかった日付、および病名の 2 つのフィールドが含まれます。 病気にかかった日付がキー・フィールドになります。

- TREATMNT セグメントには、投薬の日付、薬の名前、薬の分量、および薬を処方した医師の名前の 4 つのフィールドが含まれます。 投薬の日付がキー・フィールドになります。

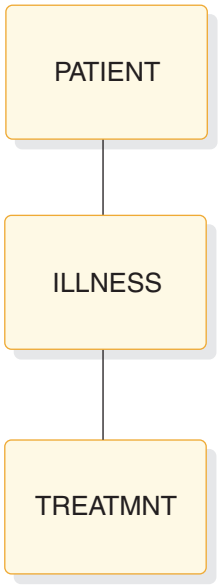


図 *16.* 患者階層

異なるキーに基づくセグメントのリトリーブ

アプリケーション・プログラムがデータベースからセグメントをリトリーブする場合には、セグメントのキー・フィールドによってセグメントを識別します。しかし、アプリケーション・プログラムがセグメントをリトリーブする際に、そのセグメントのキー・フィールドによって定義された順番とは異なる順番でリトリーブしなければならない場合もあります。副次索引を使用すると、それが可能になります。

注**:** 区分副次索引 (PSINDEX) という新規のデータベースのタイプが、High Availability Large Database (HALDB) によってサポートされます。 PSINDEX は、副次索引データベース・タイプの区分バージョンです。したがって、副次索引データベース・タイプの対応する説明が、PSINDEX に適用されます。

例えば、ある個人が病院に来たことがあるかどうかを確認する要求を処理するオンライン・アプリケーション・プログラムを使用しているとします。その患者が病院に来たことがあるかどうかが分からないと、その患者に識別番号を与えることができません。しかし、PATIENT セグメントのキー・フィールドは、患者の識別番号フィールドです。

あるセグメント・タイプのセグメント・オカレンス (例えば、患者ごとのセグメント) は、データベース内にキー・シーケンス (この場合、患者の識別番号順) に保管されます。 PATIENT セグメントについて要求を出し、患者の識別番号の代わりに名前でセグメントを識別する場合、要求した PATIENT セグメントを見つけるため

に、IMS は PATIENT セグメントのすべてを検索しなければなりません。 IMS は、患者の名前だけでは、特定の PATIENT セグメントがある場所を認識することはできません。

このアプリケーション・プログラムで、(患者の識別番号順ではなく）患者の名前順に PATIENT セグメントをリトリーブできるようにするには、PATIENT セグメントに患者名フィールドごとの索引を付け、その索引を別のデータベースに保管することができます。 その別のデータベースを 副次索引データベース と言います。

IMS に対して、患者階層データの PATIENT セグメントを副次索引データベース内の項目順に処理するように指示する場合、ユーザーが患者の名前を指定すると、IMS はその PATIENT セグメントを探し出すことができます。 IMS は、副次索引を直接読んで、ユーザーが指定した患者名によって、その PATIENT 索引項目を探し出します。 PATIENT 索引項目は患者名のアルファベット順になっています。索引項目は、患者階層データ内の PATIENT セグメントを指し示すポインターです。IMS は、ユーザーが指定した患者名の PATIENT セグメントが存在するかどうか判別し、存在する場合はそのセグメントをアプリケーション・プログラムに返すことができます。要求されたセグメントが存在しない場合は、IMS は、未検出状況コードを返して、そのセグメントが存在しないことをアプリケーション・プログラムに通知します。

関連資料: HALDB の詳細については、「*IMS V15* データベース管理」を参照してください。

副次索引で使用される用語には、次の 3 つがあります。

ポインター・セグメント
: 副次索引データベース内の索引項目です。IMS はこれを使用して、ユーザーが要求したセグメントを探し出します。上記の例では、ポインター・セグメントは、患者階層内の PATIENT セグメントを指す副次索引データベース内の索引項目です。

ソース・セグメント
: ユーザーが索引付けしているフィールドを含むセグメントです。上記の例では、ユーザーが PATIENT セグメント内の患者名フィールドで索引付けするため、ソース・セグメントは患者階層内の PATIENT セグメントです。

ターゲット・セグメント
: 処理しているデータベース内で、副次索引が指し示すセグメントです。これは、ユーザーがリトリーブしたいセグメントです。

上記の例では、ターゲット・セグメントとソース・セグメントは同一のセグメント、すなわち患者階層内の PATIENT セグメントです。ソース・セグメントとターゲット・セグメントが異なるセグメントの場合、副次索引によって処理上の矛盾が解決されます。

IMS がアプリケーション・プログラムの入出力域に返す PATIENT セグメントは、副次索引が使用されていない場合と見かけは同じになります。

キー・フィードバック域は異なります。 IMS が副次索引を使用しないでセグメントのリトリーブを行う場合、IMS はリトリーブされたセグメントの連結キーをキー・フィードバック域に入れます。連結キーには、そのセグメントの親のキーがす

べて、階層内での親の位置の順に含まれます。 最初にルート・セグメントのキー、次に階層内で 2 番目のレベルのセグメントのキー、次に 3 番目のレベルのセグメントのキーという順で、最後にリトリーブされたセグメントのキーが含まれます。

しかし、索引付きデータベースからセグメントをリトリーブする場合、要求後のキー・フィードバック域の内容は多少異なります。 DL/I は、キー・フィードバック域の左端のバイトにルート・セグメントのキーを入れずに、ポインター・セグメントのキーを入れます。『ポインター・セグメントのキー』とは、この場合のように、アプリケーション・プログラムが受け取るキーを指し示すことに注意してください。つまり、このキーにはサブシーケンス・フィールドは含まれません。

例えば、次の図に示す索引セグメント A が、セグメント C 内のフィールドに索引付けされているとします。セグメント A はターゲット・セグメント、セグメント C はソース・セグメントです。

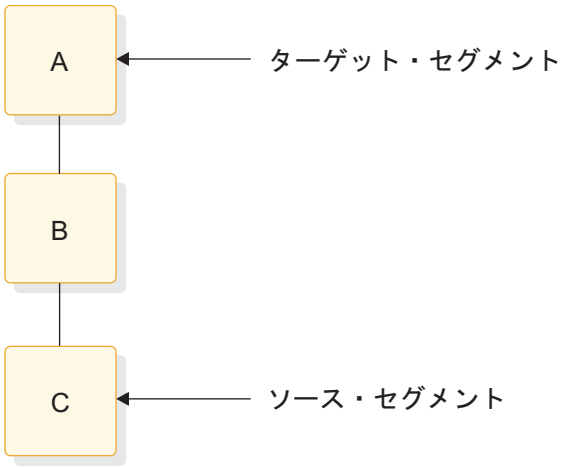


図 17. ルート・セグメントの索引付け

副次索引を使用して、この階層内のセグメントの 1 つをリトリーブする場合、キー・フィードバック域には以下のいずれかが含まれます。

- セグメント A をリトリーブする場合、キー・フィードバック域には、副次索引からのポインター・セグメントのキーが含まれます。
- セグメント B をリトリーブする場合、キー・フィードバック域には、ポインター・セグメントのキーにセグメント B のキーが連結されて含まれます。
- セグメント C をリトリーブする場合、キー・フィードバック域には、ポインター・セグメントのキー、セグメント B のキー、およびセグメント C のキーが連結されます。

この例では、ルート・セグメントについて副次索引を作成していますが、同様に従属セグメントを索引付けすることもできます。 その場合、反転構造を作成してください。反転構造では、索引付けするセグメントがルート・セグメントになり、親は従属になります。

例えば、セグメント B がセグメント C 内のフィールドに索引付けされているとします。この場合、セグメント B はターゲット・セグメント、セグメント C はソース・セグメントです。次の図は、物理データベース構造、および副次索引によって

作成される構造を示しています。

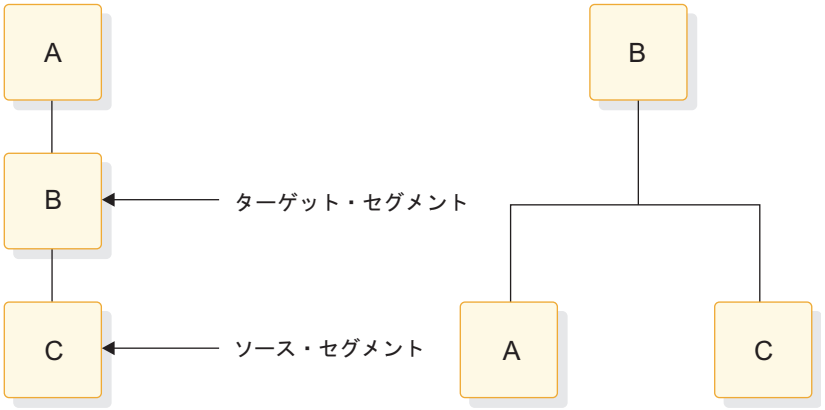


図 *18.* 索引、従属セグメント

右側の副次索引データ構造でセグメントをリトリーブする場合、IMS は次の情報をキー・フィードバック域に戻します。

- セグメント B をリトリーブする場合、キー・フィードバック域には、副次索引データベース内のポインター・セグメントのキーが含まれます。
- セグメント A をリトリーブする場合、キー・フィードバック域には、ポインター・セグメントのキーにセグメント A のキーが連結されて含まれます。
- セグメント C をリトリーブする場合、キー・フィードバック域には、ポインター・セグメントのキーにセグメント C のキーが連結されて含まれます。

従属セグメントの修飾に基づくセグメントのリトリーブ

アプリケーション・プログラムがセグメントのリトリーブを必要とすることがありますが、そのセグメントの従属のいずれかがある修飾に一致する場合にのみこれを行います。

例えば、病院で、その月に病院を訪れた患者の月次報告書を作成したいとします。この要求を処理するアプリケーション・プログラムが副次索引を使用しない場合、プログラムは PATIENT セグメントをそれぞれリトリーブし、それからそれぞれの PATIENT セグメントについて ILLNESS セグメントをリトリーブしなければなりません。プログラムは ILLNESS セグメント内の日付を調べて、ある患者がその月に病院を訪れたかどうか判別し、訪れた場合はその患者の名前を印刷します。 プログラムは、PATIENT セグメントをすべてリトリーブしてしまうまで、PATIENT セグメントと ILLNESS セグメントをリトリーブします。

しかし、副次索引を使用すると、プログラムの処理を単純化することができます。これを行うには、ILLNESS セグメント内の日付フィールドに基づいて PATIENT セグメントを索引付けします。 DBD 内に PATIENT セグメントを定義する際に、それに基づいて PATIENT セグメントを索引付けするフィールド名、および索引フィールドを含むセグメントの名前を、IMS に対して指定してください。 そうすると、アプリケーション・プログラムは、PATIENT セグメントを要求する際に、その要求を ILLNESS セグメント内の日付で修飾することができます。 アプリケーション・プログラムに返される PATIENT セグメントは、副次索引を使用しなかった場合とまったく同じように見えます。

この例では、PATIENT セグメントはターゲット・セグメント、すなわち、リトリーブしたいセグメントです。 ILLNESS セグメントは、ソース・セグメント、すなわち、PATIENT セグメントへの要求を限定するのに使用する情報を含むセグメントです。 2 次データベース内の索引セグメントは、ポインター・セグメントで PATIENT セグメントを指し示します。

新しい階層の作成: 論理関係

アプリケーション・プログラムが、異なる階層のセグメントを関連付ける必要がある場合、論理関係を使用するとそれが可能になります。

論理関係によって、次の矛盾を解決できます。

- 2 つのアプリケーション・プログラムが同じセグメントを処理する必要があるが、異なる階層を通じてそのセグメントにアクセスしなければならない場合
- あるアプリケーション・プログラムの階層ではセグメントの親であるものが、他のアプリケーション・プログラムではそのセグメントの子として機能する場合

異なるパスでのセグメントへのアクセス

アプリケーション・プログラムがデータを処理する際に、階層内に配列された順番とは異なる順番で処理しなければならない場合があります。

例えば、購買データベース内のデータを処理するアプリケーション・プログラムが、患者データベース内のセグメントにもアクセスしなければならない場合があります。

- プログラム A は、患者データベース内の病院の患者に関する情報を処理します。その情報には患者の病気、治療に関する情報が含まれます。
- プログラム B は、購買データベース内の、病院で使用する薬に関する情報を処理する在庫管理プログラムです。その情報には、薬の品目、業者、出荷情報、および、いつどのような状況で薬が投与されたかに関する情報が含まれます。

次の図は、プログラム A およびプログラム B がそれぞれの処理に必要とする階層を示しています。それぞれの処理要件は矛盾しています。双方共に患者データベース内の TREATMNT セグメントにアクセスする必要があります。 その情報とは以下のとおりです。

- 特定の薬が投与された日付
- 薬の名称
- 投薬の量
- 薬を処方した医師

プログラム B にとっては、この情報は、患者の治療に関するものではなく、薬の支出に関するものです。 購買データベースにとっては、これは支出セグメント (DISBURSE) になります。

次の図は、プログラム A およびプログラム B の階層を示しています。プログラム A には、PATIENT セグメント、ILLNESS セグメント、および TREATMNT セグメントが必要です。プログラム B には、ITEM セグメント、 VENDOR セグメント、SHIPMENT セグメント、および DISBURSE セグメントが必要です。

TREATMNT セグメントと DISBURSE セグメントには同じ情報が含まれます。

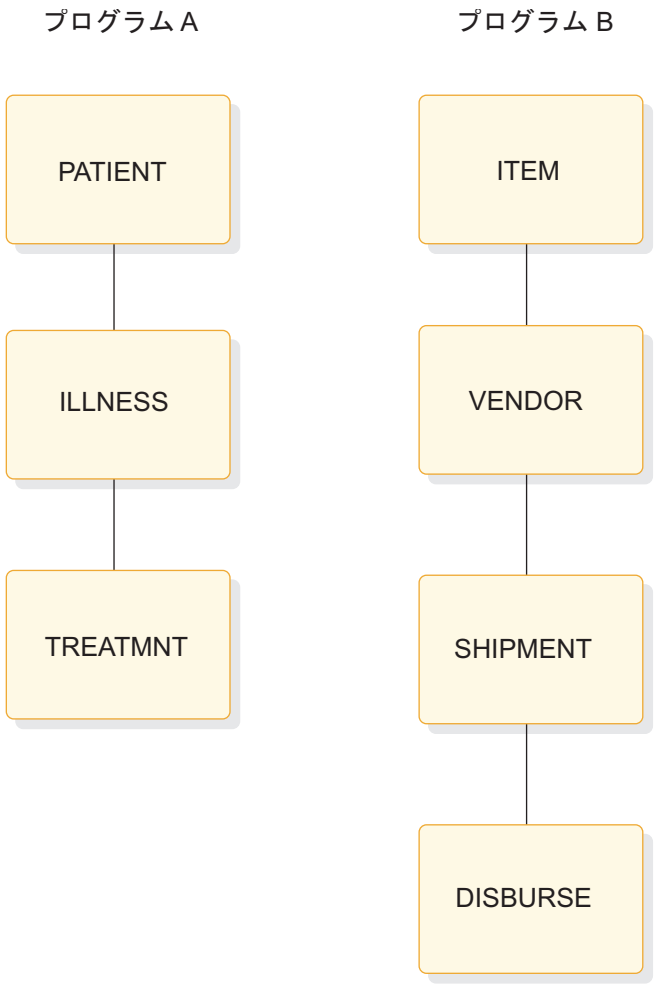


図 *19.* 患者階層と在庫階層

この情報を両方の階層内に保管せずに、論理関係を使用することができます。 論理関係を使用すると、そのセグメントが 1 つの階層で必要な地点から、そのセグメントが他の階層で存在する位置を指し示すポインターを保管することによって、この問題を解決できます。 この場合は、DISBURSE セグメント内に、医療データベースの TREATMNT セグメントを指し示すポインターを保管することができます。IMS は、購買データベースの DISBURSE セグメント内の情報を要求された場合、DISBURSE セグメントによって示された、医療データベース内の TREATMNT セグメントへ移動します。次の図は、プログラム A が処理する物理階層と、プログラム B が処理する論理階層を示しています。DISBURSE は、プログラム A の階層内の TREATMNT セグメントを指すポインター・セグメントです。

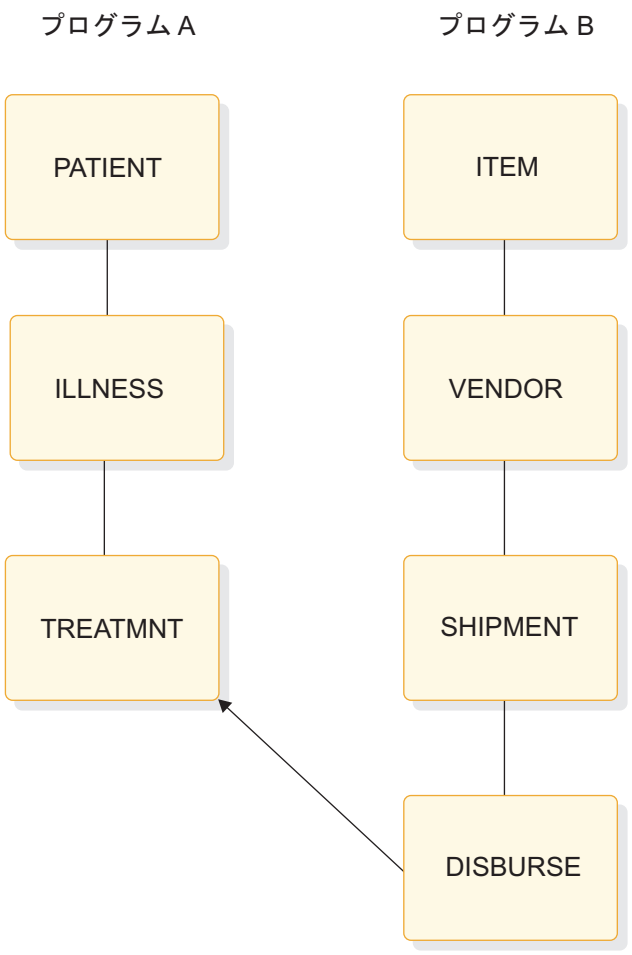


図 *20.* 論理関係の例

異なる階層内のセグメントの間の論理関係を定義するために、論理 DBD を使用します。論理 DBD は、ストレージには存在しないが、存在するものとして処理できる階層を定義します。 プログラム B は、前の図に示すような論理構造を、物理構造と同じように使用します。

親と子の関係の逆転

論理関係によって解決可能な別のタイプの矛盾としては、あるアプリケーション・プログラムではセグメントの親であるものが、他のアプリケーション・プログラムではそのセグメントの子として機能している、というものがあります。

- 在庫管理プログラム（プログラム B) は、薬をルート・セグメントとして使用して、薬に関する情報を処理する必要があります。
- 購買アプリケーション・プログラム（プログラム C) は、どの業者からどの薬を購入したかについての情報を処理します。 プログラム C は、業者をルート・セグメントとして使用して、この情報を処理しなければなりません。

次の図は、これらの各アプリケーション・プログラムの階層を示しています。

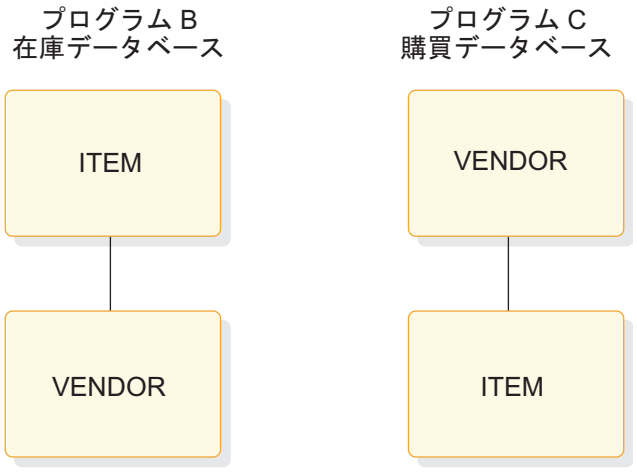


図 *21.* 在庫階層と購買階層

論理関係によって、ポインターを使用してこの問題を解決することができます。 この例でポインターを使用する場合は、購買データベースの ITEM セグメントに、在庫データベースの ITEM セグメント内の実データを指すポインターが含まれます。ただし、VENDOR セグメントは、実際には購買データベースに保管されます。在庫データベースの VENDOR セグメントは、購買データベース内に保管される VENDOR セグメントを指します。

次の図は、これらの 2 つのプログラムの階層を示しています。

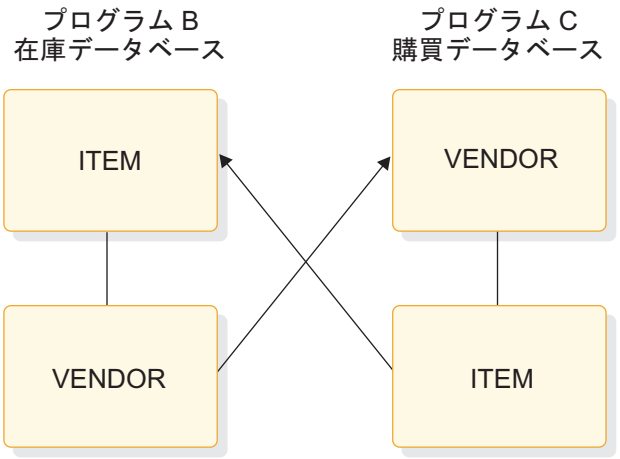


図 *22.* プログラム *B* およびプログラム *C* の階層

このような状況で論理関係を使用しないと、以下のような結果になります。

- 両方のパスに同じデータを保持するため、冗長データを保持することになります。
- 個別のファイルのデータと同様に、以下のような欠点があります。
  - データの一部が変更されるたびに、複数のセグメントの更新が必要になります。
  - より多くのストレージが必要になります。

データ・セキュリティーの提供

データ・センシティビティーおよび処理オプションを使用して、IMS アプリケーション・プログラムによってアクセスされるデータのセキュリティーを制御できます。

データ・センシティビティー
: 特定プログラムがアクセスできるデータを制御します。

処理オプション
: 特定プログラムがアクセスできるデータの処理方法を制御します。

データ可用性の提供

セグメント・センシティビティーと処理オプションを指定することは、データ可用性にも影響を与えます。 PCB が、SENSEGS の要求をできるだけ少なくし、可能な処理オプションを制限するように仕様を設定してください。 データベースの一部が使用不能であっても、データ可用性によって、プログラムは正常にデータベース内のセグメントへのアクセスと更新を続けることができます。

SENSEG ステートメントは、アプリケーション・プログラムがセンシティブと見なすセグメント・タイプをデータベース内に定義します。セグメントのタイプごとに、別々の SENSEG ステートメントを指定しなければなりません。セグメントは、あるデータベース内に物理的に存在する場合も、あるいは複数の物理データベースから派生する場合もあります。アプリケーション・プログラムが、ルート・セグメントより下のセグメントをセンシティブと見なす場合、そのプログラムは、ルート・セグメントからセンシティブ・セグメントまでの、パス内のすべてのセグメントを検知する必要があります。

関連資料 : フィールド・レベル・センシティビティーを使用したデータ・セキュリティー、および SENSEG ステートメントを使用して PCB の有効範囲を限定する方法の詳細については、「*IMS V15* データベース管理」を参照してください。

関連概念:

17 ページの『アプリケーション設計の概要』

プログラムによるデータ・アクセスの防止: データ・センシティビティー

IMS プログラムがアクセスできるデータは、そのデータに対してプログラムがセンシティブと見なすものだけです。

ユーザーは、プログラムがどのデータをセンシティブと見なすかを、 3 つのレベルで制御することができます。

- セグメント・センシティビティーを使用すると、アプリケーション・プログラムが、特定の階層内のすべてのセグメントにはアクセスできないようにすることができます。セグメント・センシティビティーは、プログラムが階層内のどのセグメントへのアクセスを許可されているかを、IMS に通知します。
- フィールド・レベル・センシティビティーを使用すると、プログラムが、特定のセグメントを構成するすべてのフィールドにはアクセスできないようにすること

ができます。フィールド・レベル・センシティビティーは、プログラムが特定のセグメント内のどのフィールドへのアクセスを許可されているかを IMS に通知します。

- キー・センシティビティーを使用すると、プログラムが、特定のセグメントに従属するセグメントにはアクセスできますが、特定のセグメントそのものにはアクセスできません。 IMS は、このタイプのセグメントのキーだけをプログラムに返します。

アプリケーション・プログラムの PSB に、これらのセンシティビティーのレベルをそれぞれ定義してください。キー・センシティビティーは、セグメントの処理オプション内に定義します。処理オプションは、特定のプログラムがデータに行ってよいことまたは行ってならないことを、 IMS に対して厳密に指示します。処理オプションは、アプリケーション・プログラムが処理する階層ごとに指定してください。指定は各階層を表す DB PCB 内に行います。階層内のすべてのセグメントに同一の処理オプションを指定することも、あるいは階層内の各セグメントに異なる処理オプションを指定することもできます。

セグメント・センシティビティーおよびフィールド・レベル・センシティビティーは、 PSB 内で特別なステートメントを使用して定義します。

セグメント・センシティビティー

アプリケーション・プログラムがどのセグメントを重要と見なすかを、それらのセグメントを含む階層の DB PCB に定義してください。

例えば、次の図に示すような患者階層を想定します。 患者階層は、医療データベースのサブセットといえます。

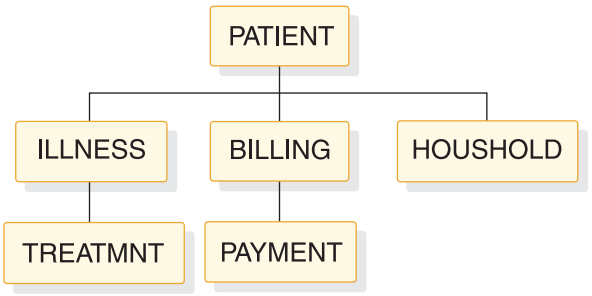


図 23. 医療データベース階層

PATIENT はルート・セグメントで、その下にある、ILLNESS、BILLING、および HOUSHOLD の 3 つのセグメントの親です。ILLNESS の下は、TREATMNT です。BILLING の下は、PAYMENT です。

アプリケーション・プログラムが、医療データベースから PATIENT、ILLNESS、および TREATMNT のセグメントだけを表示できるようにするときは、ユーザーが定義する階層にこれらの 3 つのセグメントのタイプが含まれること、およびそれらのセグメントは医療データベースのものであることを、 DB PCB に指定してください。データベース階層は DBD に定義し、アプリケーション・プログラムがデータベース階層のどの部分を表示するかは DB PCB に定義します。

フィールド・レベル・センシティビティー

フィールド・レベル・センシティビティーは、アプリケーション・プログラムのためにデータ独立性を保つだけではなく、プログラムが使用するデータのセキュリティー機構としても機能します。

プログラムが、セグメント内のいくつかのフィールドにアクセスする必要があるが、プログラムがアクセスする必要のない 1 つまたは 2 つのフィールドが機密である場合、フィールド・レベル・センシティビティーを使用することができます。アプリケーション・プログラムにとって、そのセグメントは機密ではないフィールドだけを含むように定義すれば、プログラムが機密フィールドにアクセスするのを防ぐことができます。 フィールド・レベル・センシティビティーは、アクセスを制限したいフィールドのマスクとして機能します。

キー・センシティビティー

あるセグメントにアクセスするためには、アプリケーション・プログラムは、そのセグメントのパスより高いレベルにあるセグメントすべてを、センシティブと見なす必要があります。つまり、次の図において、セグメント C にアクセスするためには、プログラムはセグメント B をセンシティブと見なす必要があります。

例えば、アプリケーション・プログラムが処理を行うために、セグメント C が必要であるとします。しかし、セグメント B に機密情報 (従業員の給与など) が含まれる場合、プログラムはセグメント C にアクセスすることはできません。キー・センシティビティーを使用すれば、アプリケーション・プログラムがセグメント B の従属セグメントをアクセスできるようにしながら、一方ではこのプログラムがセグメント B にアクセスするのを防ぐことができます。

センシティブ・セグメント・ステートメントに K の処理オプションが指定されている場合、プログラムはそのセグメントにアクセスすることはできませんが、そのセグメントを通過して、そのセグメントの従属セグメントにアクセスすることができます。プログラムがセグメントの従属セグメントにアクセスする場合、IMS が返すのは、そのセグメントではなく、そのセグメントのキーと、アクセスしたその他のセグメントのキーだけです。

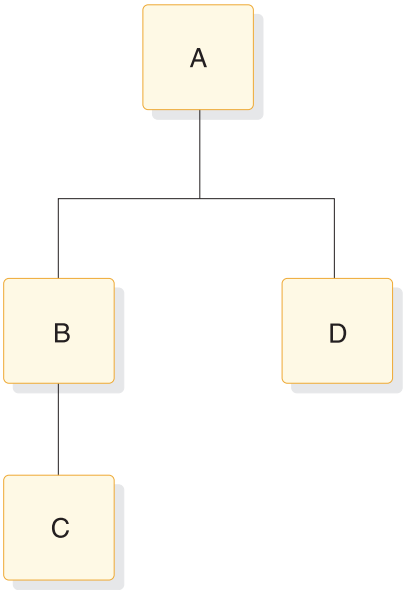


図 *24.* キー・センシティビティーの階層例

プログラムによるデータ更新の防止: 処理オプション

PCB 生成の間、PROCOPT パラメーター (DATABASE マクロ内) の 5 つのオプションを使用してプログラムが階層内のセグメントを読み取ることができるか、あるいはセグメントを更新することもできるかを、IMS に対して指示することができます。

処理オプションは、制限の大きいものから順に、以下のように分かれています。

G プログラムは、セグメントを読むことができます。

R プログラムは、セグメントの読み取りおよび置換ができます。

I プログラムは、セグメントを挿入することができます。

D プログラムは、セグメントの読み取りおよび削除ができます。

A プログラムはすべての処理オプションを実行できます。 これは、G、R、I および D をすべて指定したのと同じことになります。

関連資料: 処理オプションの完全な説明については、「*IMS V15* システム・ユーティリティー」を参照してください。

処理オプションは、ある特定のセグメントまたは階層に対して、プログラムが行える処理を制限するため、データ・セキュリティーを提供します。プログラムに必要な処理オプションだけを指定することによって、プログラムが必要のないデータ更新を行うことがなくなります。 例えば、プログラムがデータベースからセグメントを削除する必要がない場合、D オプションを指定する必要はありません。

前述の処理オプションのいずれかが指定されたアプリケーション・プログラムがセグメントをリトリーブすると、IMS はそのアプリケーション・プログラムに対しデータベース・レコードをロックします。 PROCOPT=G が指定されている場合、オ

プションが指定されている他のプログラムは、データベース・レコードを同時にアクセスすることができます。更新処理オプション (R、I、D、または A) が指定されている場合は、他のプログラムは同じデータベース・レコードを同時にアクセスすることはできません。更新が行われない場合、アプリケーション・プログラムが他のデータベース・レコード (または HDAM の場合は、他のアンカー・ポイント) に移動する際に、ロックが解除されます。

このロック・プロトコルにより、IMS は次のような決定を行うことができます。ルート・セグメントが更新されると、ルート・ロックは、コミットまで更新レベルで保持されます。ある従属セグメントが更新されると、その従属セグメントは更新レベルでロック状態になります。データベース・レコードが存在している場合は、ルート・セグメントは、読み取りレベルにデモートします。プログラムがデータベース・レコードに入り、読み取りレベルもしくは更新レベルのいずれかでロックを取得すると、ロック・マネージャーはフィードバックを返し、別のプログラムが読み取りレベルでロック取得しているかどうか示します。これにより、従属セグメントがアクセスされた場合に、これがロックされるかどうかが決定されます。HISAM の場合、主要な論理レコードはルート・セグメントとして、オーバーフローした論理レコードは、従属セグメントとして扱われます。

オンライン・プログラムおよびバッチ・プログラムで、ブロック・レベルあるいはデータベース・レベルでデータ共用を行う場合、追加の処理オプションを使用することができます。

関連資料:

- パラメーター ERASE=YES 指定の HISAM 削除バイト関連の特殊ケースについては、「*IMS V15* データベース管理」を参照してください。
- データベースおよびブロック・レベルのデータ共用の詳細については、「*IMS V15* システム管理」を参照してください。

E オプション

E オプションを使用すると、プログラムが使用する階層またはセグメントに対して排他的アクセスを行うことができます。 E オプションは、オプション G、I、D、R、および A と併用されます。E プログラムが稼働中は、その他のプログラムはそのデータにアクセスできませんが、E プログラムの PCB に指定されていないセグメントにはアクセスできる場合があります。プログラム分離による動的エンキューは行われませんが、データベース更新の動的ロギングは行われます。

GO オプション

GO オプションが指定されたプログラムがセグメントをリトリーブする場合、IMS はそのセグメントをロックしません。保全性のない読み取りプログラムがセグメントを読み取っている間は、他のプログラムに対してそのデータは使用可能なままになります。これは、ユーザーのプログラムはデータを読み取ることしかできず (読み取り専用)、データベースを更新することが許可されていないためです。このデータベースに対する呼び出しのために、プログラム分離による動的エンキューは行われません。 PROCOPT=GO を指定したプログラムと他の更新プログラムとの間のシリアライゼーションは起こらず、同一データに対する更新が同時に起こります。

セグメントが削除され、同じタイプの別のセグメントが同じロケーションに挿入されると、このセグメント・データおよびアプリケーションに返される後続のすべてのデータは、異なるデータベース・レコードからのものになる可能性があります。

保全性のない読み取りプログラムは、別のプログラムがセグメントを更新中の場合であっても、そのセグメントをリトリーブすることもできます。このことは、他のプログラムがアクセス中のセグメントに対して、プログラムが待機する必要がないことを意味します。 保全性のない読み取りプログラムが、別のプログラムによって更新中のデータを読み取り、次のコミット・ポイントに達する前に異常終了した場合、そのプログラムが更新したセグメントには無効なポインターが含まれることがあります。無効ポインターが検出されると、GO に N または T オプションが指定されていなければ、保全性のない読み取りプログラムは異常終了します。ポインターは挿入、削除、およびバックアウトの機能実行時に更新されます。

N オプション

GO オプションとともに N オプションを使用して全機能データベースまたは DEDB をアクセスしていて、リトリーブするセグメントが無効なポインターを含んでいる場合、IMS はプログラムに GG 状況コードを返します。 プログラムはその後で処理を終了すること、別のセグメントを読み取って処理を続行すること、あるいは異なるパスを使用してデータをアクセスすることができます。 N オプションは、PROCOPT=GON、GON、または GONP として指定しなければなりません。

T オプション

GO オプションとともに T オプションを使用していて、リトリーブするセグメントが無効なポインターを含んでいる場合、アプリケーション・プログラムからの応答は、プログラムが全機能データベースをアクセス中か、高速機能データベースをアクセス中かによって異なります。

全機能データベースに対する呼び出しの場合、T オプションによって、 DL/I は自動的に操作を再試行します。ユーザーは更新されたセグメントをリトリーブすることができますが、それは更新を行っているプログラムがコミット・ポイントに到達した場合、またはそのプログラムが、ユーザーが最後にセグメントのリトリーブを試みた後、その更新をバックアウトしていた場合だけです。再試行が失敗すると、GG 状況コードがプログラムに返されます。

高速機能 DEDB への呼び出しの場合には、オプション T によって、 DL/I が操作を再試行することはありません。 GG 状況コードが返されます。 T オプションは、PROCOPT=GOT、GOT または GOTP として指定しなければなりません。

GOx およびデータ保全性

非常に小規模なアプリケーションおよびデータの集まりでは、 PROCOPT=GOx には、パフォーマンスと並列処理について利点があります。しかし、アプリケーション・データ保全性は保たれません。例えば、全機能データベースにオンライン環境で PROCOPT=GOT を使用すると、パフォーマンスの低下を引き起こす場合があります。 T オプションは DASD からの再読み取りを強制し、稼働中のすべてのアプリケーションと共用データ用の、大規模バッファー・プールおよび VSAM ハイパースペースの利点を損なうことになります。 DEDB の GOx 処理オプションについて詳しくは、「*IMS V15* システム・ユーティリティー」を参照してください。

関連概念:

『保全性なしでの読み取り』

保全性なしでの読み取り

IMS データベースのデータベース・レベル共用は、更新能力のある単一バッチ・システムまたはオンライン IMS システムと、保全性なしでデータを読み取っている他の多くの IMS システムとの間で、データベースを共用するための機能をもっています。

制御インターバル (CI) 分割時に HIDAM データベースに存在するセグメントについて PROCOPT=GOx を使用するプログラムに、GE 状況コードが戻される場合があります。

IMS では、データベース・レベル共用を使用するプログラムには、そのデータ用の DBPCB の中に、PROCOPT=GOx が含まれています。バッチ・ジョブの場合、DBPCB の PROCOPT はそのデータベースについて、バッチ・ジョブのアクセス・レベルを設定します。つまり、バッチ・ジョブは、DBRC データベース許可のためのアクセス・レベルとして、データベースに対して最も高く宣言されたインテントを使用します。オンライン IMS 環境では、データベース ACCESS は、 IMS システム定義中に DATABASE マクロで指定され、 `/START DB ACCESS=RO` コマンドを使用して、それを変更することができます。オンライン IMS システムは、スケジュールされるプログラムの PSB の中で、 PROCOPT によって決まるデータ可用性を備えた、プログラムをスケジュールします。そのため、そのデータ可用性は、オンライン・システムのデータベース・アクセスによって限定されます。

PROCOPT=GON および GOT オプションには、保全性なしで読み取り中のデータの中に認識可能なポインター・エラーがある場合に、ある特定の PCB 状況コードの再試行を行う機能があります。保全性なしでの読み取り IMS インスタンスに対して、非同期的に発生する従属セグメントの更新では、状況によっては、保全性なしでデータを読み取っているプログラムと干渉しないこともあります。ただし、平均的なデータベースに対する更新アクティビティーでは、必ずしも、保全性なしでの読み取り IMS システムが、データの問題を認識できるとは限りません。

保全性なしでの読み取りの意義

各 IMS バッチまたはオンライン・インスタンスには、専用に定義された OSAM および VSAM バッファー・プールがあります。他の IMS インスタンスで起こっている並行更新を、ロックして順次化しなくても、データベース・データ・セットからの保全性なしでの読み取りは、ブロックまたは CI のコピーをストレージ内のバッファー・プールに取り込みます。バッファー・プール内のブロックまたは CI は長時間そこに残ります。 他のブロックまたは CI の保全性なしでのそれ以降の読み取りでは、より新しいデータを取り込みます。データの階層、および異なるブロックまたは CI 間における他のデータの関連性で、不整合が起こることがあります。

例えば、索引コンポーネントとデータ・コンポーネントをもつ索引データベース (VSAM KSDS) について考えます。索引コンポーネントには、データ・コンポーネント CI に関連する階層制御情報だけが含まれており、そのデータ・コンポーネント CI に指定のキー・レコードがあります。これを、索引コンポーネント CI が、各データ・コンポーネント CI 内でハイ・キーを保持するための方法として考えま

す。 すでにいっぱいの KSDS データ・コンポーネント CI にキー・レコードを挿入すると、CI が分割されます。つまり、既存の CI 内のレコードの一部には、新規の CI に移動されているものがあり、索引コンポーネントは、その新規の CI を指すように調整されます。

例えば、索引 CI が、最初のデータ CI 内のハイ・キーを KEY100 として示していて、分割が起こったとします。分割により KEY051 から KEY100 までが新規の CI に移動されるとします。これで索引 CI は、最初のデータ CI 内のハイ・キーを KEY050 として示し、別の項目が、新規の CI 内に、ハイ・キーを KEY100 として示します。

保全性なしで読み取り中のプログラムで、『古い』索引コンポーネント CI をそのバッファー・プール (ハイ・キー KEY100) に既に読み取っているものは、新しく作成したデータ CI を指すことはなく、それにアクセスしようともしません。さらに、保全性なしでの読み取りプログラムが開始される時点で KSDS に存在したキー・レコードが、見当たらないことがあります。この例では、バッファー・プール内の索引 CI の『古い』コピーが、 KEY100 までの既存のキーが最初のデータ CI にあると、依然として示していても、 KEY051 から KEY100 までは、もはや最初のデータ CI にはありません。

従属セグメントが削除され、削除されるセグメントと物理的に同じロケーションに、異なるルートの下の同じタイプのセグメントが追加された場合、単純な Get Next 処理では、データベース内にただ 1 つのルートしかないように見えてしまうという仮設的ケースも成立します。例えば、データベース内の最初のルートのもとにあるセグメントをレベル 06 セグメントに向かってアクセスすると (そのセグメントは、最初のルートから削除されていて、現在は物理的に最後のルートのもとにある場合)、データが他のルートから再現されることになります。それ以降の Get Next 呼び出しで、セグメントが他のルートからリトリーブされます。

読み取り専用 (PROCOPT=GO) 処理では、データ保全性の機能はありません。

データ・セット拡張

データベース・レベル共用を指定した IMS インスタンスは、保全性なしでの読み取り用にデータベースをオープンすることができます。

データベースがオープンされた後、そのデータベースを更新している別のプログラムがデータに対する変更を行うことができます。これらの変更は、データベース・データ・セットに対する、論理的および物理的拡張部分になります。保全性なしでの読み取りプログラムは、これらの拡張部分を認識しないため、 RBA (データ終了の飛び越え) の問題が起こることがあります。

関連概念:

110 ページの『プログラムによるデータ更新の防止: 処理オプション』

第 6 章 メッセージ処理オプションの要件収集

アプリケーション設計の作業の 1 つは、IMS システムの設計および管理に携わっている人に、アプリケーションの要件についての情報を提供することです。

制約事項: この情報は、DB/DC 環境および DCCTL 環境のみに適用されます。

関連概念:

826 ページの『IMS Java 従属領域リソース・アダプターを使用したプログラミング』

オンラインのセキュリティー要件の識別

オンライン・システムのセキュリティーは、端末を介して、データが未許可で使用されないようにすることを意味します。また、IMS システムおよびデータベースをアクセスするアプリケーション・プログラムの双方を未許可で使用しないようにすることも意味します。例えば、システムにアクセスする人すべてが、給料支払小切手を処理するプログラムを利用できるようにする必要はありません。

IMS が提供するセキュリティー機構は、サインオン、端末、パスワードのセキュリティーです。

関連資料: これらのタイプのセキュリティーをどのように設定するかについては、「*IMS V15* システム管理」を参照してください。

特定の人へのアクセス制限：サインオン・セキュリティー

サインオン・セキュリティーは、RACF® (リソース・アクセス管理機能) またはユーザー作成のセキュリティー出口ルーチンを介して利用できます。サインオン・セキュリティーを使用すると、IMS を使用する人は、RACF またはそれと同等のものに定義された後でアクセスを許可されます。

ある人が IMS にサインオンすると、IMS 制御リソースへのアクセスを許可される前に、RACF またはセキュリティー出口により、IMS の使用許可があるかどうか検査されます。このサインオン・セキュリティーを使用するには、 `/SIGN ON` コマンドを出します。入力を許可された人が使用できるトランザクション・コードおよびコマンドも、制限することができます。 この制限は、トランザクション・コードおよびコマンドと許可された人のユーザー識別 (USERID) を関連付けることによって行います。

LU 6.2 トランザクションには、USERID が含まれます。

関連資料: セキュリティーについて詳しくは、「*IMS V15* コミュニケーションおよびコネクション」を参照してください。

特定の端末へのアクセス制限： 端末セキュリティー

端末セキュリティーを使用して、トランザクション・コードの入力をシステム内の特定の端末または端末グループだけに制限してください。 どのようにしてこれを行うかは、保護しなければならないプログラムの数によって異なります。

特定のプログラムを保護するためには、トランザクション・コードをリストにした論理端末から入力されるものとして許可するか、または、ユーザーがその論理端末から入力できるトランザクション・コードのリストと各論理端末を関連付けることができます。例えば、給与支払小切手アプリケーション・プログラムと関連するトランザクション・コードを、給与計算部門の端末から入力したときだけ有効であるように定義することで、そのアプリケーション・プログラムの保護を行えます。このアプリケーションへのアクセスをさらに制限したい場合には、給与計算トランザクション・コードを、1 つの論理端末とだけ関連付けてください。そのトランザクション・コードを入力するためには、ユーザーは、その論理端末と関連付けられている物理端末で作業する必要があります。

制約事項: 共用キュー・オプションを使用している場合は、セキュリティー検査に必要なリソースを表す静的制御ブロックが、セキュリティー検査が行われる IMS システムの中で使用可能である必要があります。それ以外の場合は、セキュリティー検査がバイパスされます。

関連資料: 共用キューについて詳しくは、「*IMS V15* システム管理」を参照してください。

プログラムへのアクセス制限: パスワード・セキュリティー

アプリケーション・プログラムを保護する別の方法として、保護したいアプリケーション・プログラムと関連のあるトランザクション・コードを入力するときに、パスワードを要求する方法があります。パスワード・セキュリティーだけを使用する場合には、特定のトランザクション・コードを入力する人が、 IMS がトランザクションを処理する前に、そのトランザクションのパスワードも入力しなければなりません。

端末セキュリティーとともにパスワード・セキュリティーを使用している場合には、より一層プログラムへのアクセスを制限することができます。給与支払小切手を例にあげると、パスワード・セキュリティーと端末セキュリティーを使用した場合には、給与計算部門でも許可されていない人がプログラムを実行しないように制限することができるようになります。

制約事項: トランザクションのパスワード・セキュリティーがサポートされるのは、セキュリティー検査の必要なトランザクションが、セキュリティー検査が行われる IMS システムに定義されている場合のみです。それ以外の場合は、セキュリティー検査がバイパスされます。

セキュリティー・データへのアクセス許可: 許可セキュリティー

RACF には、ユーザー固有の情報を保管するために使用できるデータ・セットがあります。 AUTH 呼び出しは、アプリケーションが定義したリソースへのアクセスを制御する方法と同じように、RACF データ・セット・セキュリティー・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。したがって、アプリ

ケーション・プログラムは、特定のユーザーについてのセキュリティー情報を獲得することができます。

IMS セキュリティーを Db2 for z/OS セキュリティーと関連付ける方法

Db2 for z/OS セキュリティーの重要な部分に、許可 ID があります。 IMS がプログラムまたは端末のユーザーに対して使用する許可 ID は、使用するセキュリティーの種類、実行するプログラムの種類に応じて異なります。

MPP、IFP、およびトランザクション指向の BMP の場合には、許可 ID は IMS セキュリティーの種類に応じて異なります。

- サインオンが必要な場合、IMS は Db2 for z/OS にサインオン済みの USERID およびグループ名を渡します。
- サインオンが必要ない場合は、Db2 for z/OS は起点論理端末の名前を許可 ID として使用します。

バッチ指向 BMP の場合の許可 ID は、PROCLIB メンバー DFSDCxxx の中の BMPUSID= キーワードに指定されている値によって、次のように異なります。

- BMPUSID=USERID が指定されている場合、JOB ステートメントの USER= キーワードの値が使用されます。
- JOB ステートメントで USER= が指定されていない場合は、プログラムの PSB 名が使用されます。
- BMPUSID=PSBNAME が指定されている場合、または BMPUSID= がまったく指定されていない場合は、プログラムの PSB 名が使用されます。 PSBNAME が RACF に対して定義されていない場合は、現行アドレス・スペースのユーザー ID が使用されます。これは、LSO=Y または PARDLI=1 が BMP に対して指定されている場合、ホーム従属領域のものか、制御領域のものになります。 DFSBSEX0 が RC08 を返した場合も、現行アドレス・スペースのユーザー ID が使用されます。

セキュリティー情報の提供

セキュリティー要件に関してアプリケーションを評価するときには、各プログラムを個別に調べる必要があります。これを行うと、セキュリティー担当者に次の情報を提供できます。

- サインオン・セキュリティーが必要なプログラムの場合
  - IMS にアクセスできるようにする必要がある人をリストする。
- 端末セキュリティーが必要なプログラムの場合
  - 保護されなければならないトランザクション・コードをリストする。
  - これらの各トランザクション・コードを、入力できるようにしなければならない端末をリストする。リストしている端末が既にインストール済みで使用されている場合には、論理端末名で端末を識別します。そうでない場合には、その端末を使用する部門で識別してください (例えば、会計部門など)。
- パスワード・セキュリティーが必要なプログラムの場合
  - パスワードが必要なトランザクション・コードをリストする。
- セキュリティーが必要なコマンドの場合

– サインオンまたはパスワード・セキュリティーが必要なコマンドをリストする。

関連概念:

17 ページの『アプリケーション設計の概要』

画面およびメッセージ形式の分析

アプリケーション・プログラムが端末と通信する場合には、編集プロシージャーによって、メッセージが端末から入力された形式から、プログラムがメッセージを受信し処理する形式に変換されます。

IMS がどのようにしてプログラムのメッセージを編集するかという判断は、データが端末やアプリケーション・プログラムにどのように表示されるかに基づきます。プログラムからのデータを端末画面にどのように表示したいか、ならびに端末からのデータをアプリケーション・プログラムの入出力域にどのように読み込みたいかを説明する必要があります。 (入出力域には、アプリケーション・プログラムによって処理されるセグメントが含まれます。)

この判断に役立つ情報を提供するためには、IMS がどのようにしてメッセージを編集するかということに精通していなければなりません。 IMS には、以下の 2 つの編集用プロシージャーがあります。

- メッセージ形式サービス **(MFS)** は、端末およびアプリケーション・プログラムに、メッセージがどのように表示されるかを定義する制御ブロックを使用します。
- 基本編集は、すべての IMS アプリケーション・プログラムが使用することができます。基本編集は、入力メッセージから制御文字を除去し、指定した制御文字を端末への出力メッセージに挿入します。

関連資料**:** IMS 編集プロシージャーの定義および IMS ネットワークに関するその他の設計考慮事項については、「*IMS V15* コミュニケーションおよびコネクション」を参照してください。

MFS の概要

MFS は、4 種類の制御ブロックを使用して、アプリケーション・プログラムと端末間のメッセージを形式設定します。 データがアプリケーション・プログラムと端末間で渡されるときにどのように形式設定される必要があるかについて収集した情報は、この 4 つの制御ブロックに入れられます。

IMS への入力メッセージを説明する制御ブロックが 2 つあります。

- 装置入力形式 (DIF) は、メッセージが端末で入力されると、その入力メッセージがどのように表示されていたかを IMS に示します。
- メッセージ入力記述子 (MID) は、アプリケーション・プログラムがその入出力域内の入力メッセージを、どのように受け取るかを IMS に示します。

DIF および MID を使用して、IMS は入力メッセージが端末で入力される形式から、プログラムの入出力域に読み込まれるような形式に変換することができます。

IMS への出力メッセージを説明する制御ブロックが 2 つあります。

- メッセージ出力記述子 (MOD) は、出力メッセージがプログラムの入出力域にどのように表示されるかを IMS に示します。
- 装置出力形式 (DOF) は、メッセージが端末にどのように表示されるかを IMS に示します。

アプリケーション・プログラムの MFS 制御ブロックを定義するためには、端末およびアプリケーション・プログラムの入出力域に、入出力両方のデータをどのように表示したいかを理解しておく必要があります。

基本編集の概要

基本編集を行うと、アプリケーション・プログラムが受け取る前に入力メッセージから制御文字が除去され、アプリケーション・プログラムが端末にメッセージを戻すときに、指定した制御文字が挿入されます。

基本編集を使用して端末で出力メッセージを形式設定するためには、使用する端末に必要な制御文字を提供する必要があります。

アプリケーションが基本編集を使用する場合には、データを端末にどのように表示したいか、ならびにプログラムの入出力域にどのように表示したいかを説明する必要があります。

アプリケーションにおける編集の考慮事項

アプリケーションの編集要件を記述する前に、画面設計に関する標準を知っている必要があります。要件がこれらの標準に合っているかを確かめてください。

プログラムの編集要件に関する次の情報を準備してください。

- 入力データを入力する人の代わりに端末で作業している人に、どのように画面を表示させたいか。例えば、航空代理店が特定のフライトでの座席を予約する必要がある場合、この情報を要求する画面は、以下のように表示されます。

  FLIGHT#:
  NAME:
  NO. IN PARTY:
- 端末で入力メッセージを入力するとき、そのデータがどのように表示されるか。
- 入力メッセージはプログラムの入出力域にどのように表示されるか。
- プログラムがその入出力域で出力メッセージを作成するときに、データがどのように表示されるか。
- 出力メッセージは端末でどのように形式設定されるか。
- プログラムまたは端末が交換するデータの長さおよびタイプ

端末でデータをどのように表示したいか分析する場合、処理しているデータのタイプだけが考慮事項になります。さらに、アプリケーション・プログラムの効率 (アプリケーション・プログラムのパフォーマンス要因) を目的とした画面設計と対比して、端末で作業する人の必要性 (アプリケーションのヒューマン・ファクター) を比較考慮してください。しかし、ヒューマン・ファクターとパフォーマンス要因とが両立しない場合も起こり得ます。端末で作業する人にとって容易に理解し使用できる画面設計が、アプリケーション・プログラムを最も効率の良い状態にする設計であるとは限りません。確立された画面標準に従うことが最も大切です。

ヒューマン・ファクターを考慮して設計された端末画面とは、端末で作業する人を第一に考えたものです。この場合の端末画面設計は、IMS との対話をできるだけ容易にさせようとするものです。端末で作業する人が理解したり応答したりしやすくするために、アプリケーション・プログラムに対して以下のことを行うことができます。

- 一度に少量のデータを表示する。
- 明確で整とんされた形式を使用する。
- 明確で単純な指示を出す。
- 一度に 1 つのアイデアを表示する。
- 端末で作業する人からの短い応答を要求する。
- 端末で作業する人にヘルプの機能と訂正しやすい手段を提供する。

同時に、アプリケーション・プログラムの応答時間またはシステムのパフォーマンスに悪い影響を与えるような画面の設計は望ましくありません。パフォーマンス目的優先で画面を設計するときには、 IMS が各メッセージに実行しなければならない処理を減らす必要があります。このことを行うために、IMS が、さらにメッセージを処理しなくてもよいように、端末で作業する人が 1 つの画面で大量のデータを、アプリケーション・プログラムに送信できるようにします。つまり、プログラムは、端末で作業している人に対して、 1 つの画面で表示できる情報を 2 つの画面で表示する必要はありません。

プログラムが端末からどのようにしてデータを受け取るかを説明するときには、プログラム論理と作業するデータ・タイプについて考慮する必要があります。

会話型処理の要件収集

会話型処理 を使用する場合には、端末で作業する人が何らかの情報を入力し、アプリケーション・プログラムがその情報を処理し、端末に応答します。それから、端末で作業する人は、アプリケーション・プログラムに処理させる情報をさらに入力します。端末で作業する人とプログラムとの各対話は、会話における 1 ステップと呼ばれます。 MPP だけが会話型プログラムになります。高速機能プログラムや BMP は、会話型プログラムではありません。

定義 : 会話型処理とは、端末で作業する人が、アプリケーション・プログラムと対話できるという意味です。

会話中の処理

会話 とは、スクラッチパッド域 (SPA) および 1 つ以上のアプリケーション・プログラムを介して、端末で作業するユーザーと IMS とが行うダイアログと定義されます。

会話中に、端末で作業するユーザーは、要求を入力し、IMS からの情報を受信し、別の要求を入力します。ユーザーにとっては明確なものではありませんが、会話は、複数または 1 つのアプリケーション・プログラムによって処理されます。

会話を継続するためには、そのプログラムは処理を続けるために必要な情報を持っていなければなりません。 IMS は、会話のあるステップから次のステップの間

に、 SPA にデータを保管します。 同じプログラムまたは異なったプログラムが会話を継続する場合、IMSは、プログラムにその端末と関連した会話用の SPA を与えます。

前述の航空代理店の例では、最初のプログラムがフライト番号と乗客の名前を保管し、制御を次のアプリケーション・プログラムに渡し、そのフライトでのこれらの乗客の座席予約を行います。最初のプログラムは、SPA 内にこの情報を保管します。もし 2 番目のアプリケーション・プログラムが、フライト番号と乗客の名前に関する情報を持っていなかったら、この処理を行うことはできません。

会話の設計

会話を設計する際に最初に行うことは、会話のフローを設計することです。端末で作業する人からの要求を 1 つのアプリケーション・プログラムだけが処理する場合には、そのプログラムだけを設計する必要があります。複数のアプリケーション・プログラムで会話が処理される場合には、それぞれのプログラムが処理する会話ステップとその会話ステップで処理を終えたときに、各プログラムがすべきことを決定する必要があります。

端末で作業する人が、会話型と定義されたトランザクション・コードを入力すると、IMS は、そのトランザクション・コードと関連付けられている会話型プログラム (例えば、プログラム A) をスケジュールします。プログラム A がメッセージ・キューに最初の呼び出しを出すときには、 IMS は、そのトランザクション・コード用に定義されている SPA を、プログラム A の入出力域に戻します。端末で作業する人は、最初の入力画面にトランザクション・コードのみ (もしあれば、パスワードも) を入力しなければなりません。会話の各ステップで、トランザクション・コードを入力する必要はありません。 IMS は、それ以降の画面に表示されたデータを、最初の画面で開始された会話の続きとして処理します。

プログラムが SPA をリトリーブした後で、プログラム A は端末から入力メッセージをリトリーブすることができます。メッセージを処理した後は、プログラム A は会話を続行することも、あるいは終了させることもできます。

会話を継続する場合は、プログラム A は次のいずれかを実行できます。

- メッセージを送信した端末に応答します。
- 端末に応答し、別の会話型プログラム (例えば、プログラム B) に会話を渡します。これは、据え置きプログラム間通信 と言われます。

  定義 : 据え置きプログラム間通信は、プログラム A が端末に応答し、それから別の会話型プログラム (プログラム B) に制御を渡すことを意味します。制御をプログラム B に渡した後は、プログラム A は会話の一部ではありません。端末で作業している人には、このメッセージが 2 番目のプログラムに送られるということは分かりませんが、端末から入力された次の入力メッセージは、プログラム B に渡されます。

  制約事項: アプリケーションがインバウンド保護会話に関与している場合は、据え置きプログラム間通信は実行できません。アプリケーションは、この環境で据え置きプログラム間通信を実行しようとすると、状況コード X6 を受け取ります。

- 最初に発信元端末に応答せずに、別の会話型プログラムに会話の制御を渡します。これは即時プログラム間通信 と呼ばれます。

  定義 : 即時プログラム間通信は、発信元端末に応答せずに、別の会話型プログラムに制御を直接渡すものです。この通信を行った場合には、会話を渡されたプログラムが、端末への応答を行う必要があります。会話を継続するためには、プログラム B は、プログラム A が行ったものと同じ選択をします。つまり、発信元端末に応答し、制御を保持するか、あるいは据え置きプログラム間通信または即時プログラム間通信を行って、制御を渡すことができます。

  制約事項: アプリケーションがインバウンド保護会話に関与している場合は、即時プログラム間通信は実行できません。アプリケーションは、この環境で即時プログラム間通信を実行しようとすると、U711 で異常終了します。

会話を終了する場合は、プログラム A は次のいずれかを実行できます。

- SPA のトランザクション・コード域の最初のバイトにブランクを移動してから、SPA を IMS に戻します。
- 端末に応答し、非会話型プログラムに制御を渡します。これは据え置きプログラム間通信とも呼ばれますが、プログラム A は、別のアプリケーション・プログラムに制御を渡す前に、会話を終了します。 2 番目のアプリケーション・プログラムは、MPP であるか、会話型プログラムからのトランザクションを処理する、トランザクション指向 BMP になります。

スクラッチパッド域 (SPA) についての重要な考慮点

プログラム A が、プログラム B に会話の制御を渡すときには、プログラム B は、プログラム A が会話を継続するために SPA に保管したデータを受け取る必要があります。 IMS は、プログラム B がその最初のメッセージ呼び出しを出すときに、トランザクション用の SPA をプログラム B に渡します。

SPA には、メッセージが保持されます。切り捨てデータ・オプションがオンの場合、保存される SPA のサイズは、会話内のトランザクションで最も大きい SPA のものになります。

例えば、会話が TRANA (SPA=100) で開始し、プログラムが TRANB (SPA=50) に切り替えた場合は、TRANB の入力メッセージには、100 バイトの SPA セグメントが含まれます。 IMS は、TRANB が最初の 50 バイトだけを受け取るよう、SPA のサイズを調整します。

会話におけるリカバリーの考慮事項

会話には複数のステップがあり、複数のアプリケーション・プログラムが関係する場合があるため、次の事項について考慮してください。

- よりリカバリーしやすくする方法の 1 つは、すべてのデータベース更新が会話の最終ステップで行われるような会話を設計することです。会話が異常終了しても、この方法であれば、会話の同じステップですべての更新が行われるので、IMS はすべての更新をバックアウトすることができます。会話の最終ステップでデータベースを更新すると、会話の各ステップからの入力を使用できるので、これもよい方法です。

- 会話は、会話のどのステップでも異常終了する可能性がありますが、 IMS は、会話の最終ステップのデータベース更新と出力メッセージの結果だけをバックアウトします。異常終了の結果、その処理の一部が正確ではなくなった可能性がある場合でも、 IMS は、これまでのステップのデータベース更新のバックアウトまたは出力メッセージの取り消しをおこないません。
- 特定の IMS システム・サービス呼び出しは、プログラムがその一部の処理が無効であると判断したときに役立ちます。これらの呼び出しには、ROLB、SETS、SETU、および ROLS が含まれます。ロールバック呼び出し (ROLB) は、プログラムがデータベースに加えたすべての変更をバックアウトします。さらに ROLB は、プログラムの最後のコミット・ポイント以降にプログラムが作成した出力メッセージ (高速 PCB で送られたメッセージを除く) も取り消します。

  SETS または SETU、ならびに ROLS (トークンを使用する) 呼び出しは、協働して、アプリケーション・プログラムが、プログラムの呼び出し処理内に中間バックアウト・ポイントを設定するのを可能にします。アプリケーション・プログラムは、最高 9 個の中間バックアウト・ポイントを設定することができます。プログラムは、SETS または SETU を呼び出しを使用して、それぞれの点ごとにトークンを指定する必要があります。同じトークンを使用して引き続き ROLS 呼び出しを出すと、すべてのデータベース変更をバックアウトすることができ、SETS または SETU 呼び出し以降に処理された非高速メッセージをすべて破棄します。

  定義 : トークンは 4 バイトの ID です。
- プログラムは、高速 PCB を使用して、端末で作業している人およびマスター端末オペレーターにメッセージを送信することができます。アプリケーション・プログラムが、高速 PCB を使用してメッセージを挿入するときには、 IMS は、コミット・ポイントを検出するまででではなく、すべてのメッセージを受け取るまで待ってから、そのメッセージをその宛先に転送します。 (この場合、「挿入」とは、アプリケーション・プログラムがメッセージを送信し、そのメッセージを IMS が受け取った状況のことを言います。「転送」とは、IMS がメッセージを、その宛先に送信し始めるときのことを言います。) そのため、プログラムが異常終了した場合でも、 IMS がすべてのメッセージを受け取っていれば、メッセージは転送されます。高速 PCB で送信されたメッセージは、プログラムが異常終了するか、ROLB 呼び出しを出した場合でも、その最終宛先に送られます。
- 以前の処理が正確に行われたかどうかを検査し、不正確であると判断した処理を訂正するためには、会話型異常終了ルーチン DFSCONE0 を使用することができます。

  関連資料: DFSCONE0 について詳しくは、「*IMS V15* 出口ルーチン」を参照してください。
- MPP を作成して SPA を検査し、端末で作業している人に異常終了が起きたことを通知するメッセージを送り、必要なデータベース呼び出しを行い、ユーザー作成またはシステム提供の出口ルーチンを使用してそれをスケジュールすることができます。

関連概念:

124 ページの『他のプログラムまたは端末の場合』

出力メッセージの宛先の識別

アプリケーション・プログラムは、別のアプリケーション・プログラムまたは IMS 端末に、メッセージを送信することができます。出力メッセージを送信するために、プログラムは呼び出しを出して、 I/O PCB または代替 PCB を参照します。I/O PCB および代替 PCB は、論理端末とアプリケーション・プログラムが通信する他のアプリケーション・プログラムを表します。

定義**:** 代替 *PCB* は、データ通信プログラム連絡ブロック (DCPCB) であり、入力メッセージを発信した端末以外の出力メッセージの宛先を示すものです。

関連概念:

50 ページの『バッチ・メッセージ処理: トランザクション指向 BMP』

発信元端末

入力メッセージを送信した論理端末にメッセージを送るためには、プログラムは I/O PCB を使用します。 IMS は、プログラムがメッセージを受信すると、I/O PCB にメッセージを送った論理端末の名前を入れます。

結果として、プログラムは、メッセージを送信する前に I/O PCB になにも行う必要はありません。 プログラムがバッチ指向 BMP または CPI 通信ドリブン・プログラムからメッセージを受信した場合、I/O PCB に入れるために使用できる論理端末名はありません。 上記の場合には、論理端末名フィールドはブランクになります。

関連概念:

『出力メッセージの宛先の識別』

他のプログラムまたは端末の場合

入力メッセージを送信した端末とそれ以外の端末に、あるいはそれ以外の端末だけに出力メッセージを送信したい場合には、代替 PCB を使用します。プログラムの PCB が生成される場合には、特定の論理端末の代替 PSB を設定することができ、代替 PCB を修正可能として定義することができます。 プログラムの実行中に変更可能代替 PCB の宛先をプログラムは変更することができ、出力メッセージを複数の代替宛先に送信することができます。

アプリケーション・プログラムは、発信元端末で作業をする人が、それ以上メッセージを出す前に、その発信元端末に対して応答しなければならない場合があります。これは、端末が応答モードあるいは会話型モードのときに起こります。

- 応答モード は、通信回線、端末、またはトランザクションに適用することができます。応答モードが有効な場合には、IMS は、プログラムが以前の入力メッセージに応答を送信するまで、通信回線または端末からの入力を受け入れません。プログラムがトランザクションを処理し、端末に応答を戻すまで、発信元端末は使用不能になります (例えば、キーボード・ロックなど)。

  高速機能トランザクションを含む応答モード・トランザクションが処理されるときに、アプリケーションが、I/O PCB または代替 I/O PCB を使用して、端末に戻される応答を挿入するのではなく、代替 PCB (プログラム間通信) にメッセ

ージを挿入する場合、2 番目以降のアプリケーション・プログラムが発信元端末に応答し、その応答を満たす必要があります。 IMS は、端末を応答モードのままにします。

アプリケーション・プログラムが正常に終了し、I/O PCB、代替 I/O PCB、または代替 PCB のいずれかに対して、ISRT 呼び出しを出さない場合には、 IMS はシステム・メッセージ DFS2082I を発信元端末に出して、すべての応答モード・トランザクション (高速機能トランザクションを含む) に対する応答を満たします。

通信回線と端末を応答モードで操作するか、応答モードで操作しないか、あるいは応答モードと定義されたトランザクションを処理する場合にだけ、応答モードで操作するかを、定義することができます。 IMS システム定義で、TYPE マクロおよび TERMINAL マクロに、それぞれ通信回線と端末の応答モードを指定します。応答モード・トランザクションとしてすべてのトランザクションを定義することができます。 IMS システム定義で、TRANSACT マクロに指定してください。応答モードは、以下の場合に有効です。

- 通信回線が、応答モードとして定義されている場合
- 端末が、応答モードとして定義されている場合
- トランザクション・コードが、応答モードとして定義されている場合

- 会話型モードは、トランザクションに適用します。プログラムが会話型トランザクションを処理している場合には、プログラムは、端末から入力メッセージを受け取るたびに、メッセージを出した発信元端末に応答しなければなりません。

これらの処理モードでは、プログラムは、発信元端末に応答しなければなりません。ただし、発信元端末は、2 つのコンポーネント (例えばプリンターおよびディスプレイ) からなる物理端末で構成されることもあります。物理端末が 2 つのコンポーネントで構成されている場合には、各コンポーネントには異なる論理端末名があります。出力メッセージを端末のプリンター部分に送るためには、プログラムは、入力メッセージと関連のある名前ではなく、異なった論理端末名を使用しなければなりません。つまり、プログラムは代替宛先に、出力メッセージを送信しなければなりません。このような状況で、プログラムが使用できる特別な種類の代替 PCB があり、代替応答 *PCB* と呼ばれます。

定義 **:** 排他、応答、または会話型モードが有効な場合には、代替応答 PCB によりメッセージを送信することができます。詳しくは、次のセクションを参照してください。

代替応答 PCB

代替応答 PCB の宛先は、論理端末である必要があります。別のアプリケーション・プログラムを示すために、代替応答 PCB を使用することはできません。応答モードまたは会話型モードで代替応答 PCB を使用する場合、代替応答 PCB によって示される論理端末は、メッセージを出した論理端末と同じ物理端末でなければなりません。

これらの処理モードで、メッセージを受け取った後、アプリケーション・プログラムは、次のいずれかに ISRT 呼び出しを出して応答しなければなりません。

- I/O PCB

- 代替応答 PCB
- 宛先が別のアプリケーション・プログラムである代替 PCB (つまり、プログラム間通信)
- 宛先が ISC リンクである代替 PCB。 これは、フロントエンド通信メッセージの場合にだけ使用可能です。

  関連資料: フロントエンド通信メッセージの詳細については、「*IMS V15* 出口ルーチン」を参照してください。

これらの基準のいずれにも合わない場合には、メッセージ DFS2082I が端末に送信されます。

高速 PCB

代替 PCB を高速 *PCB* として指定することも考慮してください。高速指定は、プログラムが異常終了するか、ROLL、ROLB、または ROLS 呼び出しを出した場合に、アプリケーション・プログラムが挿入したメッセージが、実際に宛先に転送されるかどうかということに関連があります。すべての PCB について、プログラムが異常終了した場合、あるいは ROLL、ROLB、または ROLS 呼び出しを出した場合は、挿入されても転送できなかったメッセージは取り消されますが、転送できるメッセージは取り消されません。

定義 : 高速 PCB は、代替応答 PCB で、これを使用すると、プログラムが、非高速 PCB を使用した場合よりも速く、宛先端末にメッセージを転送することができます。

非高速 PCB の場合には、プログラムがコミット・ポイントに達するまでは、メッセージを宛先に転送することはできません。コミット・ポイントが生じるのは、プログラムが終了したとき、 CHKP 呼び出しを出したとき、次の入力メッセージを要求したとき、およびトランザクションが MODE=SNGL を指定して定義されているときです。

高速 PCB では、IMS に完全なメッセージが渡されると、メッセージを宛先に転送することができるようになります。メッセージが転送されるのは、コミット・ポイントが生じたときに加えて、アプリケーション・プログラムが、その PCB を使用して PURG 呼び出しを出すか、または次の入力メッセージを要求するときです。

アプリケーションのメッセージ処理要件に合わせるために、データ通信管理者に、以下の質問に対する答えを示してください。

- 端末が別のメッセージを入力できる状態になる前に、プログラムは、端末への応答を要求されるか。
- プログラムは、入力メッセージを送信する端末にだけ応答しているか。
- プログラムが、別の端末やプログラムにもメッセージを送信する必要があるか、または代替宛先が 1 つだけあるか。
- プログラムが、出力メッセージを送信しなければならない別の端末はどれか。
- プログラムが異常終了する前に、プログラムが出力メッセージを送信できなければならないか。

関連概念:

122 ページの『会話におけるリカバリーの考慮事項』
124 ページの『出力メッセージの宛先の識別』

第 7 章 APPC に対応したアプリケーションの設計

拡張プログラム間通信 (APPC) は、プログラム間通信のための IBM 推奨プロトコルです。アプリケーション・プログラムをネットワーク内に配布することができ、それらは、多くのハードウェア体系やソフトウェア環境内で、相互に通信できます。

関連資料 : APPC の詳細については、以下を参照してください。

- *IMS V15* コミュニケーションおよびコネクション - この資料は、LU 6.2 装置対応の APPC の概説、ならびに CPI 通信概念について説明しています。

APPC および LU 6.2 の概要

APPC では、APPC プロトコルを使用するアプリケーション・プログラムにより、LU 6.2 装置から IMS トランザクションに入ることができます。 LU 6.2 アプリケーション・プログラムは、APPC をサポートしている LU 6.2 装置上で稼働します。

APPC は、以下のことを可能にする環境を作成します。

- リモート LU 6.2 装置で、IMS のローカル・トランザクションおよびリモート・トランザクションに加わる。
- 既存のアプリケーション・プログラムのコーディング内容を変更することなく、IMS アプリケーション・プログラムでトランザクションの出力を LU 6.2 装置に挿入する。
- 新規アプリケーション・プログラムで、LU 6.2 装置の機能を十分に利用する。
- IMS により、分散同期点機能がない LU 6.2 環境で、データ保全性が提供される。

アプリケーション・プログラムのタイプ

APPC/IMS は IMS TM の一部であり、アプリケーション・プログラムと通信するために、CPI 通信インターフェースを使用します。

APPC/IMS は、LU 6.2 処理のために次のタイプのアプリケーション・プログラムをサポートします。

- 標準 DL/I
- 修正標準 DL/I
- CPI 通信ドリブン

標準 DL/I アプリケーション・プログラム

標準 DL/I アプリケーション・プログラムは、どのような CPI 通信呼び出しも出さず、あるいはどのような CPI-C 会話も確立しません。 このアプリケーション・プログラムは、 IMS API を使用している他の LU タイプ端末装置に置き換わる、LU 6.2 タイプ製品と通信することが可能です。標準 DL/I アプリケーション・プ

ログラムは、修正、再コンパイル、あるいはバインドをする必要がなく、それが現在行われているとおりに実行されます。

修正標準 DL/I アプリケーション・プログラム

修正標準 DL/I アプリケーション・プログラムは、 DL/I 呼び出しと CPI 通信呼び出しの両方を使用する、標準 DL/I オンライン IMS TM アプリケーション・プログラムです。全機能データベース、DEDB、MSDB、および Db2 for z/OS データベースにアクセスできる、 MPP、BMP、あるいは IFP が可能です。

修正標準 DL/I アプリケーション・プログラムは、 CPI 通信 (CPI-C) 呼び出しを使用して、 LU 6.2 および非 LU 6.2 の混合ネットワークをサポートします。同じアプリケーション・プログラムが、CPI 通信の ALLOCATE verb が出されないある 1 つの実行時には、標準 DL/I となり、CPI 通信 ALLOCATE verb が出される別の実行時には、修正標準 DL/I になります。

修正標準 DL/I アプリケーション・プログラムは、 I/O PCB に対する DL/I GU 呼び出しを使用してメッセージを受信し、 DL/I ISRT 呼び出しを使用して応答を出力します。また、CPI 通信呼び出しは、新規会話を割り振り、その会話のデータの送信や受信を行うこともできます。

関連資料: CPI 通信呼び出しのリストについては、「CPI Communications Reference」を参照してください。

他の LU 6.2 装置あるいは同一のネットワーク宛先との会話を確立するのに、既存の標準 DL/I アプリケーション・プログラムを使用したい場合は、修正標準 DL/I アプリケーション・プログラムを使用してください。標準 DL/I アプリケーション・プログラムは、任意で修正され、新規機能、新規のアプリケーションおよびトランザクション定義、そして修正済みの DL/I 呼び出しを使用して、LU 6.2 アプリケーション・プログラムを開始します。プログラム呼び出しおよびパラメーターは、IMS 提供の暗黙 API および CPI 通信明示 API を使用する際に利用可能です。

CPI 通信ドリブン・プログラム

CPI 通信ドリブン・アプリケーション・プログラムは、 **コミット** および **バックアウト**呼び出し、および入出力メッセージ処理に CPI 通信インターフェース呼び出しあるいは LU 6.2 verb を使用します。このアプリケーション・プログラムは CPI 通信明示 API を使用し、全機能データベース、 DEDB、MSDB、および Db2 for z/OS データベースをアクセスすることができます。 LU 6.2 装置は、会話の割り振りによってのみ、CPI 通信ドリブン・アプリケーション・プログラムを活動化することができます。

標準 DL/I あるいは修正標準 DL/I アプリケーション・プログラムとは異なり、CPI 通信ドリブン・プログラムの入出力メッセージ処理では、APPC/MVS™ バッファーを使用し、IMS メッセージ・キューイングをバイパスします。これらのこれらのアプリケーション・プログラムは、IMS メッセージ・キューを使用しないので、パートナー LU 6.2 システムを使用して、それら自身の実行を制御できます。 IMS APSB 呼び出しは、IMS データベースおよび代替 PCB にアクセスするために、PSB を割り振ることを可能にします。

アプリケーション・プログラムは、共通プログラミング・インターフェース・リソース・リカバリー (CPI-RR) SRRCMITverb を使用して、バックアウトのための IMS 同期点および CPI-RR SRRBACK verb を開始します。 CPI 通信ドリブン・アプリケーション・プログラムは、CPI-RR 呼び出しを使用して、プログラムが終了する前に IMS 同期点処理を開始します。

CPI 通信ドリブン・アプリケーション・プログラムは、以下のことを行うことが可能です。

- あらゆるタイプのデータベースへのアクセス
- 標準 DL/I および修正標準 DL/I アプリケーション・プログラムのような大容量メッセージの受信および送信
- CPI 通信呼び出しを使用する入出力のフローの制御
- パートナー LU 6.2 装置を使用する複数会話の割り振り
- 会話パートナーを使用する同期の起因
- IMS 暗黙 API (例えば、IMS キュー・サービス) の使用
- 使用されている API を問わない IMS サービス (例えば、プログラム終了の同期点) の使用

アプリケーションの目的

各アプリケーション・タイプは異なる目的を持っていて、その使いやすさは、プログラムが標準 DL/I であるか、修正標準 DL/I であるか、また CPI 通信ドリブン・アプリケーション・プログラムであるかどうかによって変わります。

次の表では、各アプリケーション・タイプ (標準 DL/I、修正標準 DL/I、および PI-C ドリブン) の、目的および使用の難易度をリストしています。この情報は、IMS リソースの使用とバランスさせてください。

表 *26. APPC* でのアプリケーション・プログラムの使用：

	使用の難易度		
アプリケーション・プログラムの目的	標準 **DL/I** プログラム	修正標準 **DL/I** プログラム	**PI-C** ドリブン・プログラム
照会	容易	無色	非常に難しい
データ入力	容易	容易	難しい
大量転送	容易	容易	無色
協調	難しい	難しい	最適
分散	難しい	無色	最適
高保全性	無色	無色	最適
クライアント・サーバー	容易	無色	非常に難しい

会話属性の選択

LU 6.2 トランザクション・プログラムは、 IMS によってトランザクションが処理される方法を指示します。同期および非同期の 2 種類の処理モードが利用可能です。

同期会話

入力データの送信に使用された同一会話において、パートナーが応答を待っている場合、会話は同期です。

同期処理は、SEND_DATA verb の後に RECEIVE_AND_WAIT verb を出すことによって要求されます。 IMS 応答モード・トランザクションおよび IMS 会話モード・トランザクションには、このモードを使用してください。

例:

```
MC_ALLOCATE TPN(MYTXN)
MC_SEND_DATA  'THIS CAN BE A RESPONSE MODE'
MC_SEND_DATA  'OR CONVERSATIONAL MODE'
MC_SEND_DATA  'IMS TRANSACTION'
MC_RECEIVE_AND_WAIT
```

非同期会話

入力データの送信後、パートナー・プログラムが正常に会話を割り振り解除する場合、会話は非同期です。出力は、DFSASYNC の TP 名に送信されます。

非同期処理は、SEND_DATA verb の後に DEALLOCATE verb を出すことによって要求されます。 IMS コマンド、メッセージ通信、および非応答トランザクション、非会話型トランザクションには、非同期処理を使用してください。

例:

```
MC_ALLOCATE TPN(OTHERTXN)
MC_SEND_DATA  'THIS MUST BE A MESSAGE SWITCH, IMS COMMAND'
MC_SEND_DATA  'OR A NON-RESP NON-CONV TRANSACTION'
MC_DEALLOCATE
```

非同期出力の送達

非同期出力は、送達のための IMS メッセージ・キューに保留されています。出力がエンキューされると、IMS はこの出力を送信しようと会話の割り振りを行います。それが失敗すると、IMS は後で送達するために、出力を保留にします。この送達は、オペレーター・コマンド (/ALLOC)、あるいはこの LU 6.2 宛先に対する新規メッセージのエンキューによって、開始することができます。

MSC の同期および非同期会話

同期および非同期どちらの APPC 会話からの MSC リモート・アプリケーション・メッセージも、複数システム結合機能 (MSC) リンク上にキューイングすることができます。その後でこれらのメッセージを、MSC リンクにまたがって、リモート IMS に処理のために送信することができます。

関連概念:

142 ページの『LU 6.2 フロー・ダイアグラム』

会話タイプ

APPC 会話タイプは、データがどのように APPC verb に渡されるか、またデータがどのように APPC verb からリトリーブされるかを定義します。

ファイル・ブロックと概念の上で類似し、会話の両方の側に影響を与えます。

APPC は、2 つの会話タイプをサポートします。

基本的会話
: この低レベル会話により、プログラムは、標準化形式のデータを交換することができます。この形式は、2 バイトの長さフィールド (LL と呼ばれる) を含むデータ・ストリームで、 LL は、次の長さフィールドの前まで続く、データの長さを指定します。代表的なデータ・パターンは以下のとおりです。

  ```
  LL, data, LL, data
  ```

  LL、データの各グループは、論理レコードと呼ばれます。基本的会話を使用して、1 つの verb で複数セグメントを送信し、 1 つの verb で最大データを受信します。

マップ式会話
: この高レベル会話により、プログラムは、アプリケーション・プログラマーが承認したデータ形式の、任意のデータ・レコードを交換することができます。 1 つの送信 verb につき、1 つの受信 verb があり、 z/OS および VTAM® はバッファリングを処理します。

関連資料: 基本的会話およびマップ式会話の詳細については、以下を参照してください。

- *Systems Network Architecture: LU 6.2 Reference: Peer Protocols* および
- *Systems Network Architecture: Transaction Programmer's Reference Manual for LU Type 6.2*

会話状態

CPI 通信は、会話状態を使用して、次に実行する一連の処置を判別します。

以下は会話状態の例です。

RESET
: 通信が始まる前の初期状態。

SEND
: プログラムは、送信あるいはオプションで受信することが可能です。

RECEIVE
: プログラムは、受信あるいは中止しなければなりません。

CONFIRM
: プログラムは、パートナーに応答しなければなりません。

APPC verb には、以下のような基本規則があります。

- 会話を開始するプログラムが、最初に話します。

- 未解決にしておける APPC verb は、一度に 1 つだけです。
- プログラムは、送信と受信を交代で行ってください。
- 会話状態は、プログラムが出せる verb を判別します。

同期レベル

APPC 同期レベルは、会話状態を変更する際に使用されるプロトコルを定義します。

APPC および IMS は、次の同期レベル値をサポートします。

SYNCLVL=NONE
プログラムが呼び出しを出さないか、または同期に関係する戻りパラメーターを認識しないことを指定します。

SYNCLVL=CONFIRM
プログラムが、会話上の処理の確認を実行できることを指定します。

SYNCLVL=SYNCPT
z/OS リソース・リカバリー・サービス (RRS) リカバリー・プラットフォームのもとで会話中に更新される、リソースに関する整合コミット処理に、プログラムが参加することを指定します。このレベルでの会話は、保護会話とも呼ばれています。

また、IMS または RRS を同期点管理機能として指定することができます。

RRS=Y
AOS=B、AOS=S、または AOS=X の場合、SYNCLVL=NONE または CONFIRM を指定したトランザクションは IMS で同期点管理機能として処理されます。

AOS=B または AOS=Y の場合、SYNCLVL=SYNCPT を指定したトランザクションは、RRS で同期点管理機能として処理されます。

フロントエンド IMS システムがバックエンド IMS システムでもある共用メッセージ・キュー環境では、SYNCLVL=SYNCPT を指定したトランザクションは、RRS で同期点管理機能として処理されます。

非共用メッセージ・キュー環境では、SYNCLVL=SYNCPT を指定したトランザクションは、RRS で同期点管理機能として処理されます。

制約事項**:** AOS= 設定は、共用メッセージ・キュー環境のみに適用可能です。

RRS=N
AOS=B、AOS=S、または AOS=X の場合、SYNCLVL=NONE または CONFIRM を指定したトランザクションは IMS で同期点管理機能として処理されます。

バックエンド IMS システムで RRS=N が指定されている場合、SYNCLVL=SYNCPT を指定したトランザクションはフロントエンド IMS システムでのみ処理されます。ただし、フロントエンド IMS システムでも RRS=N が指定されている場合は、SYNCLVL=SYNCPT を指定したトランザクションはまったく処理されません。

SYNCLVL=SYNCPT を指定した会話の割り振りには、同期点管理機能として RRS が必要です。RRS は、コミットまたはバックアウト要求を更新されるリソースの参加所有者 (リソース・マネージャー) と調整することにより、保護リソースのコミットメントを制御します。IMS は、DL/I、高速機能データ、および IMS メッセージ・キューのリソース・マネージャーです。アプリケーション・プログラムは、データをコミットさせるか打ち切るかを決定し、この決定を同期点管理機能に通知します。 次に、同期点マネージャーは、この決定をサポートして、その処置をリソース・マネージャー間で調整します。

関連概念:

保護会話の活動化 (コミュニケーションおよびコネクション)

リソース・リカバリーの紹介

ほとんどのお客さまは、各自のビジネスの存続に必要不可欠なコンピューター・リソースを保守しています。これらのリソースは、コントロールされ同期化された方法で更新されたときに、保護リソース またはリカバリー可能リソース であると呼ばれます。これらのリソースは、すべてを (同じシステムに) ローカルに常駐させることも、 (ネットワークの複数のノード間に) 分散させることもできます。 複数の保護リソースの更新を一貫性のある方法で調整するためのプロトコルとメカニズムは、z/OS リソース・リカバリー・サービス (RRS) を備えた z/OS で提供されます。

リソース・リカバリーへの参加プログラム

次の図に示されているように、リソース・リカバリー環境は 3 つの参加プログラムによって構成されます。

- 同期点マネージャー
- リソース・マネージャー
- アプリケーション・プログラム

RRS は、同期点マネージャーであり、調整プログラムとも言われています。同期点マネージャーは、コミット要求 (またはバックアウト要求) をリソース・マネージャー (更新されるリソースの参加所有者) と調整することによって、保護リソースのコミットメントを制御します。これらのリソース・マネージャーは、同期点処理の参加プログラムとして知られています。 IMS は、DL/I、Fast Path、および Db2 for z/OS データが、該当の環境で更新されている場合は、それらに対するリソース・マネージャーとして参加します。

このリソース・リカバリー・プロトコルの最終参加プログラムは、アプリケーション・プログラム(保護リソースにアクセスして更新するプログラム) です。アプリケーション・プログラムは、データをコミットさせるか打ち切るかを決定し、この決定を同期点マネージャーに知らせます。 次に、同期点マネージャーは、この決定をサポートして、その処置をリソース・マネージャー間で調整します。

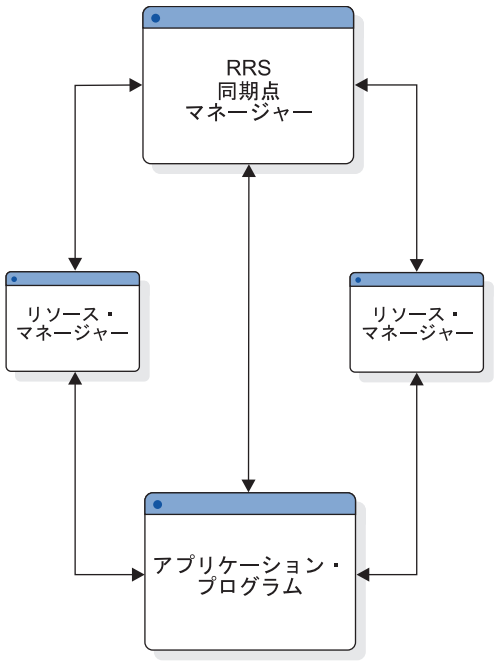


図 25. リソース・リカバリーへの参加プログラム

2 フェーズ・コミット・プロトコル

次の図に示すとおり、2 フェーズ・コミット・プロトコルは、同期点マネージャーおよびリソース・マネージャーの参加プログラムが関与し、3 番目の参加プログラム (アプリケーション・プログラム) によって行われたリソース群に対する更新の処理を、更新の全部を認めるか、どれも認めないかで保証するプロセスです。簡単に言うと、アプリケーション・プログラムがいくつかのリソースに対する変更をコミットすることを決める。このコミットが同期点マネージャーに行われる。同期点マネージャーは、このコミット呼び出しの適否についてすべてのリソース・マネージャーにポーリングする、というプロセスです。これは、準備フェーズであり、フェーズ 1 とも呼ばれます。各リソース・マネージャーは、コミットに対してイエスかノーかのボートを行います。

同期点マネージャーがすべてのボートを収集し終わると、フェーズ 2 が始まります。すべてのボートが変更をコミットすることであれば、フェーズ 2 のアクションはコミットです。それ以外の場合は、フェーズ 2 がバックアウトになります。システム障害、通信障害、リソース・マネージャー障害、またはアプリケーション障害は、2 フェーズ・コミット・プロセスの完了へのバリアにはなりません。

各種のリソース・マネージャーが行った作業は、リカバリー単位 *(UOR)* と呼ばれ、作業のある一貫性のあるポイントから別の一貫性のあるポイントまで、通常、あるコミット・ポイントから別のコミット・ポイントまでの時間を範囲とします。2 フェーズ・コミット・プロセスの対象は、このリカバリー単位です。

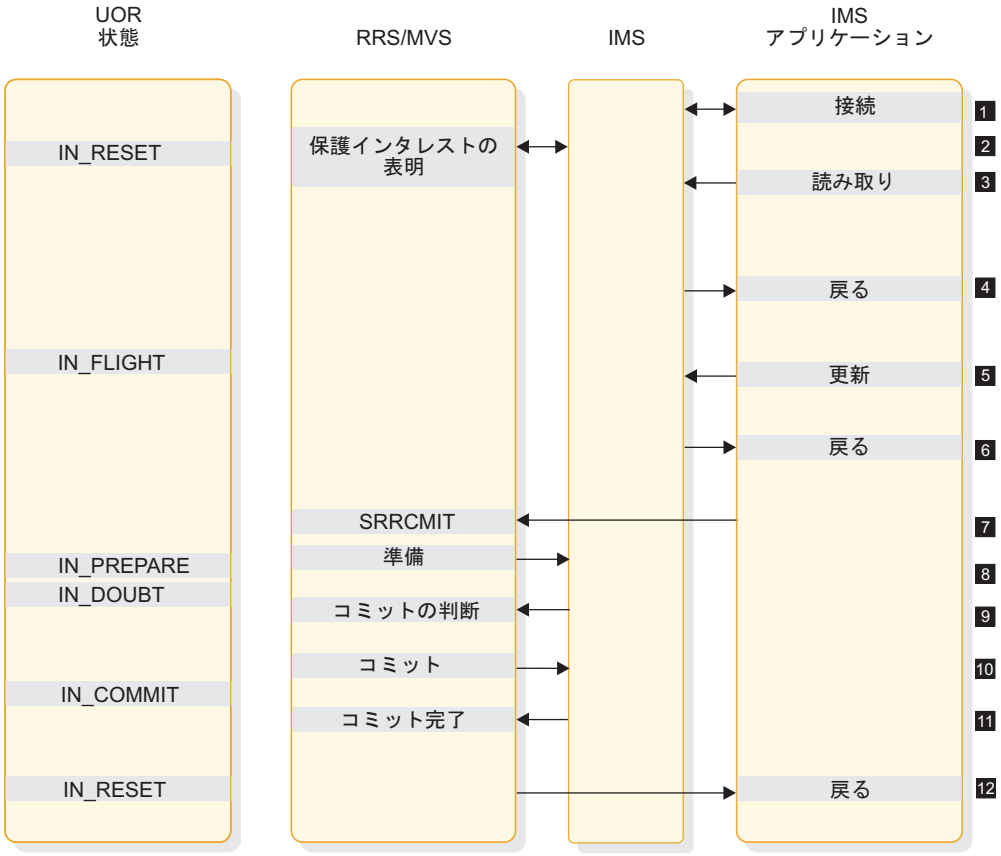


図 *26. 1* つのリソース・マネージャーによる *2* フェーズ・コミット・プロセス

注:

1. アプリケーションと IMS が接続を行う。
2. IMS が、アプリケーションによって開始された作業への保護インタレストを示す。これによって、RRS に、IMS が 2 フェーズ・コミット処理に参加することが伝えられる。
3. アプリケーションが IMS リソースに対する読み取り要求を行う。
4. 読み取り要求に続いて、アプリケーションに制御が戻される。
5. アプリケーションが保護リソースを更新する。
6. 更新要求に続いて、アプリケーションに制御が戻される。
7. z/OS アプリケーションが SRRCMIT 呼び出しによって、更新を定着させるよう要求します。
8. RRS が、準備 (フェーズ 1) 処理を行うために、IMS を呼び出す。
9. IMS が RRS にコミットについてのボートを戻す。
10. RRS が、コミット (フェーズ 2) 処理を行うために、IMS を呼び出す。
11. IMS が RRS に、フェーズ 2 を完了したことを知らせる。
12. コミット要求に続いて、アプリケーションに制御が戻される。

ローカル対分散

リカバリー処理に関与する参加プログラムの常駐場所によって、そのリカバリーがローカルと見なされるか分散と見なされるかが決められます。ローカル・リカバリー・シナリオでは、すべての参加プログラムが同じ単一システムに常駐します。分散リカバリー・シナリオでは、参加プログラムが複数システムに散らばっています。次の図は、分散リソース・リカバリーにおける Resource Manager 参加プログラム間の通信を示しています。 RRS によって提供される機能には、ローカル・リカバリーと分散リカバリーの間に概念上の違いはありません。しかし、リモート同期点マネージャーを関与させるために、元の同期点マネージャーの機能を分散するには、特殊なリソース・マネージャーが必要です。 APPC 通信リソース・マネージャーが分散環境でこのサポートを提供します。

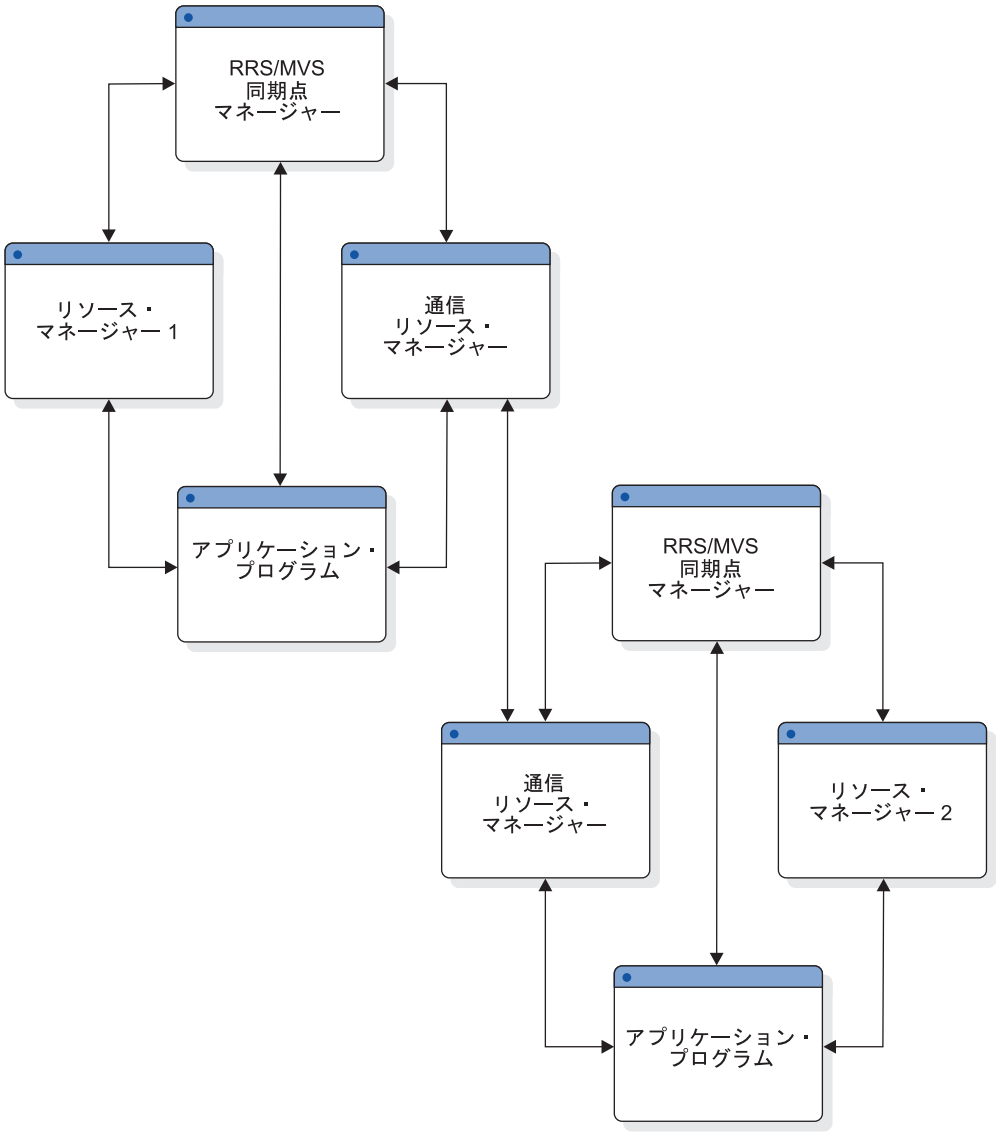


図 27. 分散リソース・リカバリー

z/OS リソース・リカバリー・サービス サポートの要約

z/OS リソース・リカバリー・サービス (RRS) は、z/OS で稼働しているアプリケーションがローカル・リソースや分散リソースにアクセスしてこれらのリソースのリカバリー管理をシステムで調整できるような、システム・リソース・リカバリー・プラットフォームを提供します。

このサポートには以下のものが含まれます。

- 2 フェーズ・コミット・プロセスを調整する同期点マネージャー
- アプリケーション・プログラムで使用するための SAA コミットおよびバックアウト呼び出し可能サービス
- リソースをアプリケーション・インスタンスに関連づけるメカニズム
- RRS との 2 フェーズ・コミット・プロセスに、リソース・マネージャーが登録し参加するためのサービス
- リソース・マネージャーが、アプリケーション・インスタンスへのインタレストを表現でき、コミットおよびバックアウト要求を通知されるようにするサービス
- リソース・マネージャーが、リソースを一貫性のある状態に復元するために、システム・データを取得できるようにするサービス
- 通信リソース・マネージャー (APPC/Protected Conversations: APPC/PC と呼ばれます)。これにより、分散アプリケーションは、参加するローカル・リソース・マネージャーとリカバリーを調整することができます。

制約事項:

- 拡張回復機能 (XRF)

  IMS-XRF 環境で保護会話を実行した場合、代替システムが、アクティブ・システムによって開始された未完了の作業を再開し解決できる保証はありません。失敗したリソース・マネージャーは、再始動するときに元の RRS システムがまだ使用可能であれば、その RRS に対して再登録しなければならないために、上記のプロセスは保証されません。アクティブ・システムの RRS が使用可能でない場合にのみ、XRF の代替機能がシスプレックスの別の RRS に登録し、アクティブにならない未完了のリカバリー単位のデータを取得することができます。

  推奨事項: IMS は、未確定のリカバリー単位を、それらが解決されるまでいつまでも保持しています。したがって、できるだけ早く元のアクティブ・システムへのスイッチバックを行い、それによって、リカバリー単位の情報をピックアップし、関与しているリソース・マネージャーのすべての作業を解決して完了する必要があります。これが不可能な場合は、未確定のリカバリー単位をコマンドによって解決することができます。

- DBCTL 環境でのバッチおよび非メッセージ・ドリブン BMP

  分散同期点は、IMS バッチ環境をサポートしません。 DBCTL 環境では、インバウンド保護会話は使用できません。しかし、DBCTL 環境での BMP はアウトバウンド保護会話を割り当てることができます。これは分散同期点および RRS にサポートされます。

分散同期点

分散同期点サポートにより、IMS およびリモート・アプリケーション・プログラム (APPC または OTMA) は、保護会話に参加することができ、整合されたリソース更新とリカバリーを行うことができます。このサポート以前は、IMS が同期点マネージャーとしての役割を果たしました。この新規シナリオでは、会話の参加プログラム (アプリケーション・プログラムと IMS) の代わりに z/OS が同期点処理を管理します (IMS の役割はリソース・マネージャーになりました)。

z/OS は、z/OS リソース・リカバリー・サービス (RRS) という、システム・リソース・リカバリー・プラットフォームを実装します。 RRS は、共通プログラミング・インターフェース - リソース・リカバリー (CPI-RR) をサポートします。これは、SAA 共通プログラミング・インターフェース の 1 つのエレメントであり、リソース・リカバリーを定義し、ローカル・リソースと 分散リソースに対して整合性のあるリソース・リカバリー管理を行います。RRS に加えて、通信リソース・マネージャー (APPC/Protected Conversations: APPC/PC と呼ばれます) がリカバリーの分散を行います。

APPC 環境では、アプリケーション・プログラムが、SYNC_LEVEL=SYNCPT と 指定された APPC 会話を割り当てたときに保護会話が開始されます。このシナリオでは、IMS と APPC の両方がリソース・マネージャーです。 OTMA 環境では、OTMA がリソース・マネージャーでないために、追加のコードが必要です。必要な追加コードは、OTMA アダプター (IBM 提供のものか同等のもの) です。このアダプターは、IMS に対して (OTMA メッセージ接頭部で)、このメッセージが保護会話の一部であること、したがって IMS とアダプターは、 RRS によって管理される整合コミット処理の参加プログラムであることを知らせます。

アプリケーション・プログラマーは、同期点マネージャーとして、IMS ではなく RRS を使用する APPC アプリケーション・プログラム (ローカルおよびリモート) とリモート OTMA アプリケーション・プログラムを開発できるようになりました。この機能強化によって、複数プラットフォームにわたるリソースを、整合性のある方法で更新およびリカバリーすることができます。

分散同期点の概念

分散同期点サポートには、次のことが必要です。

- IMS における、RRS のもとでリソース・マネージャーとして機能できるようにする変更
- アプリケーション・プログラム環境に対する、保護会話でのアプリケーションの使用をサポートする変更
- いくつかのコマンドに対する、ユーザーを援助する変更

ネットワークへの影響

ネットワーク・トラフィックは、会話の参加プログラムと、同期点マネージャー相互の通信によって増大します。

LU タイプ 6.2 のアプリケーション・プログラミング・インターフェース

IMS アプリケーション・プログラムは、IMS 暗黙 LU 6.2 API を使用して、 LU 6.2 装置にアクセスすることができます。この API から、非 LU 6.2 装置タイプとの互換性を得ることができ、同じアプリケーション・プログラムが、LU 6.2 および非 LU 6.2 装置のどちらからも使用できます。

アプリケーション・プログラムに IMS 提供の処理を組み込むことによって、API を APPC インターフェースに追加します。新規あるいは再作成した IMS アプリケーション・プログラム用の APPC 機能用として、明示の CPI 通信インターフェースを使用することができます。

暗黙 API

暗黙 API は、APPC 会話に間接的にアクセスします。この API は、標準 DL/I 呼び出し (GU、ISRT、PURG) を使用して、データの送受信を行います。この API により、LU 6.2 プロトコル向けでないアプリケーション・プログラムが、 LU 6.2 装置を使用できるようになります。

この API は、新規および変更済みの DL/I 呼び出し (CHNG、INQY、 SETO) を使用して、LU 6.2 を利用します。既存の IMS アプリケーション・プログラミングをベースに使用して、この API を使用し、CPI 通信呼び出しを使用しない、 LU 6.2 向けのアプリケーションを作成することができます。 暗黙 API は、 LU 6.2 機能のうちのいくつかを使用しているだけですが、それは多くのアプリケーションを単純化するのに役立っています。 さらに、暗黙 API は、メッセージ・キューイングおよび自動非同期メッセージ送達というような、LU 6.2 以外の機能も可能にします。

IMS は、すべての CPI 通信呼び出しを暗黙 API の下に生成します。 アプリケーション対話は、厳密に IMS メッセージ・キューに対して行われます。

リモート LU 6.2 システムは、LU 6.2 フローを処理できなければなりません。APPC/MVS は、これらのフローの生成を、暗黙 API を使用している IMS アプリケーション・プログラムによって出される、CPI 通信呼び出しから行います。IMS アプリケーション・プログラムは、明示 API を使用して、CPI 通信を直接出すことができます。 このことは、不完全な LU 6.2 をインストールしているか、または IMS 暗黙 API サポートと互換性のないリモート LU 6.2 システムの場合に便利です。

既存の API は、以下のように拡張されます。

- 非同期 LU 6.2 出力が、LU 6.2 宛先を参照する代替 PCB を使用して、生成されます。 DL/I CHNG 呼び出しは、パラメーターを与えて LU 6.2 宛先を指定します。 デフォルトは、パラメーターを省略した場合に使用されます。
- アプリケーション・プログラムは、会話タイプ (基本あるいはマップ式)、sync_level (NONE、CONFIRM、または SYNCPT)、および非同期あるいは同期会話のような、現行の会話属性をリトリーブできます。
- 端末装置メッセージ通信は、LU 6.2 装置間で使用することが可能です。

明示 API

明示 API (CPI 通信 API) は、APPC 会話に直接アクセスする、任意の IMS アプリケーション・プログラムで使用できます。

IMS リソースは、アプリケーションが APSB (Allocate PSB) 呼び出しを出す場合のみ、 CPI 通信ドリブン・アプリケーション・プログラムで利用可能です。 CPI 通信ドリブン・アプリケーション・プログラムは、IMS 同期点あるいはバックアウトを開始するか、あるいは SYNCLVL=SYNCPT が指定されている場合は、同期点の決定を z/OS リソース・リカバリー・サービス 同期点管理プログラムに通知するには、 CPI-RR SRRCMIT および SRRBACK verb を使用しなければなりません。

関連資料: SRRCMIT および SRRBACK verb に関する記述については、「*SAA CPI* 資源回復解説書」を参照してください。

LU 6.2 パートナー・プログラム設計

LU 6.2 装置から送信されたトランザクションのフローは、会話属性および同期レベルによって異なります。異なる結果が発生すると、パートナー・システムはそれに応じた処置をとります。

LU 6.2 フロー・ダイアグラム

以下のダイアグラムは、LU 6.2 デバイスから送信されるトランザクションのフローを示します。

以下の図は次のことを示しています。

- 同期あるいは非同期の LU 6.2 アプリケーション・プログラムと、単一 (ローカル) IMS システム内の、IMS アプリケーション・プログラムの間のフロー。
- 単一 (ローカル) IMS システム内の同期あるいは非同期の LU 6.2 アプリケーション・プログラムと、リモート IMS システム内の IMS アプリケーション・プログラムとの間の、複数システム結合機能 (MSC) リンクを経由したフロー。
- SYNC_LEVEL=SYNCPT を使用したバックアウト・シナリオ。

バッファリングと、制御データのユーザー・データとのカプセル化に差異があると、フロー内に変化が生じます。制御データは、Receive APPC verb からの戻りフィールドで、 Status_received、Data_received、Request_to_send_received の 3 つがあります。これらの差異に基づく変化が生じても、フローの機能や使用に影響はありません。

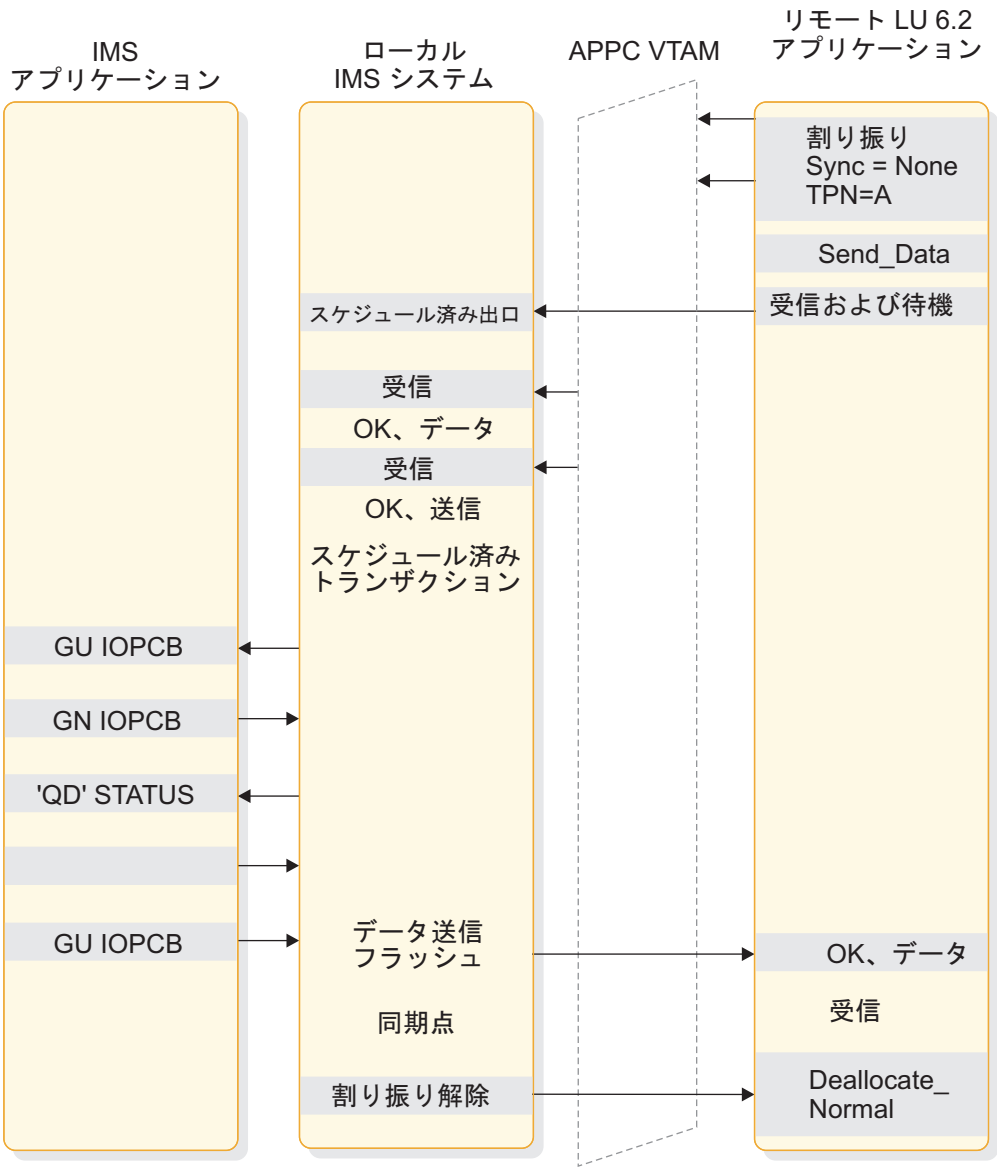


図 28. *Sync_level=None* を指定する場合の、ローカル *IMS* 同期トランザクションのフロー

144 ページの図 29 は、Sync_level が Confirm である場合のローカル同期トランザクションのフローを示しています。

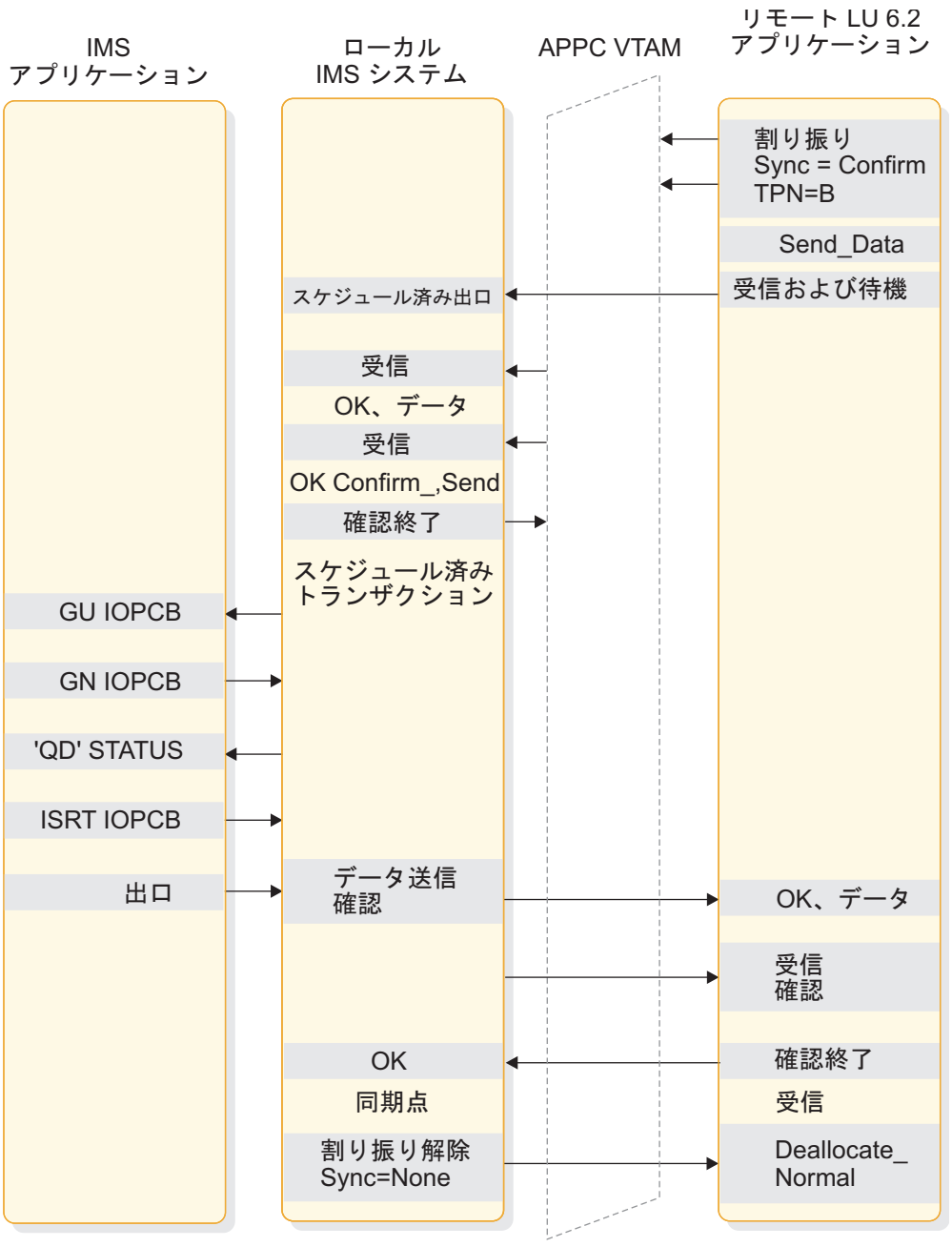


図 29. *Sync_level=Confirm* を指定する場合の、ローカル *IMS* 同期トランザクションのフロー

145 ページの図 30 は、Sync_level が None である場合のローカル非同期トランザクションのフローを示しています。

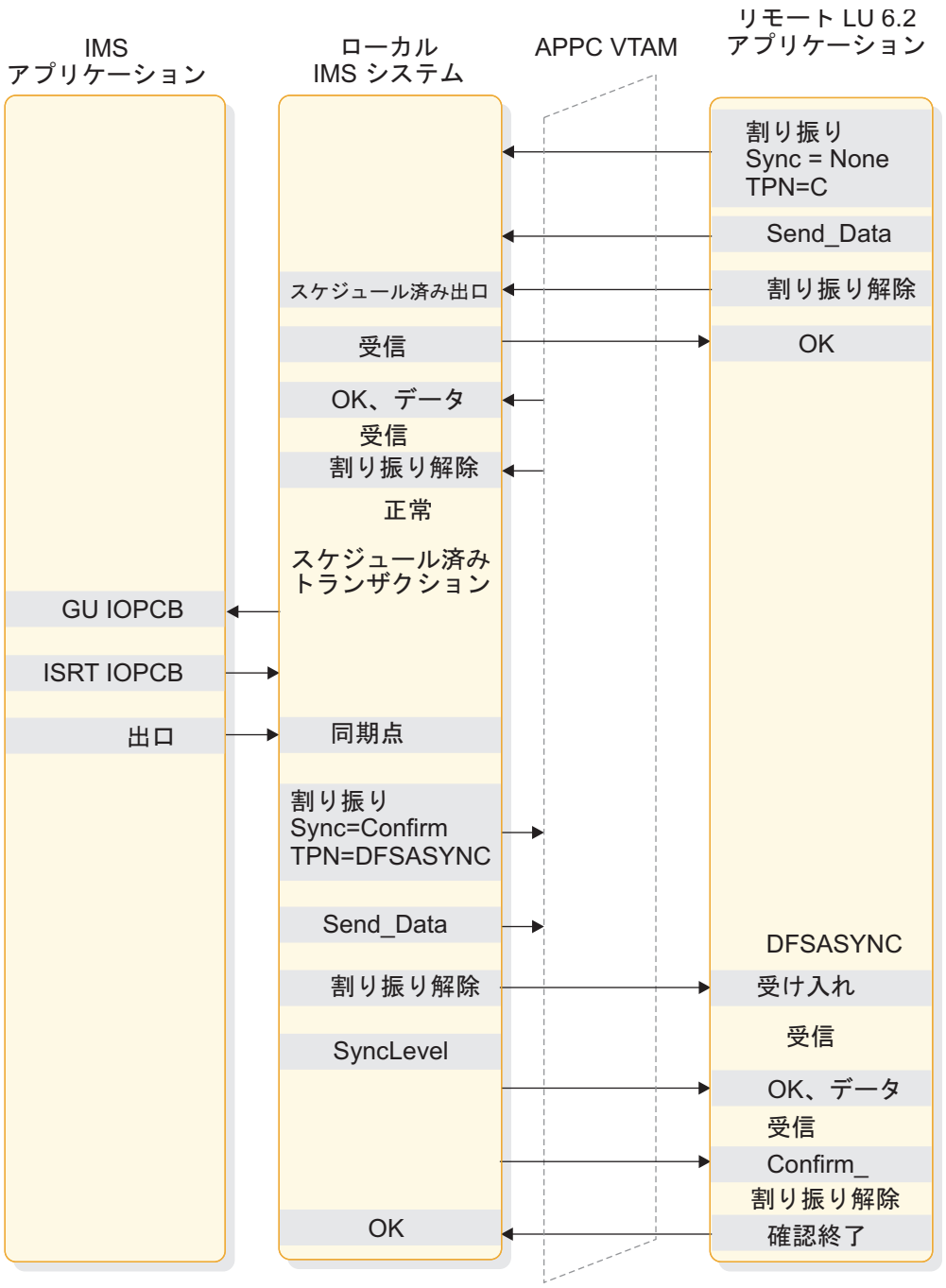


図 30. *Sync_level=None* を指定する場合の、ローカル *IMS* 非同期トランザクションのフロー

146 ページの図 31 は、Sync_level が Confirm である場合のローカル非同期トランザクションのフローを示しています。

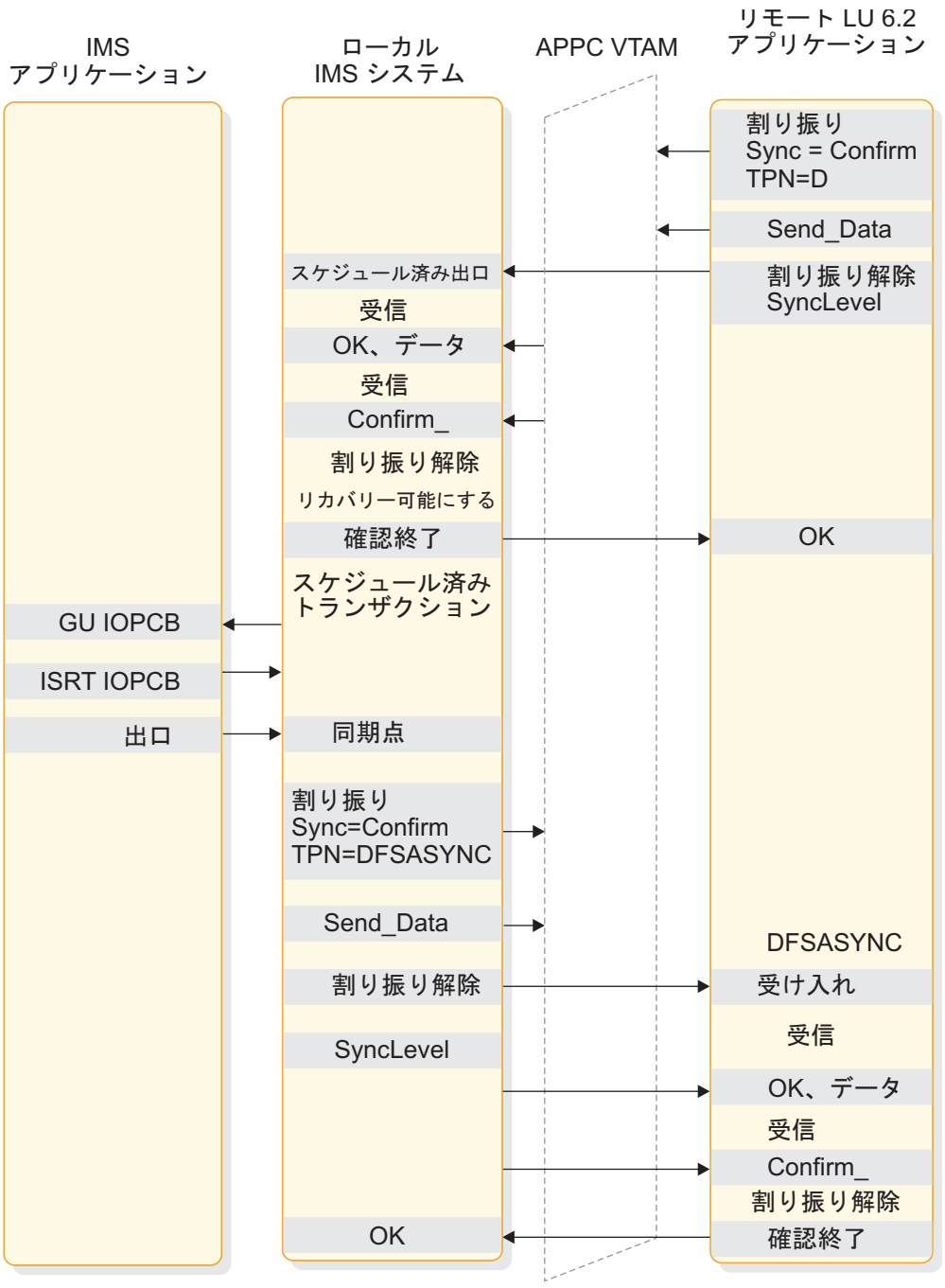


図 *31. Sync_level=Confirm* を指定する場合の、ローカル *IMS* 非同期トランザクションのフロー

次の図は、Sync_level が None である場合の、ローカル会話型トランザクションのフローを示しています。

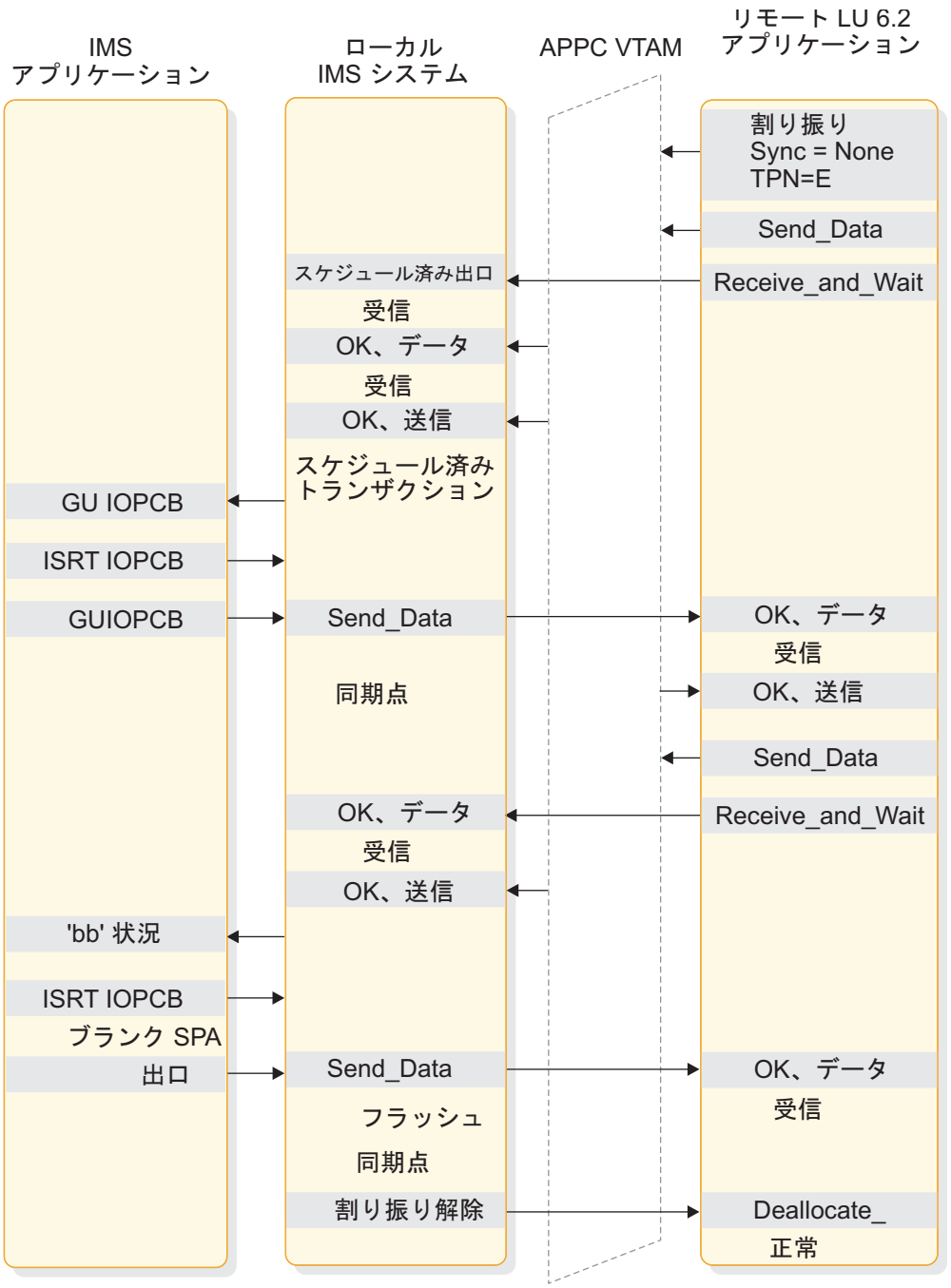


図 32. *Sync_level=None* を指定する場合の、ローカル *IMS* 会話型トランザクションのフロー

次の図は、Sync_level が None である場合の、ローカル IMS コマンドのフローを示しています。

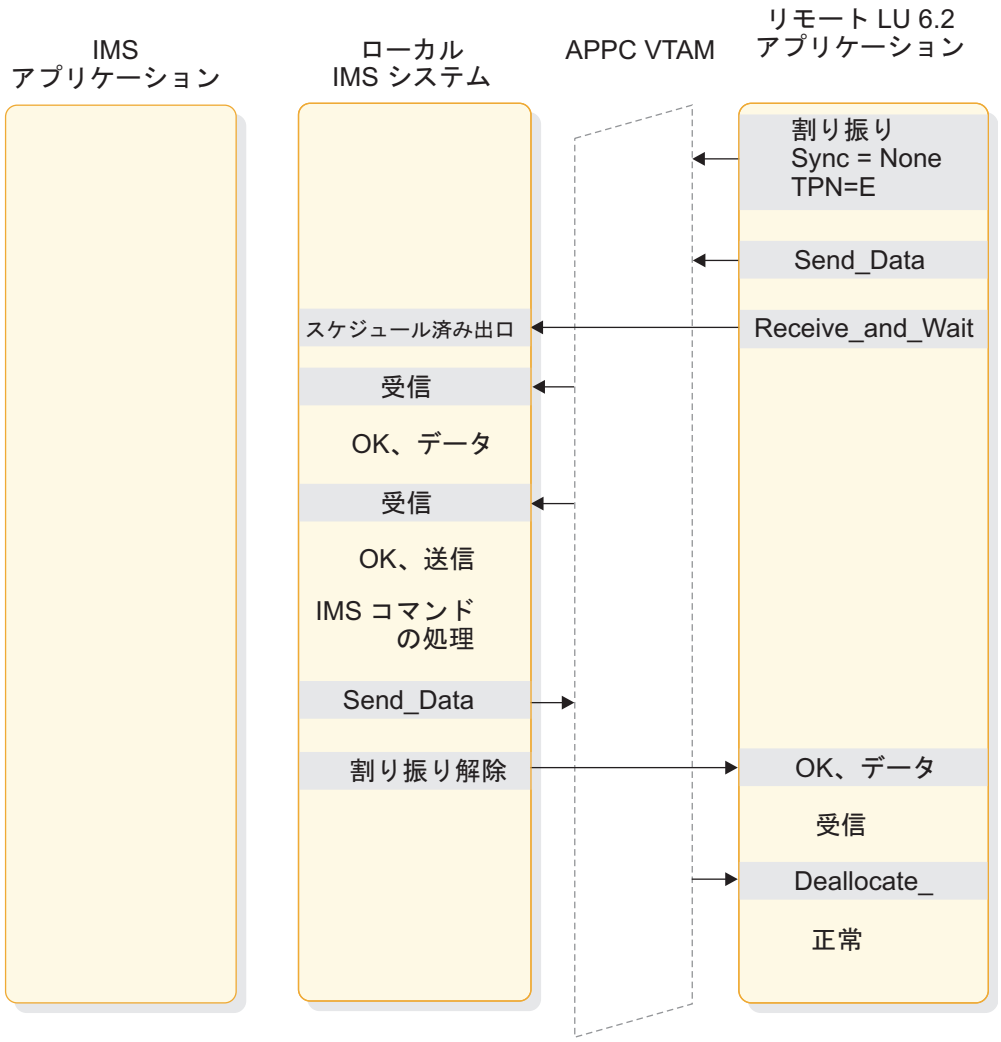


図 33. *Sync_level=None* を指定する場合の、ローカル *IMS* コマンドのフロー

次の図は、Sync_level が Confirm である場合の、ローカル非同期コマンドのフローを示しています。

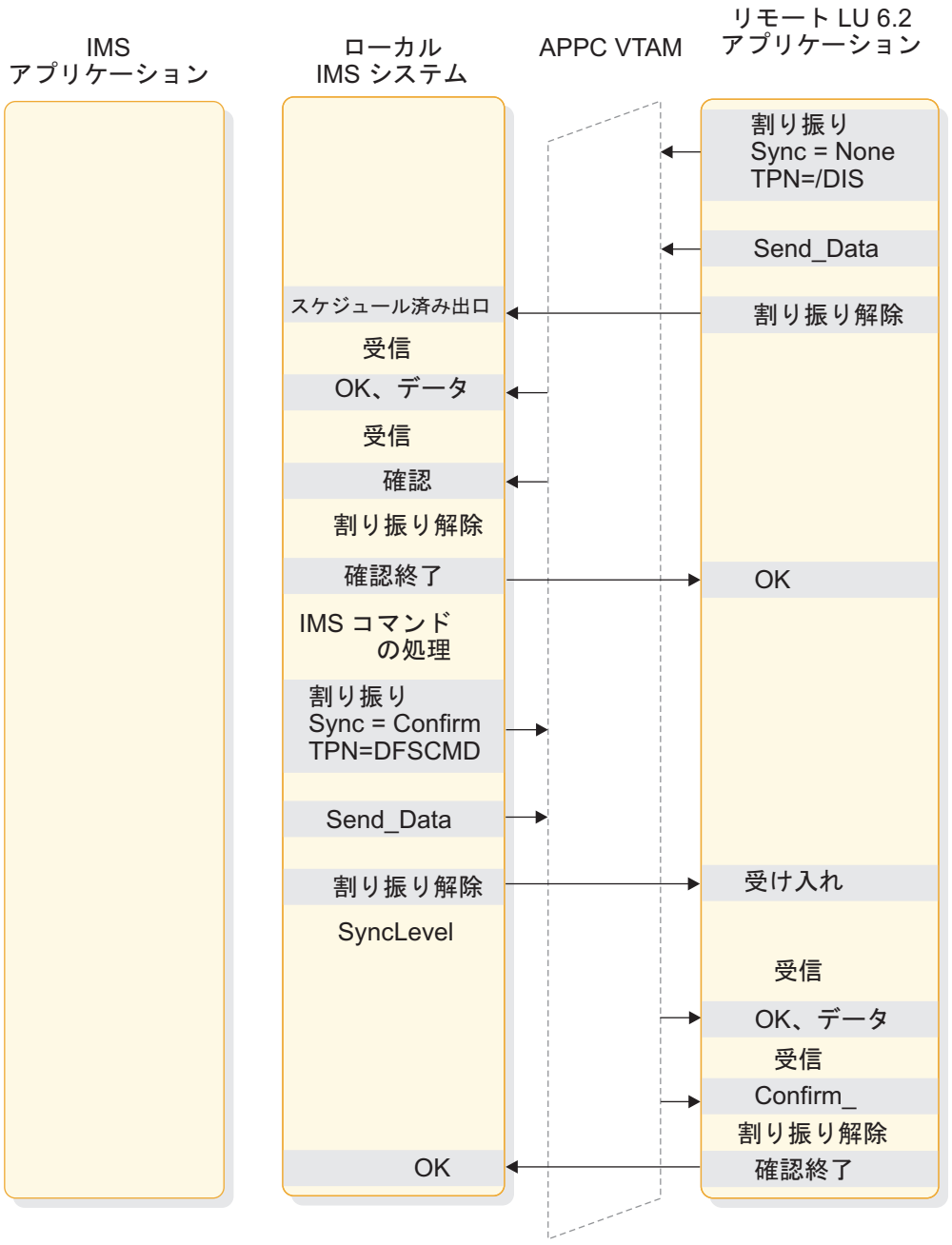


図 *34. Sync_level=Confirm* を指定する場合の、ローカル *IMS* 非同期コマンドのフロー

次の図は、Sync_level が None である場合の、メッセージ通信のフローを示しています。

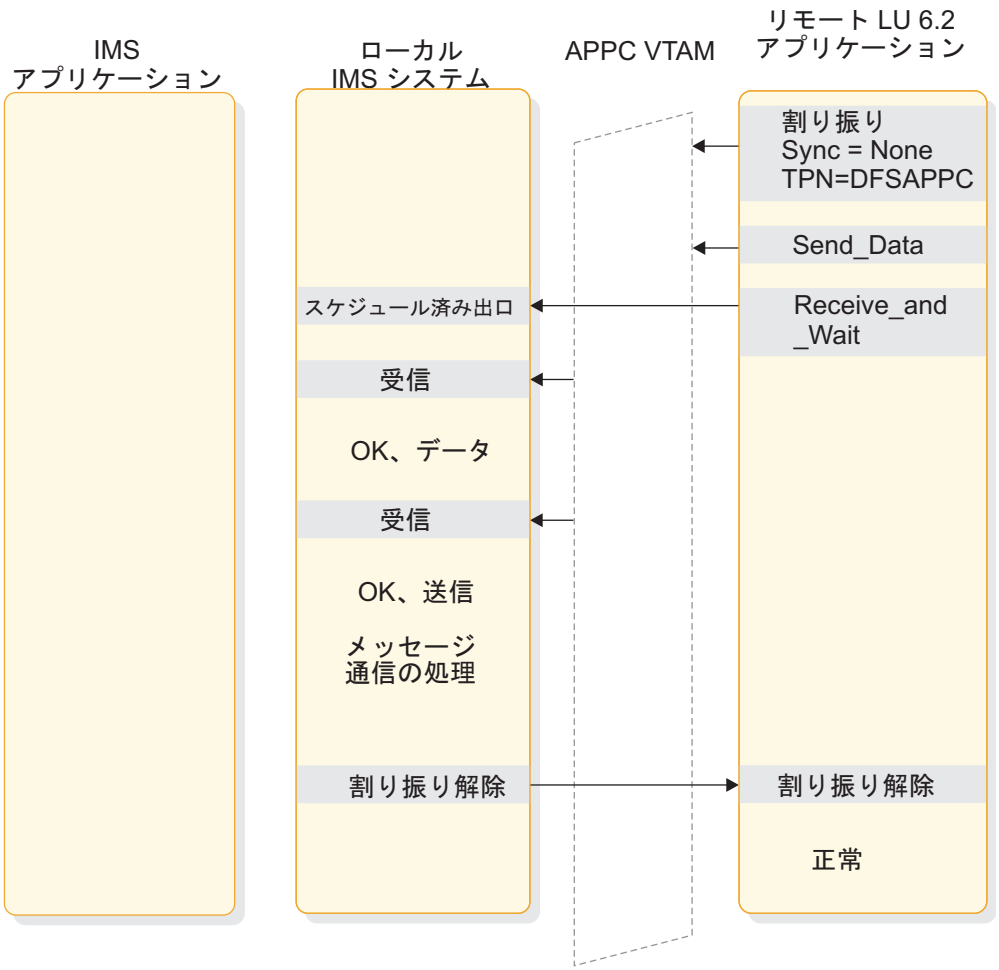


図 *35. Sync_level=None* を指定する場合の、メッセージ通信のフロー

同期を使用して、DFSAPPC の処理中にエラーが発生しなかったかどうかを確認します。エラーが発生していた場合、DEALLOCATE の前にエラー・メッセージが戻されます。

次の図は、Sync_level が None である場合の、CPI-C ドリブン・プログラムのフローを示しています。

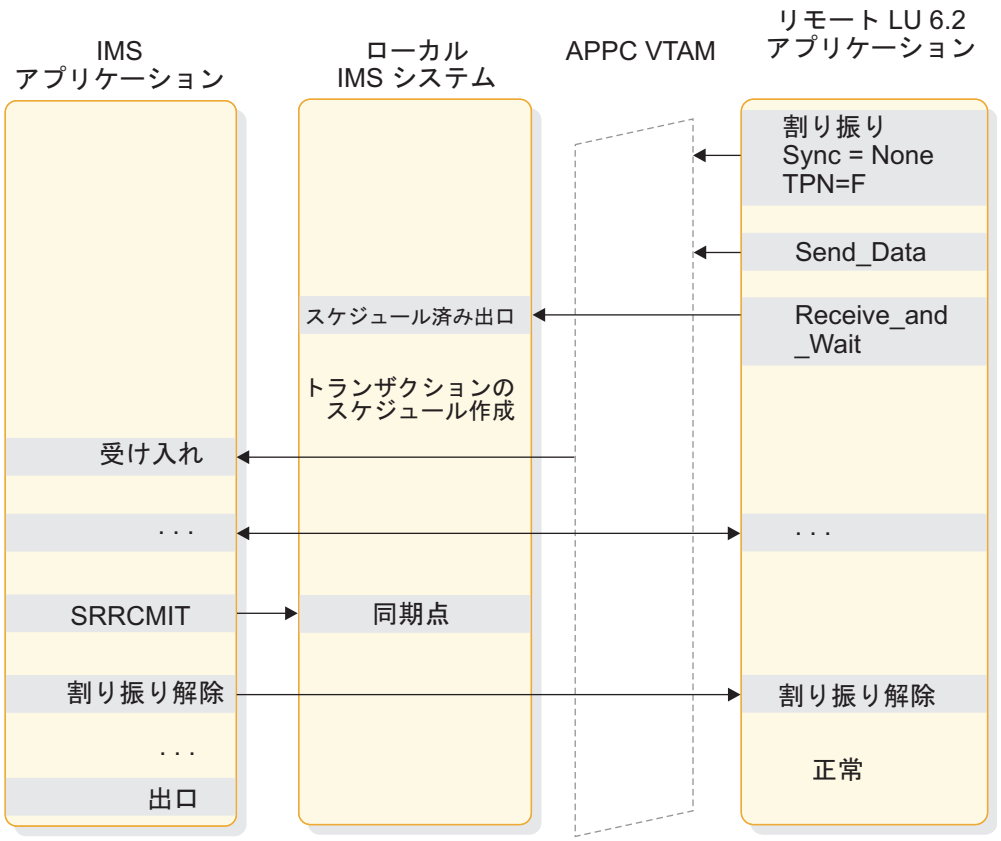


図 *36. Sync_level=None* を指定する場合の、ローカル *CPI* 通信ドリブン・プログラムのフロー

次の図は、Sync_level が None である場合の、リモート同期トランザクションのフローを示しています。

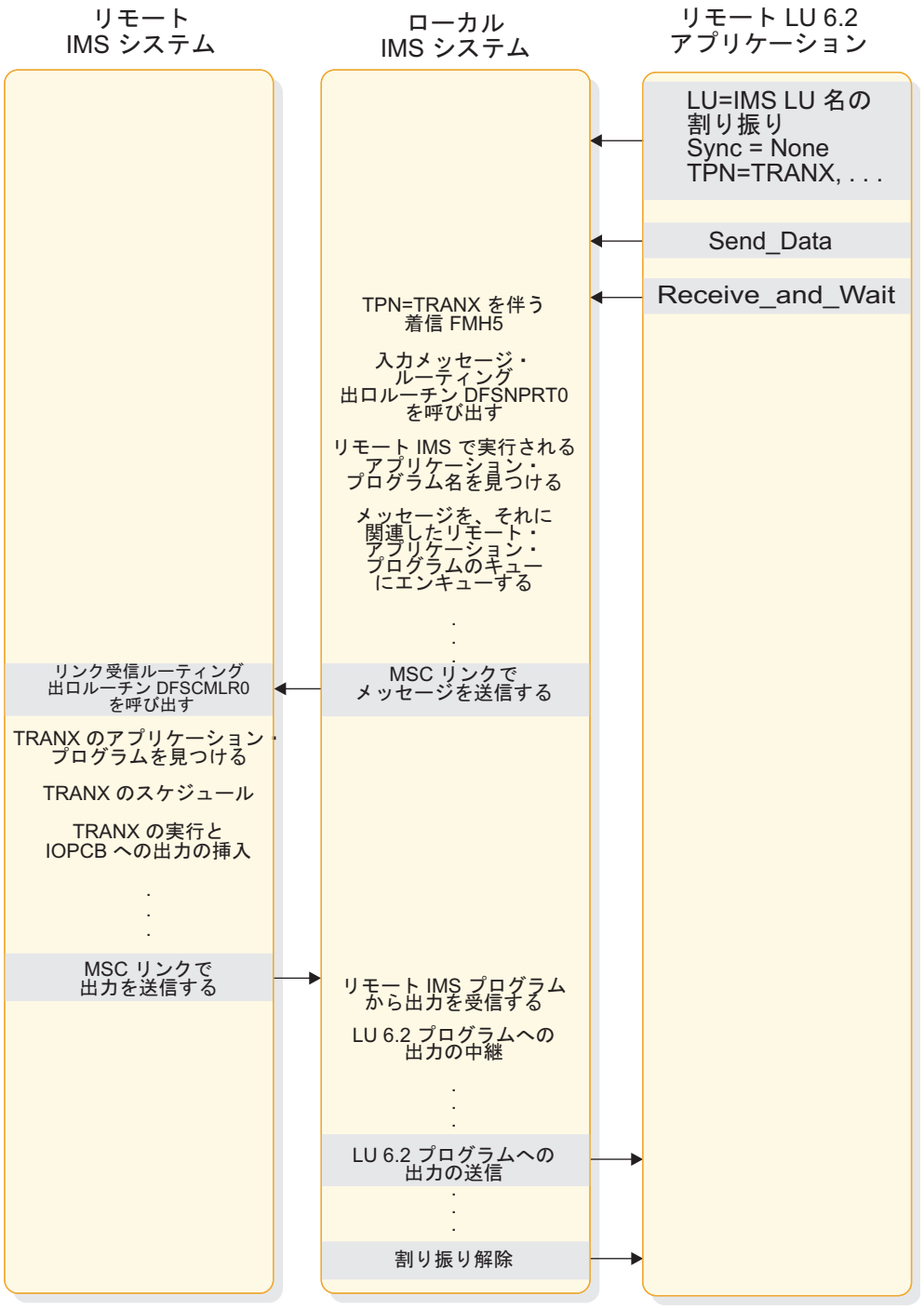


図 37. *Sync_level=None* を指定する場合の、リモート *IMS* 同期トランザクションのフロー

次の図は、Sync_level が None である場合の、リモート非同期トランザクションのフローを示しています。

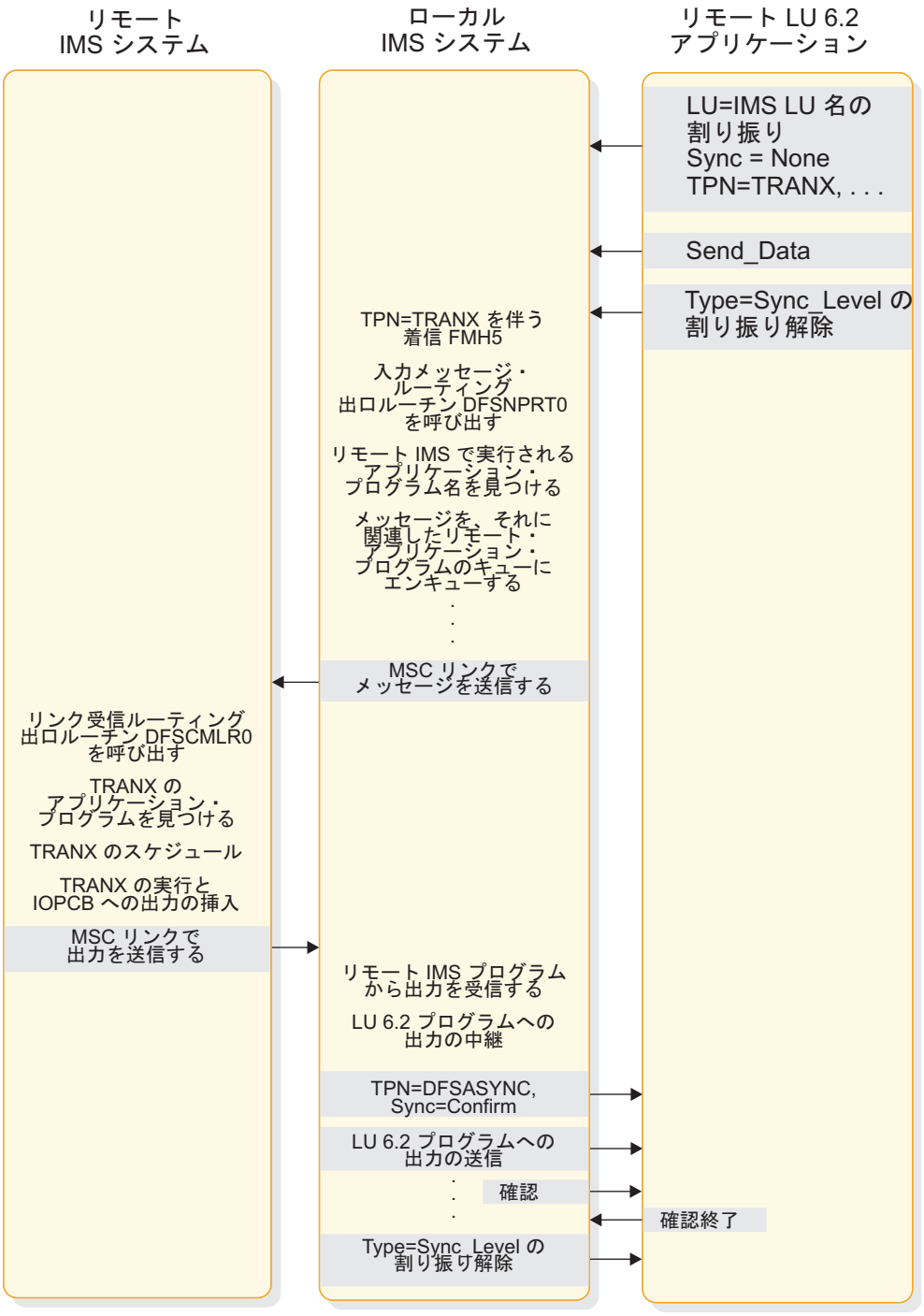


図 38. *Sync_level=None* を指定する場合の、リモート *IMS* 非同期トランザクションのフロー

次の図は、Sync_level が Confirm である場合の、リモート非同期トランザクションのフローを示しています。

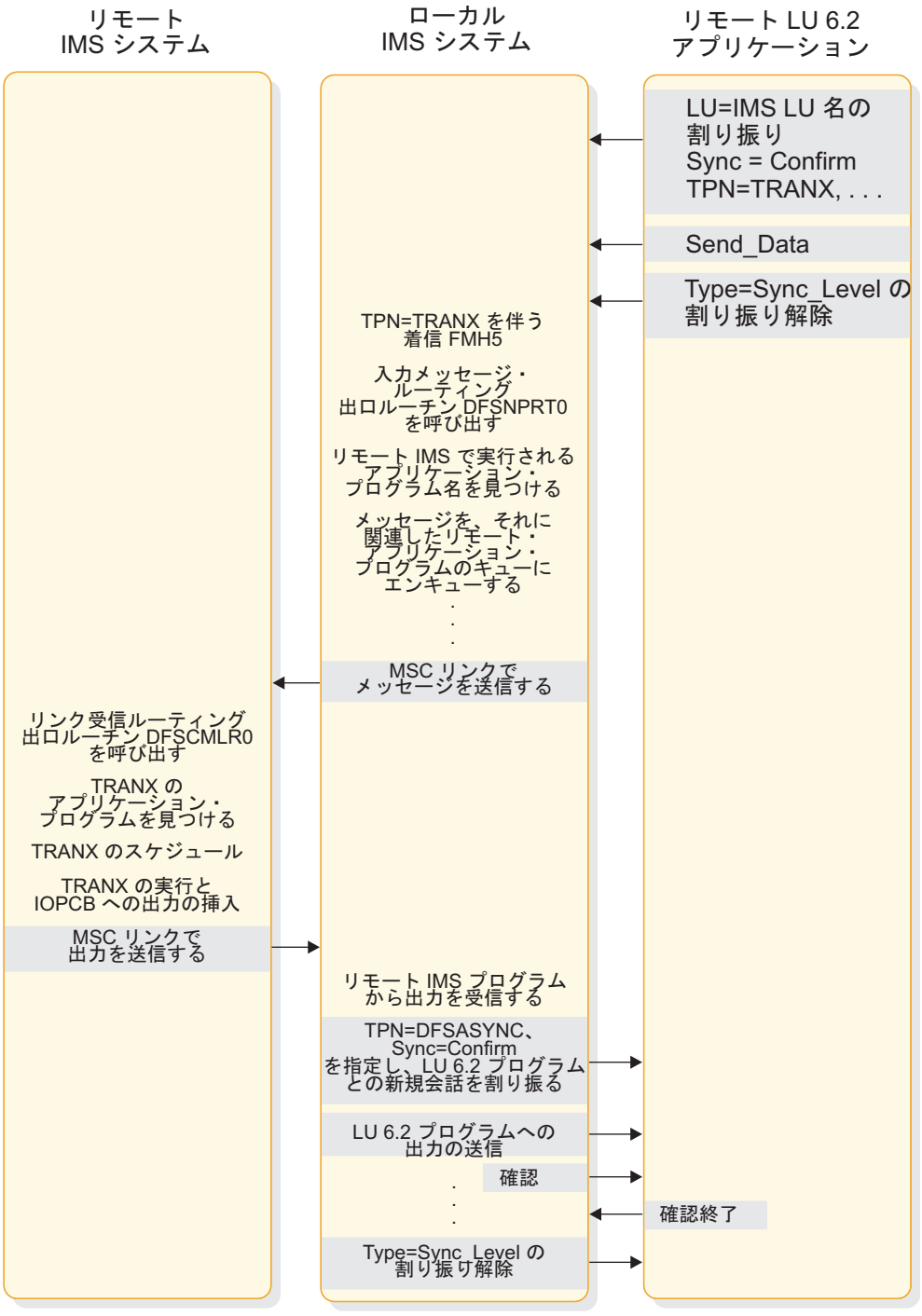


図 *39. Sync_level=Confirm* を指定する場合の、リモート *IMS* 非同期トランザクションのフロー

次の図は、Sync_level が Confirm である場合の、リモート同期トランザクションのフローを示しています。

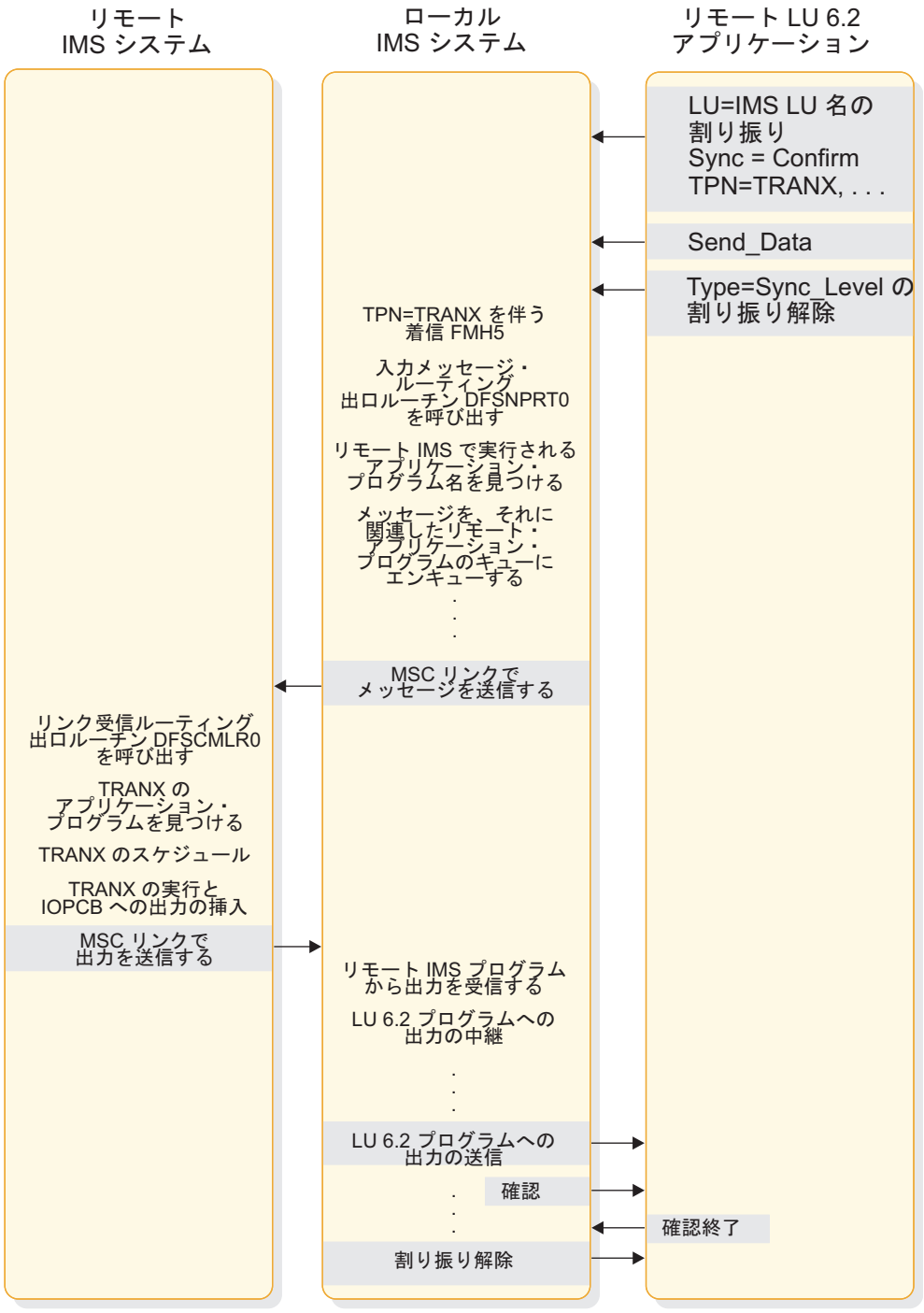


図 *40. Sync_level=Confirm* を指定する場合の、リモート *IMS* 同期トランザクションのフロー

次の図に示すシナリオは、サポートされるアプリケーション・プログラム・タイプでの 2 フェーズ処理の例です。サポートされる関数とコンポーネント間のインターフェースを説明するために、 LU 6.2 verb が使用されています。例に関連するパラメーターだけが表示されています。これは、他のパラメーターがサポートされないことを意味するものではありません。

次の図は、Sync_level=Syncpt を指定する場合の、標準 DL/I プログラム・コミット・シナリオを示しています。

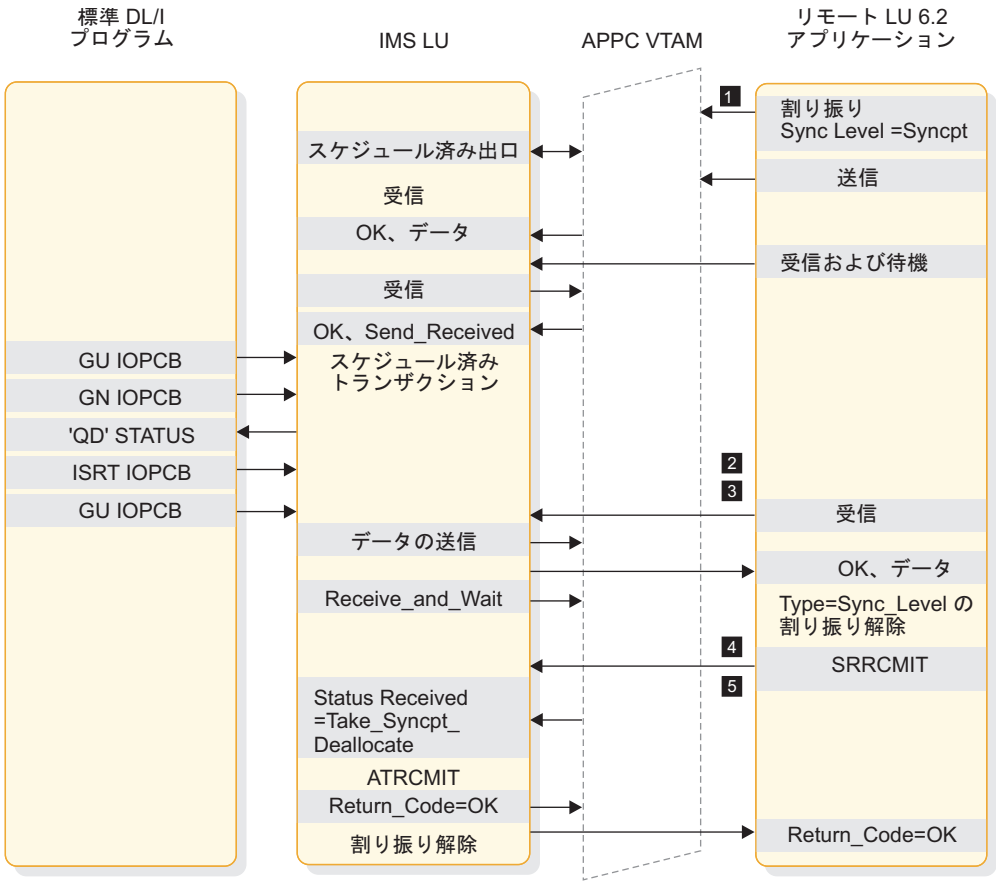


図 *41. Sync_Level=Syncpt* を指定する場合の、標準 *DL/I* プログラム・コミット・シナリオ

注:

1 `Sync_Level=Syncpt` が、保護リソースの更新を起動します。

2 このアプリケーション・プログラムが、リモート・アプリケーションの出力を IMS メッセージ・キューに挿入します。

3 `GU` が、出力の転送を開始します。

4 リモート・アプリケーションが、データ (出力) を受信した後 Confirmed を送信します。

5 IMS が、 `ATRCMIT` (SRRCMIT と同等) を出して、2 フェーズ処理を開始します。

次の図は、Sync_level=Syncpt を指定する場合の、CPI-C ドリブン・コミット・シナリオを示しています。

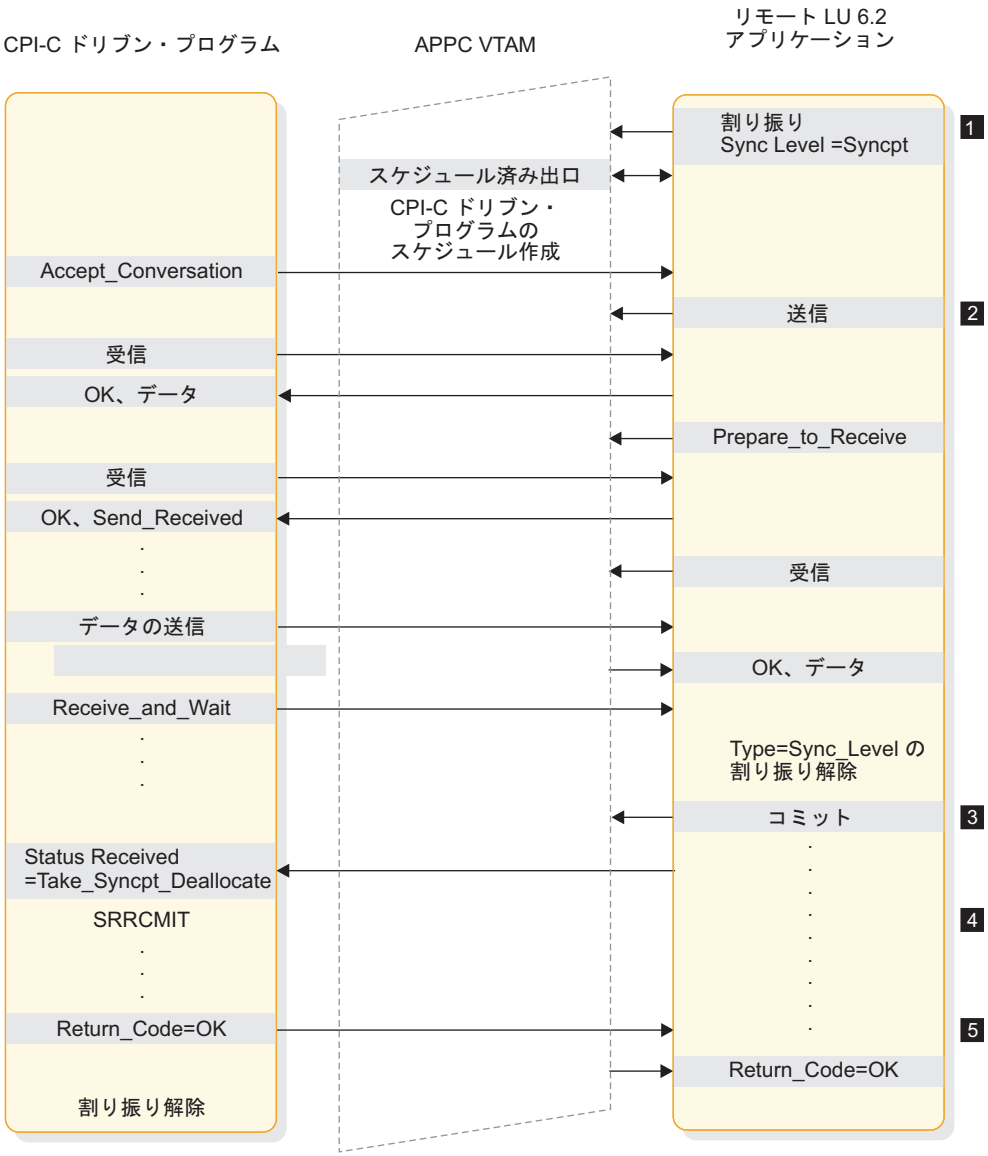


図 42. *Sync_level=Syncpt* を指定する場合の、*CPI-C* ドリブン・コミット・シナリオ

注:

1. `Sync_Level=Syncpt` が、保護リソースの更新を起動します。
2. プログラムが、受信データを送信します。
3. リモート・アプリケーションが、更新のコミットを決定します。
4. CPI-C プログラムが、`SRRCMIT` を出して、その変更をコミットします。
5. コミット戻りコードが、リモート・アプリケーションに戻されます。

次の図は、Sync_level=Syncpt を指定する場合の、標準 DL/I プログラム・バックアウト・シナリオを示しています。

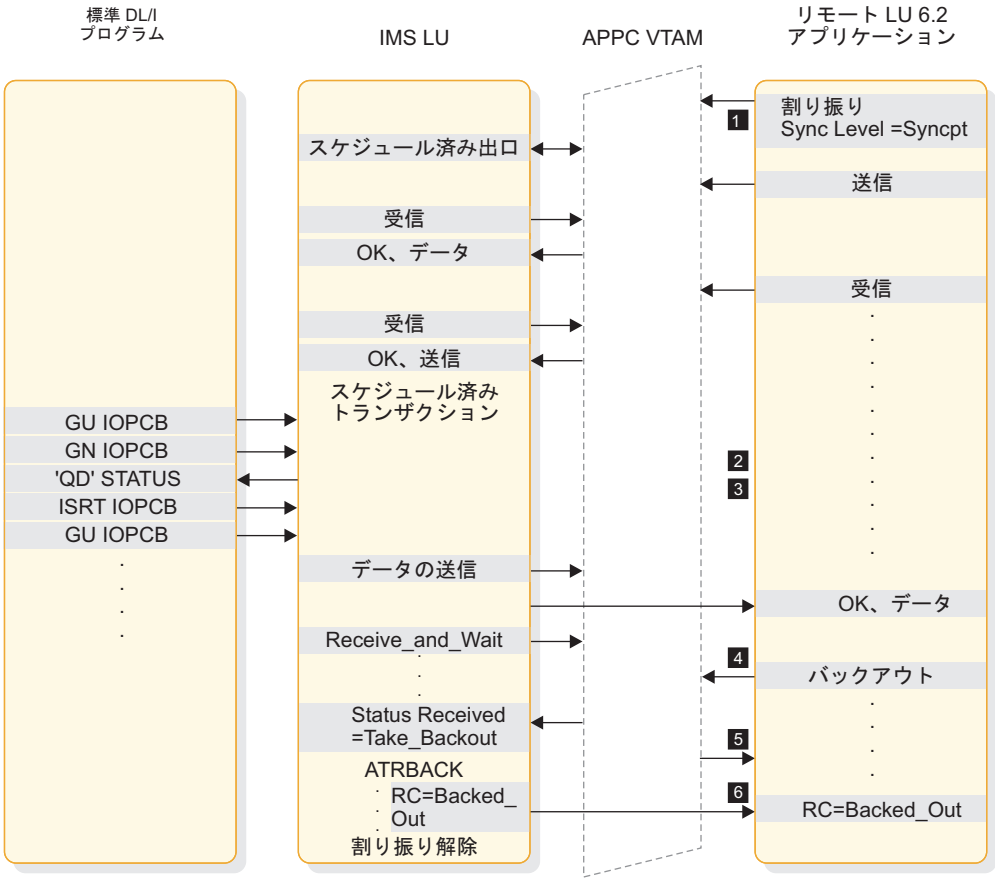


図 *43. Sync_Level=Syncpt* を指定する場合の、標準 *DL/I* プログラム *U119* バックアウト・シナリオ

注:

1 `Sync_Level=Syncpt` が、保護リソースの更新を起動します。

2 このアプリケーション・プログラムが、リモート・アプリケーションの出力を IMS メッセージ・キューに挿入します。

3 `GU` が、出力の転送を開始します。

4 リモート・アプリケーションは、更新のバックアウトを決定します。

5 IMS が、アプリケーションを U119 で異常終了し、アプリケーションをバックアウトします。

6 バックアウト戻りコードが、リモート・アプリケーションに戻されます。

次の図は、Sync_level=Syncpt を指定する場合の、標準 DL/I プログラム・バックアウト・シナリオを示しています。

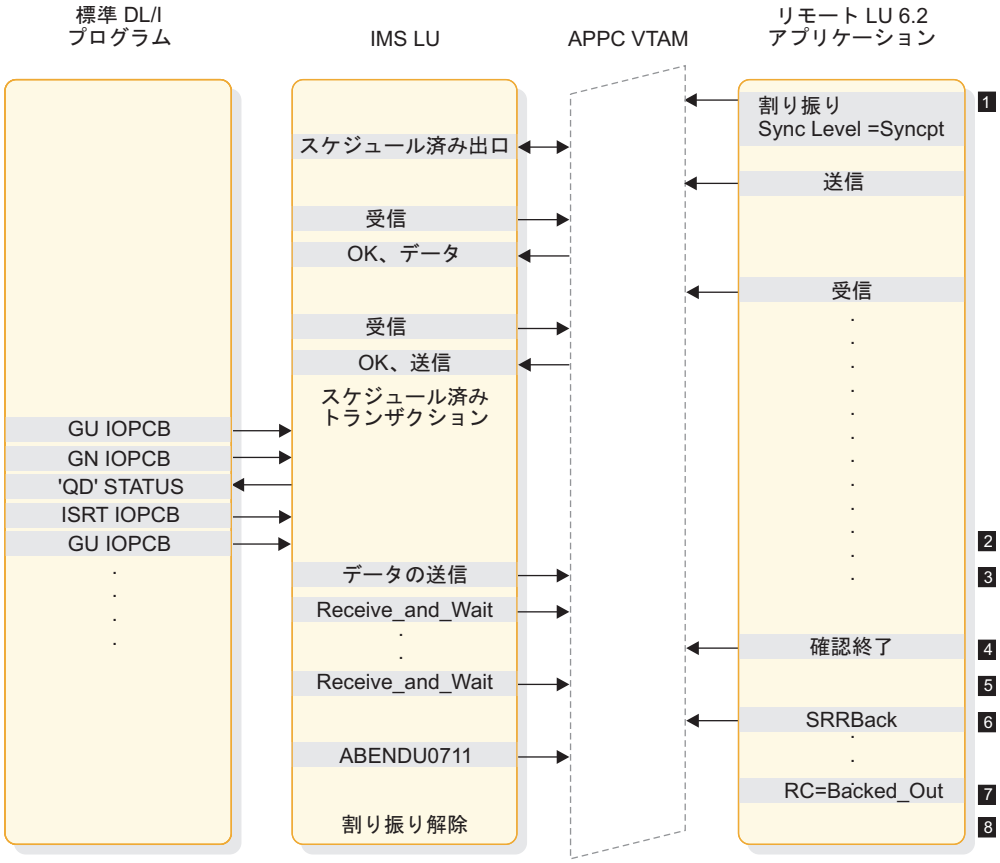


図 44. *Sync_Level=Syncpt* を指定する場合の、標準 *DL/I* プログラム *U0711* バックアウト・シナリオ

注:

1 `Sync_Level=Syncpt` が、保護リソースの更新を起動します。

2 このアプリケーション・プログラムが、リモート・アプリケーションの出力を IMS メッセージ・キューに挿入します。

3 `GU` が、出力の転送を開始します。

4 リモート・アプリケーションが、データ (出力) を受信した後 Confirmed を送信します。

5 IMS が、DL/I アプリケーションの代わりに `ATBRCVW` を出して、コミットまたはバックアウトを待ちます。

6 リモート・アプリケーションが、更新のバックアウトを決定します。

7 IMS が、アプリケーションを U0711 で異常終了し、アプリケーションをバックアウトします。

8 バックアウト戻りコードが、リモート・アプリケーションに戻されます。

次の図は、Sync_level=Syncpt を指定する場合の、標準 DL/I プログラム ROLB シナリオを示しています。

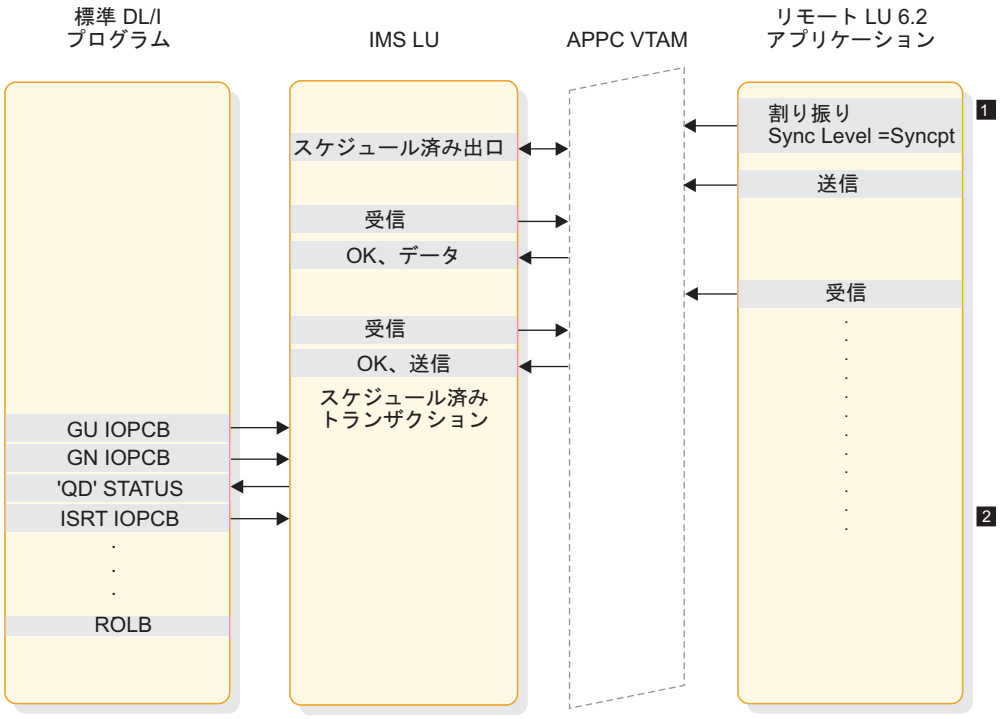


図 *45. Sync_Level=Syncpt* を指定する場合の、標準 *DL/I* プログラム *ROLB* シナリオ

注:

1 `Sync_Level=Syncpt` が、保護リソースの更新を起動します。

2 このアプリケーション・プログラムが、リモート・アプリケーションの出力を IMS メッセージ・キューに挿入します。

次の図は、Sync_Level=Syncpt を指定する場合の、同じコミットの中の複数トランザクションを示しています。

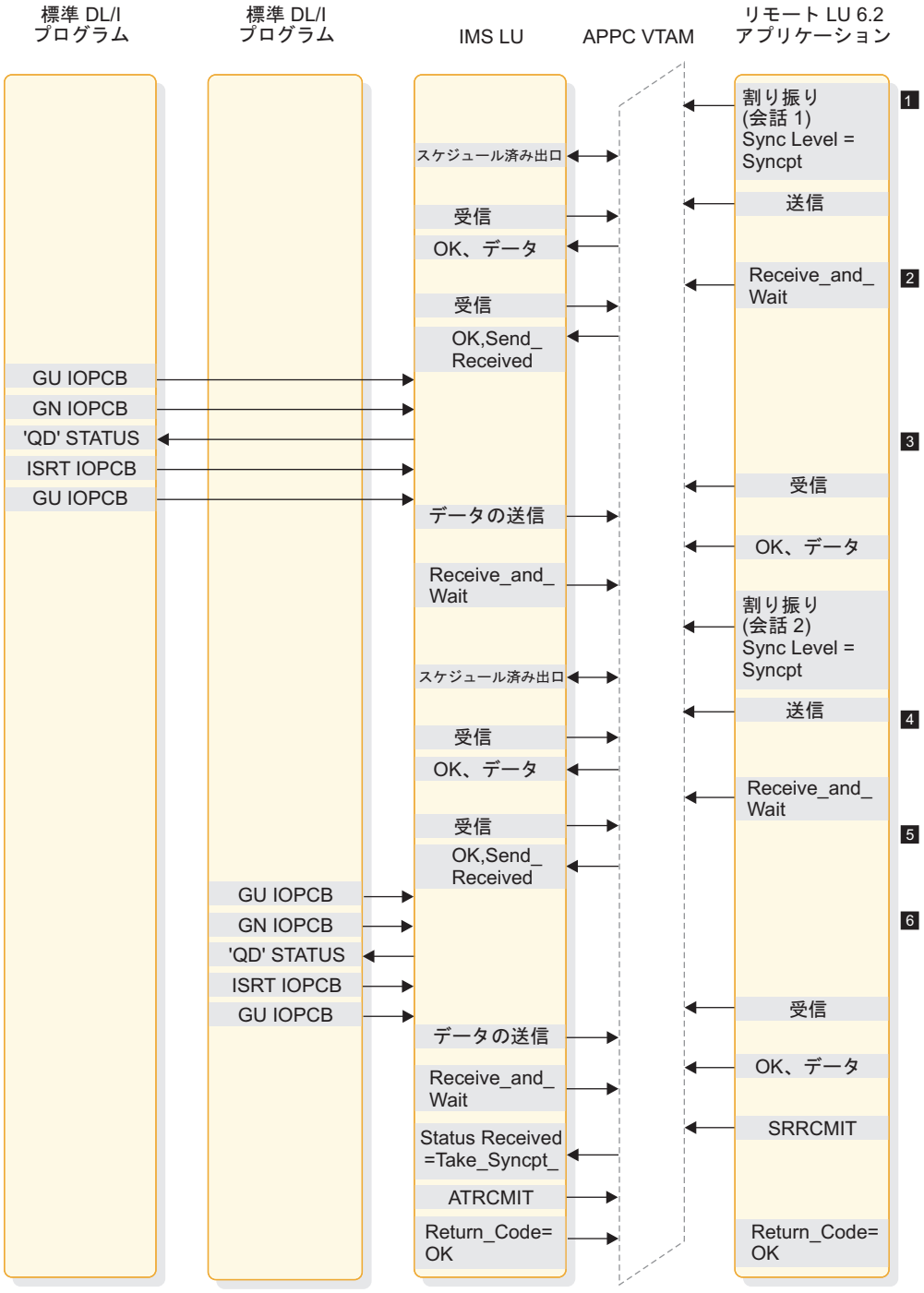


図 *46. Sync_Level=Syncpt* を指定する場合の、同じコミットの中の複数トランザクション

注:

1 `Sync_Level=Syncpt` を指定した割り当てが、会話 1 での保護リソースの更新を起動します。

2 最初のトランザクションが、会話 1 の出力を出します。

3 `Sync_Level=Syncpt` を指定した割り当てが、会話 2 での保護リソースの更新を起動します。

4 2 回目のトランザクションが、会話 2 の出力を出します。

5 リモート・アプリケーションが SRRCMIT を出し、両方のトランザクションをコミットします。

6 IMS が各 DL/I アプリケーションの代わりに ATRCMIT を出して、2 フェーズ処理を開始します。

関連概念:

131 ページの『アプリケーションの目的』

保全性の表

続くいくつかの表では、会話でのメッセージの保全性、保全性で妥協があった場合の処理結果、および IMS が APPC メッセージをリカバリーする方法を示しています。

次の表は、IMS パートナー・システムの視点からの、会話の正常な終了、セッション障害が原因の会話の異常終了、非セッション障害が原因の会話の異常終了の各結果を示しています。これらの結果は、非同期会話と同期会話、および入出力に適用されます。この表は、メッセージの結果と、障害が検出された場合にパートナー・システムがとる処置を示しています。 『LU 6.2 セッション障害』の項の下の処置の例は、プログラマブル・ワークステーション (PWS) 再送です。

表 27. 会話のメッセージ保全性

会話属性	**Normal**	**LU 6.2** セッション障害[1]	その他の障害[2]
同期 Sync_level=NONE	入力: 信頼性あり 出力: 信頼性あり	入力: PWS 再送 出力: PWS 再送	入力: 信頼性あり 出力: 信頼性あり
同期 Sync_level=CONFIRM	入力: 信頼性あり 出力: 信頼性あり	入力: PWS 再送 出力: 信頼性あり	入力: 信頼性あり 出力: 信頼性あり
同期 Sync_level=SYNCPT	入力: 信頼性あり 出力: 信頼性あり	入力: PWS 再送 出力: 信頼性あり	入力: 信頼性あり 出力: 信頼性あり
非同期 Sync_level=NONE	入力: 未確定 出力: 信頼性あり	入力: 検出不能 出力: 信頼性あり	入力: 検出不能 出力: 信頼性あり
非同期 Sync_level=CONFIRM	入力: 信頼性あり 出力: 信頼性あり	入力: PWS 再送 出力: 信頼性あり	入力: 信頼性あり 出力: 信頼性あり
非同期 Sync_level=SYNCPT	入力: 信頼性あり 出力: 信頼性あり	入力: PWS 再送 出力: 信頼性あり	入力: 信頼性あり 出力: 信頼性あり

注:

1. セッション障害 は、ネットワークの接続性の障害です。
2. 非セッション障害 は、その他のすべての種類の障害で、無効なセキュリティー許可などがあります。
3. CONFIRM が失われた場合、IMS は非同期出力を再送します。したがって、PWS は重複出力を許容する必要があります。

次の表は、保全性が危うくなった (メッセージが失われるか、メッセージ状態が未確定) 場合の、処理時間枠の特性を示しています。この表では、各時間枠のオカレンスの相対的確率と、出力が逸失または重複するかどうかを示しています。

値が NONE である Sync_level は、非同期出力には適用されません。IMS はこのような出力には、通常は Sync_level=CONFIRM を使用するからです。

表 *28.* 保全性が危うくなった場合の処理結果

会話属性	トランザクション受け入れ前の時間枠状態[1]	時間枠状態の確率	応答送信中に可能な処置	応答送信中の処置の確率
同期 Sync_level=NONE	ALLOCATE から PREPARE_TO_RECEIVE 戻り	中	脱落あるいは重複出力の送信	中
同期 Sync_level=CONFIRM	PREPARE_TO_RECEIVE から PREPARE_TO_RECEIVE 戻り	低	IMS 受信の CONFIRM。重複出力が生じる可能性あり。	低
同期 Sync_level=SYNCPT	PREPARE_TO_RECEIVE から PREPARE_TO_RECEIVE 戻り	低	IMS 受信の CONFIRM。重複出力が生じる可能性あり。	低
非同期 Sync_level=NONE	割り振りから割り振り解除	高	IMS 受信の CONFIRMED。 重複出力が生じる可能性あり。	低
非同期 Sync_level=CONFIRM	PREPARE_TO_RECEIVE から PREPARE_TO_RECEIVE 戻り	低[2]	IMS 受信の CONFIRMED。 重複出力が生じる可能性あり。	低
非同期 Sync_level=SYNCPT	PREPARE_TO_RECEIVE から PREPARE_TO_RECEIVE 戻り	低[2]	IMS 受信の CONFIRMED。 重複出力が生じる可能性あり。	

注:

1. 時間枠 という用語は、特定のイベント (この表に記載している結果など) が発生する可能性がある期間を指します。
2. 回復可能です。

次の表では、IMS ウォーム・スタート、XRF テークオーバー、APPC セッション障害、および MSC リンク障害において、 IMS が APPC トランザクションをリカバリーする方法を示しています。

表 *29. APPC* メッセージのリカバリー

メッセージ・タイプ	**IMS** ウォーム・スタート **(NRE** または **ERE)**	**XRF** テークオーバー	**APPC (LU 6.2)** セッション障害	**MSC** リンク障害
ローカル・リカバリー可能トランザクション、非応答、非会話 - APPC 同期会話モード - APPC 非同期会話モード	廃棄 (2) リカバリー	廃棄 (4) リカバリー	廃棄 (6) リカバリー (1)	該当せず (9) 該当せず (9)
ローカル・リカバリー可能トランザクション、会話または応答モード - APPC 同期会話モード - APPC 非同期会話モード	廃棄 (2) 該当せず (8)	廃棄 (4) 該当せず (8)	廃棄 (6) 該当せず (8)	該当せず (9) 該当せず (8,9)

表 29. APPC メッセージのリカバリー (続き)

メッセージ・タイプ	IMS ウォーム・スタート (NRE または ERE)	XRF テークオーバー	APPC (LU 6.2) セッション障害	MSC リンク障害
ローカル・リカバリー不能トランザクション、- APPC 同期会話モード - APPC 非同期会話モード	廃棄 (2) 廃棄 (2)	 廃棄 (4)	廃棄 (6) リカバリー (1)	該当せず (9) 該当せず (9)
リモート・リカバリー可能トランザクション、非応答、非会話 - APPC 同期会話モード - APPC 非同期会話モード	廃棄 (2,5) リカバリー	廃棄 (3,5) リカバリー	リカバリー (1) リカバリー (1)	リカバリー (7) リカバリー (7)
リモート・リカバリー可能トランザクション、会話または応答モード - APPC 同期会話モード - APPC 非同期会話モード	廃棄 (2,5) 該当せず (8)	廃棄 (3,5) 該当せず (8)	リカバリー (1) 該当せず (8)	リカバリー (7) 該当せず (8)
リモート・リカバリー不能トランザクション、- APPC 同期会話モード - APPC 非同期会話モード	廃棄 (2,5) 廃棄 (2,5)	廃棄 (3,5) 廃棄 (3,5)	リカバリー (1) リカバリー (1)	リカバリー (7) リカバリー (7)

注:

1. このリカバリーのシナリオは、メッセージが障害以前に、エンキューされていたと想定しています。そうでない場合にはメッセージは廃棄されます。
2. IMS ウォーム・スタート処理中に、メッセージは廃棄されます。
3. MSC リンクが再始動されたとき、およびメッセージがキューから取り除かれた (リンクにまたがって上を送信するため) ときには、メッセージは廃棄されます。
4. メッセージ領域が開始されたとき、およびメッセージがキューから取り除かれたとき (アプリケーション・プログラムが処理するため) には、メッセージは廃棄されます。
5. すべてのリモート MSC APPC トランザクションでは、障害がローカル IMS 内で起きたときに、メッセージが既に MSC リンクにまたがってリモート・システムに送信されている場合には、そのメッセージは処理されます。リモート・アプリケーション・プログラムがメッセージを処理して、応答メッセージをローカル・システムに送り返した後は、元のトランザクションを実行要求した、 LU 6.2 装置またはプログラムの DFSASYNC TP 名のところに、メッセージがエンキューされます。
6. 同期点で、ユーザー・メッセージ制御エラー出口ルーチン (DFSCMUX0) によって、トランザクションの異常終了が防止され、出力メッセージは転送 (リカバリー) されます。

   出口ルーチンの詳細については、「*IMS V15* 出口ルーチン」を参照してください。
7. 標準 MSC リンク・リカバリー・プロトコルは、キューイングされているメッセージ、あるいはリンク障害時に MSC リンクにまたがって送信処理中のメッセージを、すべてリカバリーします。
8. IMS 会話型モードおよび応答モードのトランザクションは、 APPC 非同期会話セッションから実行要求することができません。 APPC 同期会話モードを使用する必要があります。
9. MSC リンク障害は、ローカル・トランザクションに影響を与えません。

DFSAPPC メッセージ通信

DFSAPPC は、IMS システム・サービスを行う LU 6.2 記述子です。

これによって、LU 6.2 アプリケーション・プログラムは、次のものにメッセージを送信できます。

- アプリケーション・プログラム (トランザクション)

- IMS 管理のローカルまたはリモート LTERM (メッセージ通信)
- LU 名および TP 名

LTERM= オプションを使用して送信されたメッセージは、IMS が管理しているローカルまたはリモート LTERM へ送られます。LTERM= オプションを使用しないで送信されたメッセージは、適切な LU 6.2 アプリケーションまたは IMS アプリケーション・プログラムへ送信されます。

LTERM は LU 6.2 記述子名である可能性があります。その結果、LU 6.2 装置が明示的に選択されたのと同じように、メッセージが LU 6.2 アプリケーション・プログラムに送信されます。

DFSAPPC を使用する場合、メッセージは非同期で配信されます。 メッセージが割り振られてその割り振りが失敗した場合、正常に送達が行われるまでメッセージは IMS メッセージ・キューに保留されます。

例: LU 6.2 会話の例では、IMS アプリケーションは、LU 名を FRED、TPN 名を REPORT と指定して、そのパートナーに対して DFSAPPC メッセージ通信を出します。REPI はユーザーのデータです。

```
DFSAPPC  (TPN=REPORT LU=FRED) REP1
```

LU= フィールドに 17 バイトのネットワーク修飾名を使用することができます。

制約事項: LU 6.2 アーキテクチャーは、LU 6.2 会話の CHNG 呼び出しで、ALTRESP PCB の使用を禁止しています。LU 6.2 会話が対応できるのは、IOPCB だけです。アプリケーションは、既存の LU 6.2 会話 (同期) でメッセージを送信するか、あるいは IMS が IOPCB を使用して、新規の会話 (非同期) を作成します。 LU 6.2 会話に関連した LTERM がないので、IOPCB だけが元の LU 6.2 会話を表します。

関連資料: DFSAPPC について詳しくは、「*IMS V15* コミュニケーションおよびコネクション」を参照してください。

第 8 章 IMS アプリケーション・プログラムのテスト

IMS アプリケーション・プログラムに対してプログラム単体テストを実行し、入力データ、処理、出力データをプログラムが正しく処理することを確かめる必要があります。実行するテストの量およびタイプは、個々のプログラムによって異なります。

IMS プログラムのテストに関する推奨事項

プログラムのテストを開始する前に、確立されたテスト手順を知っている必要があります。

テストを開始するには、以下の 3 項目が必要です。

- テスト JCL
- テスト・データベース。実動データベースを使用してプログラムのテストを決して行わないでください。失敗すると、有効なデータが損傷する可能性があるからです。
- テスト入力データ。使用する入力データは現行のものである必要はありませんが、有効なデータでなければなりません。有効な入力データを使用しない限り、出力データが有効であることが確認できません。

プログラム・テストの目的は、起こる可能性のあるあらゆる状態を、プログラムが正しく処理できるかどうか確認することです。徹底的にプログラムをテストするには、プログラムがとりうるできるだけたくさんの経路をテストしてください。

推奨事項:

- プログラムに各分岐を実行させる入力データを使用して、プログラムの各経路をテストしてください。
- プログラムがエラー・ルーチンをテストしていることを確かめてください。 プログラムにできるだけ多くのエラー条件をテストさせる入力データを使用してください。
- プログラムで使用している編集ルーチンをテストしてください。 プログラムにできるだけたくさんの異なるデータの組み合わせを与えて、入力データが正しく編集されていることを確認してください。

IMS プログラムのテストに先立つ DL/I 呼び出しシーケンスのテスト (DFSDDLT0)

DL/I テスト・プログラム DFSDDLT0 は、任意のデータベースに対して、指定した DL/I 呼び出しを実行する IMS アプリケーション・プログラムです。

制約事項: コーディネーター・コントローラー (CCTL) を使用している場合、DFSDDLT0 は機能しません。

DFSDDLT0 を使用する利点は、プログラムをコーディングする前に使用する、DL/I 呼び出しシーケンスのテストが可能であるという点です。プログラムをテストする前に、DL/I 呼び出しのシーケンスをテストすると、デバッグが容易になります。プログラムをテストする前に DL/I 呼び出しが正しいかどうかわかるからです。プログラムをテストしても正しく実行されなかった場合、既に DFSDDLT0 を使用して DL/I 呼び出しをテストした後であれば、 DL/I 呼び出しが問題の一部ではないことが分かります。

テストしたい DL/I 呼び出しのそれぞれについて、呼び出しおよびその呼び出しとともに使用している SSA を DFSDDLT0 に指定します。次に DFSDDLT0 を実行すると、呼び出しの結果が得られます。各呼び出しの後、DFSDDLT0 は DB PCB マスクおよび入出力域の内容を示します。これは、各呼び出しについて、セグメントに定義したアクセス・パスや呼び出しの結果を、DFSDDLT0 が検査することを表しています。 DFSDDLT0 は IMS アプリケーション制御ブロックを表示できるので、デバッグの際に役立ちます。

実行したい呼び出しを DFSDDLT0 に指示するには、4 つのタイプの制御ステートメントを使用します。

状況ステートメントは、DFSDDLT0 の出力用に印刷オプションを確立します。また、指定した呼び出しに使用する DB PCB を選択します。

コメント・ステートメントを使用すると、コメントを与えたいかどうか選択できます。

呼び出しステートメントは、実行したい呼び出し、その呼び出しとともに使用したい SSA、および呼び出しを何回実行したいか、DFSDDLT0 に指示します。

比較ステートメントは、呼び出し実行後の結果とユーザーが準備した結果を比較したいことを DFSDDLT0 に指示します。

呼び出しシーケンスをテストして正しく稼働しているか調べる他に、DFSDDLT0 を使用して呼び出しシーケンスのパフォーマンスを検査することもできます。

BTS を使用した IMS プログラムのテスト

IMS Batch Terminal Simulator for z/OS (BTS) は、呼び出しシーケンスのテストに使用できるので、プログラムをテストするには有効なツールです。BTS が作成する資料はデバッグに役立ちます。また、オンライン・アプリケーション・プログラムをオンラインで稼働しないで、テストすることができます。

制約事項: CCTL を使用していたり、DBCTL 下で稼働していたりする場合には、BTS は機能しません。

関連資料: BTS の使用方法については、「*IMS Batch Terminal Simulator for z/OS*ユーザーズ・ガイド」を参照してください。

イメージ・キャプチャーを使用する IMS プログラム用の DL/I 呼び出しトレース

DL/I イメージ・キャプチャー・プログラム (DFSDLTR0) は、すべてのタイプの IMS アプリケーション・プログラムが出す DL/I 呼び出しをトレースし、記録できるトレース・プログラムです。

制約事項**:** イメージ・キャプチャー・プログラムは、高速機能データベースへの呼び出しはトレースしません。

DB/DC あるいはバッチ環境内で、イメージ・キャプチャー・プログラムを実行して、以下のことが可能です。

- プログラムのテスト

  イメージ・キャプチャー・プログラムは、トレースしている呼び出しでエラーを検出すると、できるだけ多くの呼び出しを複製します。 ただし、エラーが発生した場所を文書化することはできず、また常にすべての SSA を複製できるとは限りません。

- **DFSDDLT0** 用入力の作成

  イメージ・キャプチャー・プログラムが作成した出力を、 DFSDDLT0 への入力として使用することができます。イメージ・キャプチャー・プログラムは、状況ステートメント、コメント・ステートメント、呼び出しステートメント、および比較ステートメントを、DFSDDLT0 用に作成します。

- プログラムのデバッグ

  プログラムが異常終了する場合、イメージ・キャプチャー・プログラムを使用してプログラムを再実行できます。イメージ・キャプチャー・プログラムは、プログラム障害を引き起こす条件を複製し、文書化します。イメージ・キャプチャー・プログラムが作成した報告書の情報を使用して、問題を検出して修正することができます。

イメージ・キャプチャーの DFSDDLT0 との併用

イメージ・キャプチャー・プログラムは、DFSDDLT0 への入力として使用できる次の制御ステートメントを作成します。

- 状況ステートメント

  イメージ・キャプチャー・プログラムを呼び出すと、状況ステートメントが作成されます。状況ステートメントは、以下のことを行います。

  – 印刷オプションを設定します。これにより DFSDDLT0 は、すべての呼び出しトレース・コメント、すべての DL/I 呼び出し、およびすべての比較の結果を印刷します。
  – アプリケーション・プログラムの実行中に PCB に変更が発生するたびに、新規の相対 PCB 番号を判別します。

- コメント・ステートメント

  イメージ・キャプチャー・プログラムを呼び出すと、コメント・ステートメントも作成されます。コメント・ステートメントから以下のものが得られます。

- IMS がトレースを開始した時刻および日付
- トレースされている PSB の名前

またイメージ・キャプチャー・プログラムは、IMS によるエラー検出の対象となるどの呼び出しよりも前に、コメント・ステートメントを作成します。

- 呼び出しステートメント

  イメージ・キャプチャー・プログラムは、アプリケーション・プログラムが出す DL/I 呼び出しのそれぞれについて、呼び出しステートメントを作成します。また、トレースを開始する際、および各コミット・ポイントあるいは CHKP 要求の後で、CHKP 呼び出しを生成します。

- 比較ステートメント

  COMP を TRACE コマンド (イメージ・キャプチャー・プログラムをオンラインで実行時) に指定、あるいは DLITRACE 制御ステートメント (イメージ・キャプチャー・プログラムをバッチ・ジョブとして実行時) に指定する場合、イメージ・キャプチャー・プログラムは、データおよび PCB 比較ステートメントを作成します。

イメージ・キャプチャー出力を使用する際の制約事項

イメージ・キャプチャー呼び出しの状況ステートメントは、相対 PCB 位置に基づくものです。

PCB パラメーター LIST=NO が指定された場合、以下の示すとおり、PCB を選択するように状況ステートメントを変更することが必要な場合があります。

- すべての PCB のパラメーターが LIST=YES である場合、状況ステートメントを変更する必要はありません。
- すべての PCB のパラメーターが LIST=NO である場合、状況ステートメントは、相対 PCB 番号から正確な PCB 名に変更する必要があります。
- いくつかの PCB にパラメーター LIST=YES があり、他の PCB にはパラメーター LIST=NO がある場合、状況ステートメントは、以下のとおり変更する必要があります。
  - PCB 相対位置は、LIST=YES と見なされたすべての PCB に基づきます。
  - PCB 名がある PCB について、相対 PCB 番号に基づく PCB 名を使用するように状況ステートメントを変更できます。
  - LIST=YES であるが PCB 名がない PCB では、相対 PCB 番号を変更して、LIST=YES および LIST=NO を使用している PCB リストを探し、ユーザー・リストの相対 PCB 番号を参照してください。

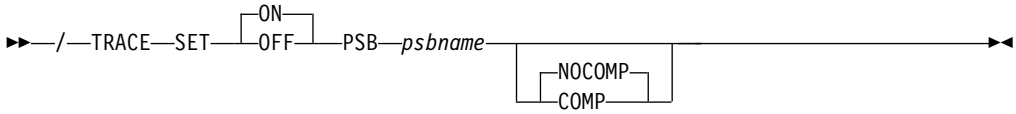
オンラインでのイメージ・キャプチャーの実行

オンラインでイメージ・キャプチャー・プログラムを実行する場合、トレース出力は IMS ログ・データ・セットに送られます。オンラインでイメージ・キャプチャー・プログラムを実行するには、 IMS マスター端末から IMS TRACE コマンドを出してください。

BMP あるいは MPP をトレースし、 DFSDDLT0 にトレース結果を使用しようとしている場合、 BMP あるいは MPP には、処理を行っているデータベースへの、

排他的書き込みアクセスがなければなりません。アプリケーション・プログラムに排他的アクセスがない場合、 DFSDDLT0 の結果はアプリケーション・プログラムの結果と異なってしまう可能性があります。 GSAM データベースにアクセスする BMP をトレースする場合、 //IMSERR DD ステートメントを組み込んで、GSAM 制御ブロックの定様式ダンプを獲得する必要があります。

次の図は、TRACE コマンドの形式を示しています。



SET ON|OFF
トレースのオン、オフを切り替えます。

PSB *psbname*
トレースしたい PSB 名を指定します。各 PSB ごとに別々の TRACE コマンドを出すことにより、複数の PSB を同時にトレースすることができます。

COMP|NOCOMP
DFSDDLT0 への入力として使用されるデータおよび PCB 比較ステートメントを、イメージ・キャプチャー・プログラムに作成させるかどうかを指定します。

バッチ・ジョブとしてのイメージ・キャプチャーの実行

イメージ・キャプチャー・プログラムをバッチ・ジョブとして実行するには、DFSVSAMP DD データ・セットに DLITRACE 制御ステートメントを使用します。

DLITRACE 制御ステートメントに、以下を指定してください。

- プログラムが出したすべての DL/I 呼び出しをトレースしたいのか、いくつかの呼び出しグループだけをトレースしたいのか。
- 以下のいずれへの出力をトレースしたいのか。

  指定する順次データ・セット

  IMS ログ・データ・セット

  順次データ・セットと IMS ログ・データ・セットの両方

トレースされていプログラムが、 CHKP 呼び出しおよび XRST 呼び出しを出す場合は、 DFSDDLT0 とともにトレース出力を使用する際に、チェックポイントおよび再始動の情報を直接複製することはできません。

トレース出力を使用して、IMS DL/I または DBB バッチ領域で DFSDDLT0 を稼働する場合、その結果はアプリケーション・プログラムの結果と同じになりますが、それはデータベースが更新されていなかった場合だけです。

DFSVSAMP DD データ・セット中の DLITRACE 制御ステートメントの形式については、「*IMS V15* システム定義」のトピック『DL/I 呼び出しイメージ・トレースの定義』を参照してください。

ログ・データ・セットからのイメージ・キャプチャー・データのリトリーブ

トレース出力が IMS ログ・データ・セットに送信される場合は、ユーティリティー DFSERA10 および DL/I 呼び出しトレース出口ルーチン DFSERA50 を使用してリトリーブできます。DFSERA50 は、リトリーブされるイメージ・キャプチャー・プログラム・レコードの非ブロック化、形式設定、および番号付けを行います。

DFSERA50 を使用するには、DFSERA10 入力ストリームに、順次出力データ・セットを定義する DD ステートメントを挿入しなければなりません。 この DD ステートメントのデフォルト DD 名は、TRCPUNCH です。ステートメントは、BLKSIZE=80 を指定する必要があります。

例えば、SYSIN データ・セットに以下の DFSERA10 入力制御ステートメントの例を使用して、ログ・データ・セットからイメージ・キャプチャー・プログラム・データをリトリーブすることができます。

- すべてのイメージ・キャプチャー・プログラム・レコードを印刷する。

```
Column 1       Column 10
OPTION         PRINT OFFSET=5,VALUE=5F,FLDTYP=X
```

- **PSB** 名ごとに選択したイメージ・キャプチャー・プログラム・レコードを印刷する。

```
Column 1       Column 10
OPTION         PRINT OFFSET=5,VALUE=5F,COND=M
OPTION         PRINT OFFSET=25,VLDTYP=C,FLDLEN=8,
               VALUE=psbname, COND=E
```

- イメージ・キャプチャー・プログラム・レコードを形式設定する **(DFSDDLT0** への入力として使用可能な形式に**)**。

```
Column 1       Column 10
OPTION         PRINT OFFSET=5,VALUE=5F,COND=M
OPTION         PRINT EXITR=DFSERA50,OFFSET=25,FLDTYP=C
               VALUE=psbname,FLDLEN=8,DDNAME=OUTDDN,COND=E
```

要確認**:** DDNAME= パラメーターは、 DFSERA50 が使用する DD ステートメントの名前を指定します。OUTDDN DD ステートメントに定義されたデータ・セットが、デフォルト TRCPUNCH DD ステートメントの代わりに使用されます。この例では、DD は以下のようになります。

```
//OUTDDN DD ...,DCB=(BLKSIZE=80),...
```

IMS プログラムのモニターおよびデバッグの要求

STAT および LOG 要求を使用すると、プログラムをデバッグする際に役立ちます。

- 統計 (STAT) 呼び出しは、データベース統計をリトリーブします。
- LOG (ログ) 呼び出しは、アプリケーション・プログラムが、システム・ログにレコードを書き込めるようにします。

拡張 OSAM 呼び出しおよび VSAM STAT 呼び出しは、特定の必要性のためにシステムのパフォーマンス・モニターと微調整用の追加情報を提供します。

拡張 STAT 呼び出しが出されると、次の情報が戻されます。

- 定義された各サブプールに関する OSAM 統計
- ハイパースペース 統計も含む VSAM 統計
- 10 桁まで拡張された、OSAM および VSAM カウント・フィールド

データベース統計のリトリーブ: STAT 呼び出し

STAT 呼び出しは、IMS データベース統計をリトリーブするので、プログラムをデバッグする際に役立ちます。また、パフォーマンスのモニターおよび微調整を行う際にも役立ちます。 STAT 呼び出しは、OSAM データベース・バッファー・プール統計、および VSAM データベース・バッファー・サブプール統計をリトリーブします。

このトピックにはプロダクト・センシティブ・プログラミング・インターフェース情報が含まれています。

STAT 呼び出しを出す場合には、以下のことを指示してください。

- 統計が戻される入出力域
- 9 バイト域の名前であり、戻したい統計のタイプおよび形式を説明する内容の統計機能。 この領域の内容は、次のように定義されます。
  - 最初の 4 バイトは、要求された統計のタイプ (OSAM あるいは VSAM) を定義しています。
  - 5 番目のバイトは、戻される形式 (定様式、不定様式、あるいは要約) を定義しています。
  - 残りの 4 バイトは、次のように定義されます。
    - 通常または強化 STAT 呼び出しには、4 バイトのブランクが含まれます。
    - 拡張 STAT 呼び出しには、4 バイトのパラメーター ' E1 ' (最初の 1 バイトがブランク、次の 2 バイトが文字ストリングで最後の 1 バイトがブランク) が含まれます。

関連資料:

STAT 呼び出し (アプリケーション・プログラミング API)

OSAM バッファー・プール統計の形式

OSAM バッファー・プール統計では、統計機能パラメーターおよびデータ形式の値をアプリケーション・プログラムに戻すことが可能です。 OSAM バッファー・プールがない場合は、GE 状況コードがプログラムに戻されます。

DBASF

この機能値は、定様式の書式で、全 OSAM データベース・バッファー・プール統計を提供します。 アプリケーション・プログラム入出力域は、少なくとも 360 バイトなければなりません。 3 つの 120 バイトのレコード (印刷用に形式設定) が、2 行の見出しと 1 行の統計として用意されます。次の図はデータ形式を示します。

```
  BLOCK    FOUND    READS     BUFF     OSAM   BLOCKS     NEW   CHAIN
    REQ  IN POOL   ISSUED     ALTS   WRITES  WRITTEN  BLOCKS  WRITES
nnnnnnn  nnnnnnn    nnnnn  nnnnnnn  nnnnnnn  nnnnnnn   nnnnn   nnnnn
```

```
WRITTEN  LOGICAL    PURGE  RELEASE
 AS NEW      CYL      REQ      REQ  ERRORS
          FORMAT
nnnnnnn  nnnnnnn  nnnnnnn  nnnnnnn  nn/nn
```

BLOCK REQ
受信したブロック要求の数

FOUND IN POOL
要求されたブロックがバッファー・プールで検出された回数

READS ISSUED
出された OSAM 読み取りの数

BUFF ALTS
プールで更新されたバッファーの数

OSAM WRITES
出された OSAM 書き込みの数

BLOCKS WRITTEN
プールから書き込まれたブロックの数

NEW BLOCKS
プールで作成された新規のブロックの数

CHAIN WRITES
出されたチェーン OSAM 書き込みの数

WRITTEN AS NEW
作成されたブロックの数

LOGICAL CYL FORMAT
出された論理シリンダーのフォーマット要求の数

PURGE REQ
ユーザー要求の除去の数

RELEASE REQ
所有権解放要求の数

ERRORS
現在プール内にある、書き込みエラー・バッファーの数、または今回の実行中にプールで起こった、エラーの最大回数

DBASU

この機能値は、不定様式の書式で、全 OSAM データベース・バッファー・プール統計を示します。アプリケーション・プログラム入出力域は、少なくとも 72 バイトなければなりません。 フルワードの 2 進データが 18 個、以下のように提供されます。

ワード 内容

1 後続のワード数のカウント

2-18 DBASF 機能値から与えられるものと同一順序の統計値

DBASS

この機能値は、定様式の書式で OSAM データベース・バッファー・プール統計の要約を示します。アプリケーション・プログラム入出力域は、少なくとも 180 バイトなければなりません。 60 バイトのレコード（印刷用に形式設定）が 3 個、提供されます。次の図はデータ形式を示します。

```
DATA BASE BUFFER POOL:  SIZE nnnnnnn
  REQ1 nnnnn REQ2 nnnnn READ nnnnn WRITES nnnnn LCYL nnnnn
  PURG nnnnn OWNRR nnnnn ERRORS nn/nn
```

SIZE バッファー・プールのサイズ

REQ1 ブロック要求の数

REQ2 プール内で適合しているブロック要求数および作成された新規のブロックの数

READ
出された読み取り要求の数

WRITES
出された OSAM 書き込みの数

LCYL 形式論理シリンダー要求の数

PURG
ユーザー要求の除去の数

OWNRR
所有権解放要求の数

ERRORS
現在プール内にある、永続エラーの数、またはこの実行中に起こった、永続エラーの最大回数

VSAM バッファー・サブプール統計の形式

VSAM データベースに対して、複数のバッファー・サブプールがある可能性があるので、 STAT 呼び出しは、これらの統計を要求するときに繰り返されます。複数の VSAM ローカル共用リソース・プールが定義されている場合、すべての VSAM ローカル共用リソース・プールについて、定義された順序で統計がリトリーブされます。各ローカル共用リソース・プールでは、バッファー・サイズに応じて、各サブプールについて統計がリトリーブされます。

最初に呼び出しが出される際、最小バッファー・サイズを使用したサブプールに関する統計が得られます。後続の各呼び出し (PCB の使用を介在させない) では、次に大きなバッファー・サイズを使用したサブプールについて、統計が供給されます。

索引サブプールがローカル共用リソース・プールの内部にある場合、索引サブプール統計は、必ず、データ・サブプールの統計の後になります。また索引サブプール統計は、バッファー・サイズに基づいた昇順でリトリーブされます。

一連の呼び出しの最終呼び出しでは、PCB 内の GA 状況コードを戻します。戻された統計は、すべてのローカル共用リソース・プール内のすべてのサブプールについての合計です。 VSAM バッファー・サブプールが一つもない場合、GE 状況コードがプログラムに戻されます。

VBASF

この機能値は、定様式の書式で、全 VSAM データベース・サブプール統計を提供します。 アプリケーション・プログラム入出力域は、少なくとも 360 バイトなければなりません。 3 つの 120 バイトのレコード（印刷用に形式設定）が、2 行の見出しと 1 行の統計として用意されます。後続の各呼び出しは、次のデータ・サブプールに対する統計を戻します。索引サブプールについての統計は、もしあれば、データ・サブプールについての統計の後に続きます。

次の図はデータ形式を示します。

```
                    BUFFER HANDLER STATISTICS
BSIZ NBUF RET RBA RET KEY ISRT ES ISRT KS BFR ALT   BGWRT SYN PTS
nnnK  nnn nnnnnnn nnnnnnn nnnnnnn nnnnnnn nnnnnnn nnnnnnn nnnnnnn

                  VSAM STATISTICS POOLID: xxxx
   GETS  SCHBFR   FOUND   READS USR WTS NUR WTS ERRORS
nnnnnnn nnnnnnn nnnnnnn nnnnnnn nnnnnnn nnnnnnn  nn/nn
```

POOLID
ローカル共用リソース・プールの ID

BSIZ この VSAM サブプールのバッファー・サイズ最終呼び出しの際、このフィールドは ALL に設定されます。

NBUF
このサブプールのバッファー数。 最終呼び出しの際、これはすべてのサブプールのバッファー数になります。

RET RBA
バッファー・ハンドラーで受信される、RBA によるリトリーブ呼び出しの数

RET KEY
バッファー・ハンドラーで受信される、キーによるリトリーブ呼び出しの数

ISRT ES
ESDS 内に挿入された論理レコードの数

ISRT KS
KSDS 内に挿入された論理レコードの数

BFR ALT
このサブプールで更新された論理レコードの数。 KSDS からのレコードの消去結果である削除呼び出しは、カウントされません。

BGWRT
バッファー・ハンドラーが実行した、バックグラウンド書き込み機能の回数

SYN PTS
バッファー・ハンドラーで受信した同期化呼び出しの数

GETS バッファー・ハンドラーが出した VSAM GET 呼び出しの数

SCHBFR
バッファー・ハンドラーが出した VSAM SCHBFR 呼び出しの数

FOUND
サブプール内の制御インターバルを既に VSAM が検出した回数

READS
外部記憶装置から制御インターバルを VSAM が読み取った回数

USR WTS
IMS が開始した VSAM 書き込みの数

NUR WTS
サブプールにスペースを作成するために開始した VSAM 書き込みの数

ERRORS
現在サブプール内にある、書き込みエラー・バッファーの数、またはこの実行中にサブプールで起こった、書き込みエラーの最大回数

VBASU

この機能値は、不定様式の書式で全 VSAM データベース・サブプール統計を示します。 アプリケーション・プログラム入出力域は、少なくとも 72 バイトなければなりません。 フルワードの 2 進データが 18 個、以下のように各サブプールについて提供されます。

ワード 内容

1 後続のワード数のカウント

2-18 この不定様式の書式に含まれない POOLID を除いて、 VBASF 機能値から与えられるものと同一順序の統計値

VBASS

この機能値は、定様式の書式で、 VSAM データベース・サブプール統計の要約を示します。アプリケーション・プログラム入出力域は、少なくとも 180 バイトなければなりません。 60 バイトのレコード (印刷用に形式設定) が 3 個、提供されます。

次の図はデータ形式を示します。

```
DATA BASE BUFFER POOL:  BSIZE nnnnnnn  POOLID xxxx  Type x
  RRBA nnnnn RKEY nnnnn BFALT nnnnn NREC nnnnn SYN PTS nnnnn
  NMBUFS nnn VRDS nnnnn FOUND nnnnn VWTS nnnnn  ERRORS nn/nn
```

BSIZE
この VSAM サブプールのバッファー・サイズ

POOLID
ローカル共用リソース・プールの ID

TYPE データ (D) サブプールあるいは索引 (I) サブプールの区別。

RRBA
RBA によるリトリーブ要求の数

RKEY キーによるリトリーブ要求の数

BFALT
更新された論理レコードの数

NREC
作成された新規の VSAM 論理レコードの数

SYN PTS
同期点要求の数

NMBUFS
この VSAM サブプール内のバッファー数

VRDS
VSAM 制御インターバル読み取りの数

FOUND
サブプール内の要求された制御インターバルを既に VSAM が検出した回数

VWTS
VSAM 制御インターバル書き込みの数

ERRORS
現在サブプール内にある永続書き込みエラーの数、またはこの実行中に起こったエラーの最大回数

強化/拡張 OSAM バッファー・サブプール統計の形式

強化 OSAM バッファー・プール統計は、定義済みサブプールごとに生成された追加情報を示します。 OSAM データベースに対して複数のバッファー・サブプールがある可能性があるので、強化 STAT 呼び出しはこれらの統計を繰り返し要求します。最初に呼び出しが出される際、最小バッファー・サイズを使用したサブプールについて統計が得られます。後続の各呼び出し (PCB の使用を介在させない) では、次に大きなバッファー・サイズを使用したサブプールについて、統計が供給されます。

一連の呼び出しの最終呼び出しでは、PCB 内の GA 状況コードを戻します。戻された統計は、すべてのサブプールの合計です。 OSAM バッファー・サブプールが 1 つもない場合、 GE 状況コードが戻されます。

拡張 OSAM バッファー・プール統計は、強化呼び出し機能の後に 4 バイト・パラメーター 'bE1b' を含めることによってリトリーブできます。拡張 STAT 呼び出しでは、強化呼び出しによって返されるすべての統計値に加え、カップリング・ファシリティー・バッファーの無効化、OSAM キャッシング、順次バッファリングでの IMMED/SYNC 読み取りカウントが返されてきます。

制約事項: 拡張形式のパラメーターは、DBESO、DBESU、DBESF の各関数でのみサポートされます。

DBESF

この機能値は、定様式の書式で、全 OSAM サブプール統計を示します。アプリケーション・プログラムの入出力域は、少なくとも 600 文字なければなりません。 OSAM サブプールについて、120 バイトのレコード (印刷用に形式設定) が 5 個提供されます。 3 つのレコードが見出し行で、2 つのレコードがサブプール統計行です。

以下に拡張 STAT 呼び出し形式の例を示します。

```
      B U F F E R   H A N D L E R   O S A M   S T A T I S T I C S    FIXOPT=X/X   POOLID: xxxx
BSIZ  NBUFS      LOCATE-REQ    NEW-BLOCKS   ALTER- REQ   PURGE- REQ  FND-IN-POOL  BUFRS-SRCH  READ- REQS   BUFSTL-WRT
                   PURGE-WRTS    WT-BUSY-ID   WT-BUSY-WR   WT-BUSY-RD   WT-RLSEOWN   WT-NO-BFRS    ERRORS
nn1K  nnnnnnnn   nnnnnnnnnn    nnnnnnnnnn   nnnnnnnnnn   nnnnnnnnnn  nnnnnnnnnn   nnnnnnnnnn  nnnnnnnnnn   nnnnnnnnnn
                   nnnnnnnnnn    nnnnnnnnnn   nnnnnnnnnn   nnnnnnnnnn   nnnnnnnnnn   nnnnnnnnnn    nnnnnnn/nnnnnnn
```

以下に拡張 STAT 呼び出し形式の例を示します。

```
      B U F F E R   H A N D L E R   O S A M   S T A T I S T I C S   STG CLS=     FIXOPT=N/N  POOLID:
BSIZ  NBUFS      LOCATE-REQ    NEW-BLOCKS   ALTER- REQ   PURGE- REQ  FND-IN-POOL  BUFRS-SRCH  READ- REQS   BUFSTL-WRT
                   PURGE-WRTS    WT-BUSY-ID   WT-BUSY-WR   WT-BUSY-RD   WT-RLSEOWN   WT-NO-BFRS    ERRORS
nn1K  nnnnnnn5   nnnnnnnnn0    nnnnnnnnn0   nnnnnnnnn0   nnnnnnnnn0  nnnnnnnnn0   nnnnnnnnn0  nnnnnnnnn0   nnnnnnnnn0
                   nnnnnnnnnn    nnnnnnnnnn   nnnnnnnnnn   nnnnnnnnnn   nnnnnnnnnn   nnnnnnnnnn    nnnnnnn/nnnnnnn
      CF-READS   EXPCTD-NF    CFWRT-PRI  CFWRT-CHG  STGCLS-FULL       XI-CNT        VECTR-XI    SB-SEQRD   SB-ANTICIP
      nnnnnnnnnn nnnnnnnnnn   nnnnnnnnnn nnnnnnnnnn  nnnnnnnnnn   nnnnnnnnnn      nnnnnnnnnn  nnnnnnnnnn   nnnnnnnnnn
```

FIXOPT

このサブプールに関する固定長オプション。 Y あるいは N は、データ・バッファー接頭部およびデータ・バッファーが、固定長かどうかを表示します。

POOLID

ローカル共用リソース・プールの ID

BSIZ このサブプールのバッファー・サイズ。 合計行については、ALL に設定します。要約合計 (BSIZ=ALL) の場合、OSM= フィールドは FIXOPT および POOLID フィールドを置換します。 このフィールドは、OSAM サブプールの合計サイズです。

NBUFS

このサブプールのバッファー数。 これは、ALL 行についてのプール内の合計バッファー数です。

LOCATE-REQ

LOCATE タイプ呼び出しの数

NEW-BLOCKS

新規ブロックを作成する要求の数

ALTER-REQ

バッファー更新呼び出しの数。 このカウントには、NEW BLOCK 呼び出しと BYTALT 呼び出しの回数が含まれます。

PURGE-REQ

PURGE呼び出しの数

FND-IN-POOL

OSAM プールに既にデータが存在するこのサブプールに対する LOCATE タイプ呼び出しの数

BUFRS-SRCH

すべての LOCATE タイプ呼び出しが検索したバッファー数

READ-REQS

READ 入出力要求の数

BUFSTL-WRT

バッファー・スチール・ルーチンが開始した単一ブロック書き込みの数

PURGE-WRTS

除去によって書き込まれたこのサブプールのブロック数

WT-BUSY-ID
使用中の ID のため待っていた LOCATE 呼び出しの数

WT-BUSY-WR
書き込み使用中のバッファーを待っていた LOCATE 呼び出しの数

WT-BUSY-RD
読み取り使用中のバッファーを待っていた LOCATE 呼び出しの数

WT-RLSEOWN
所有権が解放されるのを待っていたバッファー・スチール要求あるいは除去要求の数

WT-NO-BFRS
スチールに利用可能なバッファーがなかったため待機していたバッファー・スチール要求の数

ERRORS
このサブプールに対する入出力エラーの合計数、または書き込みエラーのためプールにロックされたバッファー数

CF-READS
CF から読み取られたブロックの数

EXPCTD-NF
読み取りが予期されたが読み取られなかったブロックの数

CFWRT-PRI
CF に書き込まれたブロック数 (基本)

CFWRT-CHG
CF に書き込まれたブロックの数 (変更)

STGGLS-FULL
書き込まれなかったブロック数 (STG CLS がいっぱい)

XI-CNTL
XI バッファー無効化呼び出しの回数。

VECTR-XI
VECTOR 呼び出しで XI により無効化が検出されたバッファー数

SB-SEQRD
即時 (SYNC) 順次読み取り数 (SB stat)

SB-ANTICIP
先行読み取り数 (SB stat)

DBESU

この機能値は、不定様式の書式で、全 OSAM 統計を示します。アプリケーション・プログラム入出力域は、少なくとも 84 バイトなければなりません。フルワードの 2 進データが 21 個、以下のように各サブプールについて提供されます。

ワード 内容

1 後続のワード数のカウント

2-19 DBESF 機能値から与えられるものと同一順序で提供される統計

20 サブプールが定義されるときに指定された POOLID

21 2 番目のバイトは、このサブプールに関する次のような固定長オプションを含みます。

- X'04' = DATA BUFFER PREFIX は固定長
- X'02' = DATA BUFFERS は固定長

要約合計 (ワード 2=ALL) にはワード 21 に対して、OSAM プールの合計サイズが入ります。

22-30 DBESF 呼び出しと同じ順序の拡張 STAT データ。

DBESS

この機能値は、定様式の書式で OSAM データベース・バッファー・プール統計の要約を示します。アプリケーション・プログラム入出力域は、少なくとも 360 バイトなければなりません。 60 バイトのレコード（印刷用に形式設定）が 6 つ提供されます。この STAT 呼び出しは、10 桁のカウント・フィールドに再構造化された DBASF STAT 呼び出しです。その他、サブプール・ヘッダー・ブロックからプール内の OSAM バッファーの合計数が分かります。

以下にデータ形式を示します。

```
DATA BASE BUFFER POOL:  NSUBPL nnnnnn  NBUFS nnnnnnnn
  BLKREQ nnnnnnnnnn   INPOOL nnnnnnnnnn   READS  nnnnnnnnnn
  BUFALT nnnnnnnnnn   WRITES nnnnnnnnnn   BLKWRT nnnnnnnnnn
  NEWBLK nnnnnnnnnn   CHNWRT nnnnnnnnnn   WRTNEW nnnnnnnnnn
  LCYLFM nnnnnnnnnn   PURGRQ nnnnnnnnnn   RLSERQ nnnnnnnnnn
  FRCWRT nnnnnnnnnn   ERRORS nnnnnnnn/nnnnnnnn
```

NSUBPL
OSAM バッファー・プールについて定義されたサブプールの数

NBUFS
OSAM バッファー・プールに定義されたバッファーの合計数

BLKREQ
受信したブロック要求の数

INPOOL
バッファー・プールでブロック要求が検出された回数

READS
出された OSAM 読み取りの数

BUFALT
プールで更新されたバッファーの数

WRITES
出された OSAM 書き込みの数

BLKWRT
プールから書き込まれたブロックの数

NEWBLK
プールで作成されたブロックの数

CHNWRT
出されたチェーン OSAM 書き込みの数

WRTNEW
作成されたブロックの数

LCYLFM
出された論理シリンダーのフォーマット要求の数

PURGRQ
ユーザー要求の除去の数

RLSERQ
所有権解放要求の数

FRCWRT
強制書き込み呼び出しの数

ERRORS
現在プール内にある、書き込みエラー・バッファーの数、または今回の実行中にプールで起こった、エラーの最大回数

DBESO

この機能値は、定様式の書式で、 /DIS POOL コマンドの結果として戻されたオンライン統計に対して、全 OSAM データベース・サブプール統計を示します。また、この呼び出しは、ユーザー・アプリケーションの STAT 呼び出しにもなります。アプリケーション DL/I STAT 呼び出しとして出される場合、プログラム入出力域は少なくとも 360 バイトなければなりません。 60 バイトのレコード (印刷用に形式設定) が 6 つ提供されます。

例: 以下に拡張 STAT 呼び出し形式を示します。

```
OSAM DB BUFFER POOL:ID xxxx BSIZE nnnnnK NBUFnnnnnnnn  FX=X/X
  LCTREQ  nnnnnnnnnn  NEWBLK  nnnnnnnnnn  ALTREQ  nnnnnnnnnn
  PURGRQ  nnnnnnnnnn  FNDIPL  nnnnnnnnnn  BFSRCH  nnnnnnnnnn
  RDREQ   nnnnnnnnnn  BFSTLW  nnnnnnnnnn  PURGWR  nnnnnnnnnn
  WBSYID  nnnnnnnnnn  WBSYWR  nnnnnnnnnn  WBSYRD  nnnnnnnnnn
  WRLSEO  nnnnnnnnnn  WNOBFR  nnnnnnnnnn  ERRORS nnnnn/nnnnn
```

例: 以下に拡張 STAT 呼び出し形式を示します。

```
OSAM DB BUFFER POOL:ID xxxx BSIZE nnnnnK NBUFnnnnnnnn  FX=X/X
  LCTREQ  nnnnnnnnnn  NEWBLK  nnnnnnnnnn  ALTREQ  nnnnnnnnnn
  PURGRQ  nnnnnnnnnn  FNDIPL  nnnnnnnnnn  BFSRCH  nnnnnnnnnn
  RDREQ   nnnnnnnnnn  BFSTLW  nnnnnnnnnn  PURGWR  nnnnnnnnnn
  WBSYID  nnnnnnnnnn  WBSYWR  nnnnnnnnnn  WBSYRD  nnnnnnnnnn
  WRLSEO  nnnnnnnnnn  WNOBFR  nnnnnnnnnn  ERRORS nnnnn/nnnnn
  CFREAD  nnnnnnnnnn  CFEXPC  nnnnnnnnnn  CFWRPR nnnnn/nnnnn
  CFWRCH  nnnnnnnnnn  STGCLF  nnnnnnnnnn  XIINV  nnnnn/nnnnn
  XICLCT  nnnnnnnnnn  SBSEQR  nnnnnnnnnn  SBANTR nnnnn/nnnnn
```

POOLID
ローカル共用リソース・プールの ID

BSIZE
このサブプールのバッファー・サイズ。 要約合計行には、ALL を設定します。要約合計 (BSIZE=ALL) の場合、OSAM= フィールドは FX= フィールドを置換します。 このフィールドは、OSAM バッファー・プールの合計サイズです。POOLID は表示されません。要約合計 (BSIZE=ALL) の場合、

OSAM= フィールドは FX= フィールドを置換します。 このフィールドは、OSAM バッファー・プールの合計サイズです。POOLID は表示されません。

NBUF
このサブプールのバッファー数。 ALL 行の場合はプール内の合計バッファー数。

FX= このサブプールに関する固定長オプション。 Y あるいは N は、データ・バッファー接頭部およびデータ・バッファーが、固定長かどうかを表示します。

LCTREQ
LOCATE タイプ呼び出しの数

NEWBLK
新規ブロックを作成する要求の数

ALTREQ
バッファー更新呼び出しの数。 このカウントには、NEW BLOCK 呼び出しと BYTALT 呼び出しの回数が含まれます。

PURGRQ
PURGE呼び出しの数

FNDIPL
OSAM プールに既にデータが存在するこのサブプールに対する LOCATE タイプ呼び出しの数

BFSRCH
すべての LOCATE タイプ呼び出しが検索したバッファー数

RDREQ
READ 入出力要求の数

BFSTLW
バッファー・スチール・ルーチンが開始した、単一ブロック書き込みの数

PURGWR
除去のため書き込んだバッファーの数

WBSYID
使用中の ID のため待っていた LOCATE 呼び出しの数

WBSYWR
書き込み使用中のバッファーを待っていた LOCATE 呼び出しの数

WBSYRD
読み取り使用中のバッファーを待っていた LOCATE 呼び出しの数

WRLSEO
所有権が解放されるのを待っていたバッファー・スチール要求あるいは除去要求の数

WNOBRF
スチールに利用可能なバッファーがなかったため待機していたバッファー・スチール要求の数

ERRORS
このサブプールに対する入出力エラーの合計数、または書き込みエラーのためプールにロックされたバッファー数

CFREAD
CF から読み取られたブロックの数

CFEXPC
読み取りが予期されたが読み取られなかったブロックの数

CFWRPR
CF に書き込まれたブロック数 (基本)

CFWRCH
CF に書き込まれたブロックの数 (変更)

STGCLF
書き込まれなかったブロック数 (STG CLS がいっぱい)

XIINV
XI バッファー無効化呼び出しの回数。

XICLCT
VECTOR 呼び出しで XI により無効化が検出されたバッファー数

SBSEQR
即時 (SYNC) 順次読み取り数 (SB stat)

SBANTR
先行読み取り数 (SB stat)

拡張 VSAM バッファー・サブプール統計の形式

強化 VSAM バッファー・サブプール統計は、仮想記憶域およびハイパースペース内の、VSAM サブプールの合計サイズ情報を示します。すべてのカウント・フィールドは 10 桁です。

VSAM データベースに対して複数のバッファー・サブプールがある可能性があるので、拡張 STAT 呼び出しはこれらの統計を繰り返し要求します。複数の VSAM ローカル共用リソース・プールが定義されている場合、すべての VSAM ローカル共用リソース・プールについて、定義された順序で統計がリトリーブされます。各ローカル共用リソース・プールでは、バッファー・サイズに応じて、各サブプールについて統計がリトリーブされます。

最初に呼び出しが出される際、最小バッファー・サイズを使用したサブプールに関する統計が得られます。後続の各呼び出し (PCB の使用を介在させない) では、次に大きなバッファー・サイズを使用したサブプールについて、統計が供給されます。

索引サブプールがローカル共用リソース・プールの内部にある場合、索引サブプール統計は通常、データ・サブプールの統計の後になります。また索引サブプール統計は、バッファー・サイズに基づいた昇順でリトリーブされます。

一連の呼び出しの最終呼び出しでは、PCB 内の GA 状況コードを戻します。戻された統計は、すべてのローカル共用リソース・プール内のすべてのサブプールについての合計です。 VSAM バッファー・サブプールが一つもない場合、GE 状況コードがプログラムに戻されます。

VBESF

この機能値は、定様式の書式で、全 VSAM データベース・サブプール統計を提供します。 アプリケーション・プログラム入出力域は、少なくとも 600 バイトなければなりません。 各共用リソース・プール ID について、最初の呼び出しは、 120 バイトのレコード (印刷用に形式設定) を 5 個戻します。 3 つのレコードが見出し行で、2 つのレコードがサブプール統計行です。

以下にデータ形式を示します。

```
     B U F F E R   H A N D L E R   S T A T I S T I C S  /  V S A M   S T A T I S T I C S   FIXOPT=X/X/X   POOLID: xxxx
BSIZ NBUFFRS HS-NBUF  RETURN-RBA  RETURN-KEY  ESDS-INSRT  KSDS-INSRT  BUFFRS-ALT  BKGRND-WRT  SYNC-POINT  ERRORS
               VSAM-GETS  SCHED-BUFR  VSAM-FOUND  VSAM-READS  USER-WRITS   VSAM-WRITS  HSRDS-SUCC  HSWRT-SUCC HSR/W-FAIL
nn1K  nnnnnn nnnnnnn  nnnnnnnnnn  nnnnnnnnnn  nnnnnnnnnn  nnnnnnnnnn  nnnnnnnnnn  nnnnnnnnnn  nnnnnnnnnn  nnnnnn/nnnnnn
           nnnnnnnnnn  nnnnnnnnnn  nnnnnnnnnn  nnnnnnnnnn  nnnnnnnnnn  nnnnnnnnnn  nnnnnnnnnn  nnnnnnnnnn nnnnn/nnnnn
```

FIXOPT
このサブプールに関する固定長オプション。 Y あるいは N は、データ・バッファー接頭部、索引バッファー、およびデータ・バッファーが固定長かどうかを指示します。

POOLID
ローカル共用リソース・プールの ID

BSIZ このサブプールのバッファー・サイズ。 合計行については、ALL に設定します。要約合計 (BSIZ=ALL) の場合、 VS= フィールドおよび HS= フィールドは、FIXOPT および POOLID フィールドを置換します。 VS= フィールドは、仮想記憶域の VSAM サブプールの合計サイズです。 HS= フィールドは、ハイパースペースの VSAM サブプールの合計サイズです。

NBUFFRS
このサブプールのバッファー数。 ALL 行に示される VSAM プールのバッファー合計数。

HS-NBUF
このサブプールについて定義されたハイパースペース・バッファー数

RETURN-RBA
バッファー・ハンドラーで受信される、RBA によるリトリーブ呼び出しの数

RETURN-KEY
バッファー・ハンドラーで受信される、キーによるリトリーブ呼び出しの数

ESDS-INSRT
ESDS 内に挿入された論理レコードの数

KSDS-INSRT
KSDS 内に挿入された論理レコードの数

BUFFRS-ALT
このサブプールで更新された論理レコードの数。 KSDS からのレコードの消去結果である削除呼び出しは、カウントされません。

BKGRND-WRT
バッファー・ハンドラーが実行したバックグラウンド書き込み機能の回数

SYNC-POINT
バッファー・ハンドラーで受信した同期化呼び出しの数

ERRORS
現在サブプール内にある、書き込みエラー・バッファーの数、またはこの実行中にサブプールで起こった、書き込みエラーの最大回数

VSAM-GETS
バッファー・ハンドラーが出した VSAM Get 呼び出しの数

SCHED-BUFR
バッファー・ハンドラーが出した、VSAM スケジュール・バッファー呼び出しの数

VSAM-FOUND
バッファー・プール内の制御インターバルを VSAM が検出した回数

VSAM-READS
外部記憶装置から制御インターバルを VSAM が読み取った回数

USER-WRITS
IMS が開始した VSAM 書き込みの数

VSAM-WRITS
サブプールにスペースを作成するために開始した VSAM 書き込みの数

HSRDS-SUCC
ハイパースペース・バッファーからの正常 VSAM 読み取りの数。

HSWRT-SUCC
ハイパースペース・バッファーからの正常 VSAM 書き込みの数

HSR/W-FAIL
ハイパースペース・バッファーからの、失敗した VSAM 読み取りの数/ハイパースペース・バッファーへの失敗した VSAM 書き込みの数。これは、DASD 入出力の結果である、ハイパースペースに対する VSAM READ/WRITE 要求の回数を示します。

VBESU

この機能値は、不定様式の書式で全 VSAM 統計を示します。アプリケーション・プログラム入出力域は、少なくとも 104 バイトなければなりません。各サブプールについて、フルワードの 2 進データが 25 個提供されます。

ワード 内容

1 後続のワード数のカウント

2-23 VBESF 機能値から与えられるものと同一順序で提供される統計

24 サブプールが定義されるときに指定される POOLID

25 最初のバイトはサブプール・タイプ、3 番目のバイトはこのサブプールについて次の固定長オプションを示します。

- X'08' = INDEX BUFFERS は固定長

- X'04' = DATA BUFFER PREFIX は固定長
- X'02' = DATA BUFFERS は固定長

ワード 25 およびワード 26 に代わる要約合計 (ワード 2=ALL) には、仮想プールおよびハイパースペース・プールのサイズが入ります。

VBESS

この機能値は、定様式の書式で、 VSAM データベース・サブプール統計の要約を示します。アプリケーション・プログラム入出力域は、少なくとも 360 バイトなければなりません。各共用リソース・プール ID について、最初の呼び出しは 60 バイトのレコード (印刷用に形式設定) を 6 個戻します。

以下にデータ形式を示します。

```
VSAM DB BUFFER POOL:ID xxxx   BSIZE nnnnnnK   TYPE x   FX=X/X/X
  RRBA  nnnnnnnnnn    RKEY       nnnnnnnnnn   BFALT  nnnnnnnnnn
  NREC  nnnnnnnnnn    SYNC PT    nnnnnnnnnn   NBUFS  nnnnnnnnnn
  VRDS  nnnnnnnnnn    FOUND      nnnnnnnnnn   VWTS   nnnnnnnnnn
  HSR-S nnnnnnnnnn    HSW-S      nnnnnnnnnn   HS NBUFS nnnnnnnn
  HS-R/W-FAIL   nnnnn/nnnnn       ERRORS    nnnnnn/nnnnnn
```

POOLID
ローカル共用リソース・プールの ID

BSIZE
この VSAM サブプールのバッファー・サイズ

TYPE データ (D) サブプールあるいは索引 (I) サブプールの区別。

FX このサブプールに関する固定長オプション。 Y あるいは N は、データ・バッファー接頭部、索引バッファー、およびデータ・バッファーが固定長かどうかを指示します。

RRBA
バッファー・ハンドラーで受信される、RBA によるリトリーブ呼び出しの数

RKEY バッファー・ハンドラーで受信される、キーによるリトリーブ呼び出しの数

BFALT
更新された論理レコードの数

NREC
作成された新規の VSAM 論理レコードの数

SYNC PT
同期点要求の数

NBUFS
この VSAM サブプール内のバッファー数

VRDS
VSAM 制御インターバル読み取りの数

FOUND
サブプール内の要求された制御インターバルを既に VSAM が検出した回数

VWTS
VSAM 制御インターバル書き込みの数

HSR-S
ハイパースペース・バッファーからの正常 VSAM 読み取りの数。

HSW-S
ハイパースペース・バッファーへの正常 VSAM 書き込みの数

HS NBUFS
このサブプールについて定義された VSAM ハイパースペース・バッファー数

HS-R/W-FAIL
ハイパースペース・バッファーからの失敗した VSAM 読み取りの数、およびハイパースペース・バッファーへの失敗した VSAM 書き込みの数。これは、DASD 入出力の結果である、ハイパースペースに対する VSAM READ/WRITE 要求の回数を示します。

ERRORS
現在サブプール内にある永続書き込みエラーの数、またはこの実行中に起こったエラーの最大回数

システム・ログへの情報書き込み: LOG 要求

アプリケーション・プログラムは、LOG 呼び出しを出すことにより、システム・ログにレコードを書き込むことができます。

LOG 要求を出す場合、システム・ログに書き込みたいレコードの入った入出力域を指定します。ログにどんな情報でも書き込むことができ、また、異なるログ・コードを使用して異なる情報タイプを区別することができます。

関連資料: LOG 要求のコーディングについては、適切なアプリケーション・プログラミング情報を参照してください。

IMS プログラムの異常終了時の処置

プログラムが異常終了した場合、次の処置を実行すると、問題の検出および修正の作業を単純化することができます。

- 異常終了したプログラム下の状況について、できるだけたくさんの情報を記録してください。
- 確かな初期設定および実行エラーについて検査してください。

IMS プログラムの異常終了後の推奨処置

多くの場所に、プログラムが異常終了した場合の処置に関するガイドラインがあります。ここでは、共通のガイドラインを提案します。

- エラー状態を文書化して、調査および訂正に役立ててください。次の情報が役立つことがあります。
  - プログラムの PSB 名
  - プログラムが処理していたトランザクション・コード (オンライン・プログラムのみ)
  - 処理されていた入力メッセージのテキスト (オンライン・プログラムのみ)
  - 呼び出し機能

  - 起点論理端末名 (オンライン・プログラムのみ)
  - 実行されていた呼び出しで参照されていた PCB の内容
  - 問題が発生した際の入出力域の内容
  - データベース呼び出しが実行されていた際に呼び出しが使用していた SSA
  - 日時
- プログラムでエラーが発生した場合、すべての必要なエラー情報を標準エラー・ルーチンに渡すことができます。 プログラムで STAE ルーチンまたは ESTAE ルーチンを使用しないでください。 IMS は STAE ルーチンまたは ESTAE ルーチンを使用して、アプリケーション・プログラムのあらゆる異常終了を制御領域に通知します。独自の STAE ルーチンまたは ESTAE ルーチンを呼び出す場合、異常終了が起こると IMS が制御を失う可能性があります。
- オンライン・プログラムは、メッセージを発信元論理端末に送信して、エラーが発生していることを、端末装置で作業している人に通知することができます。 CCTL を使用していない場合、プログラムは I/O PCB から論理端末名を得て、高速 PCB に記憶するとともに、1 つあるいは複数の ISRT 呼び出しを出して、メッセージを送信することができます。
- さらに、オンライン・プログラムは、プログラムの終了についての情報を知らせるメッセージをマスター端末オペレーターに送信することができます。このためには、プログラムはマスター端末の論理端末名を高速 PCB に記憶し、 1 つあるいは複数の ISRT 呼び出しを出します。 (CCTL を使用している場合は、これは適用されません。)
- さらに、メッセージをプリンターに送信して、エラー・レコードのハードコピーを得ることができます。
- LOG 要求を出してシステム・ログにメッセージを送信できます。
- 場所によっては、一日の終わりに BMP を実行して、その日発生したすべてのエラーをリストするところもあります。お客様のショップでこれを行う場合、その BMP 用に設定された宛先をもつ高速 PCB を使用して、メッセージを送信できます。 (CCTL を使用している場合は、これは適用されません。)

IMS プログラムの異常終了の診断

テストしているプログラムおよび実行しているプログラムが正しく稼働しない場合、問題を分離する必要があります。問題はプログラミング・エラー (例えば、要求したうちの一つをコーディングした方法のエラー) から、システム問題までのものである可能性があります。このセクションでは、アプリケーション・プログラマーを対象に、プログラムが実行に失敗する、異常終了する、または誤った結果を生成するという場合に取ることができる手順についてのいくつかのガイドラインを示しています。

IMS プログラムの初期設定エラー

プログラムが制御を受け取る前に、IMS は、アプリケーション・プログラムが使用している PSB および DBD を正しくロードし、初期設定しなければなりません。この領域に問題があると、システム・プログラマーまたは DBA (あるいは同等の資格の専門家) に修正を依頼することがしばしば必要になります。アプリケーション・プログラマーが行えるのは、DBD、PSB、および生成された制御ブロックに最近変更があったかどうかを見つけ出すことです。

IMS プログラム実行エラー

初期設定エラーがない場合には、以下の点を検査してください。

1. コンパイラーからの出力。すべてのエラー・メッセージが解決されているかどうかを確かめてください。
2. バインダーからの出力。
   - すべての外部参照が解決されていますか?
   - すべての必要なモジュールが組み込まれていましたか?
   - 言語インターフェース・モジュールが正しく組み込まれていましたか?
   - 正しいエントリー・ポイントが指定されていますか?
3. 使用中の JCL。
   - データベースを含むファイルを説明した情報は正確ですか? 正しくない場合、DBA に相談してください。
   - 正しい形式で DL/I パラメーター・ステートメントを組み込みましたか?
   - EXEC ステートメントに領域サイズ・パラメーターを組み込みましたか? IMS およびプログラムに必要なストレージに対して、十分な大きさの領域あるいは区画を指定しましたか?
   - PCB マスクのフィールドを正しく宣言しましたか?
   - プログラムがアセンブラー言語プログラムである場合には、レジスターを正しく保管あるいは復元しましたか? 入力点で PCB アドレスのリストを保管しましたか? どの DL/I 呼び出しを出すよりも前に、レジスター 1 は、フルワードのパラメーター・リストを指示しましたか?
   - COBOL for z/OS および PL/I for MVS and VM について、DL/I 呼び出しの引数に対して使用しているリテラルは、予想どおりの結果を出しましたか? 例えば、PL/I for MVS and VM では、パラメーター・カウントはフルワードの代わりにハーフワードとして生成されていますか? また機能コードは、4 バイトの要求されたフィールドを作成していますか?
   - できるだけ多くの PCB を使用して、プログラム内で間違った結果を出しているのは何かを判別してください。

関連概念:

63 ページの『IMS プログラムでの STAE または ESTAE、および SPIE の使用』

第 9 章 CICS アプリケーション・プログラムのテスト

CICS アプリケーション・プログラムに対してプログラム単体テストを実行し、入力データ、処理、出力データをプログラムが正しく処理することを確かめる必要があります。実行するテストの量およびタイプは、個々のプログラムによって異なります。

CICS プログラムのテストに関する推奨事項

プログラムのテストの準備ができたら、テストを開始する前に、確立されたテスト手順について確認してください。

テストを開始するには、以下の 3 項目が必要です。

- テスト JCL
- テスト・データベース。プログラムをテストする場合、実動データベースに対してテストを実行してはなりません。失敗すると、プログラムが有効なデータを破壊する恐れがあります。
- テスト入力データ。使用する入力データは現行のものである必要はありませんが、有効なデータでなければなりません。有効な入力データを使用しない限り、出力データが有効であることが確認できません。

プログラム・テストの目的は、起こる可能性のあるあらゆる状態を、プログラムが正しく処理できるかどうか確認することです。

徹底的にプログラムをテストするには、プログラムがとりうるできるだけたくさんの経路をテストしてください。以下に例を示します。

- プログラムに各分岐を実行させる入力データを使用して、プログラムの各経路をテストしてください。
- プログラムがエラー・ルーチンをテストしていることを確かめてください。 プログラムにできるだけ多くのエラー条件をテストさせる入力データを使用してください。
- プログラムで使用している編集ルーチンをテストしてください。 プログラムにできるだけたくさんの異なるデータの組み合わせを与えて、入力データが正しく編集されていることを確認してください。

CICS プログラムのテスト

CICS プログラムは、プログラムのタイプに応じて異なるツールを使用してテストできます。

次の表では、オンライン DBCTL、バッチ、および BMP の各プログラムで使用可能なツールを要約しています。

表 30. プログラムのテストに使用できるツール：

ツール	オンライン (DBCTL)	バッチ	BMP
実行診断機能 (EDF)	可[1]	なし	なし
CICS ダンプ管理	あり	なし	なし
CICS トレース管理	あり	あり	なし
DFSDDLT0	なし	可[2]	可[2]
DL/I イメージ・キャプチャー・プログラム	あり	あり	あり

注:

1. オンラインでは、コマンド・レベル・プログラムのみです。
2. 呼び出しレベル・プログラムのみです。 (コマンド・レベル・バッチ・プログラムでは、最初に DL/I イメージ・キャプチャー・プログラムを使用して、 DFSDDLT0 に関する呼び出しを作成できます。)

実行診断機能の使用 (コマンド・レベルのみ)

実行診断機能 (EDF) を使用して、オンラインのコマンド・レベル・プログラムをテストすることができます。 EDF はオンライン・プログラムの EXEC CICS および EXEC DLI コマンドを表示できますが、 DL/I 呼び出しを代行受信することはできません。

EDF では、以下の処置が可能です。

- 作業用ストレージの表示および修正。DIB の値を変更できます。
- 実行前のコマンドの表示および修正。引数の値を修正してからコマンドを実行できます。
- コマンド実行後の戻りコードの修正。コマンドを実行したが制御がアプリケーション・プログラムに戻る前に、コマンドが代行受信され、応答および CICS が設定した引数の値を表示します。

テストするプログラムと同一端末装置上で、EDF を稼働できます。

関連資料: EDF の使用について詳しくは、「CICS Transaction Server for z/OS CICS アプリケーション・プログラミング・リファレンス」の『実行診断機能 (EDF)』を参照してください。

CICS ダンプ管理の使用

CICS ダンプ管理機能を使用して、仮想記憶域、CICS の表、およびタスクに関するストレージ域をダンプすることができます。 CICS ダンプ管理機能の使用に関する詳細については、使用中の CICS のバージョンに対応する CICS アプリケーション・プログラミング解説書を参照してください。

CICS トレース管理の使用

トレース管理機能を使用すると、DBCTL 環境のでオンライン・プログラムのデバッグおよびモニターに役立ちます。トレース管理要求を使用して、トレース・テーブルの項目を記録できます。仮想記憶域と補助記憶域のどちらにもトレース・テー

ブルを記録することができます。仮想記憶域にある場合は、ダンプを調査するとトレース・テーブルにアクセスできます。補助記憶域にある場合は、トレース表を印刷できます。トレース項目を作成するため使用する制御ステートメントに関する詳細については、使用中の CICS のバージョンに対応するアプリケーション・プログラミング解説書のトレース管理に関する説明を参照してください。

イメージ・キャプチャーを使用する DL/I 呼び出しのトレース

DL/I イメージ・キャプチャー・プログラム (DFSDLTR0) は、バッチ、BMP、およびオンライン (DBCTL 環境) プログラムが出す DL/I 呼び出しをトレースし、記録できるトレース・プログラムです。コマンド・レベル・プログラムにイメージ・キャプチャー・プログラムを使用することもでき、 DFSDDLT0 への入力データとして使用する呼び出しを作成できます。

イメージ・キャプチャー・プログラムを使用して、以下のことを行うことができます。

- プログラムのテスト

  イメージ・キャプチャー・プログラムは、トレースしている呼び出しでエラーを検出すると、できるだけ多くの呼び出しを複製します。 ただし、エラーが発生した場所を文書化することはできず、また常にすべての SSA を複製できるとは限りません。

- **DFSDDLT0 (DL/I** テスト・プログラム**)** 用入力の作成

  イメージ・キャプチャー・プログラムが作成した出力を、 DFSDDLT0 への入力として使用することができます。イメージ・キャプチャー・プログラムは、状況ステートメント、コメント・ステートメント、呼び出しステートメント、および比較ステートメントを、DFSDDLT0 用に作成します。例えば、コマンド・レベル・プログラムにイメージ・キャプチャー・プログラムを使用して、DFSDDLT0 用の呼び出しを作成することができます。

- プログラムのデバッグ

  プログラムが異常終了する場合、イメージ・キャプチャー・プログラムを使用してプログラムを再実行できます。イメージ・キャプチャー・プログラムは、プログラム障害を引き起こす条件を複製し、文書化します。イメージ・キャプチャー・プログラムが作成した報告書の情報を使用して、問題を検出して修正することができます。

イメージ・キャプチャーの DFSDDLT0 との併用

イメージ・キャプチャー・プログラムは、DFSDDLT0 への入力として使用できる次の制御ステートメントを作成します。

- 状況ステートメント

  イメージ・キャプチャー・プログラムを呼び出すと、状況ステートメントが作成されます。状況ステートメントは、以下のことを行います。

  – 印刷オプションを設定します。これにより DFSDDLT0 は、すべての呼び出しトレース・コメント、すべての DL/I 呼び出し、およびすべての比較の結果を印刷します。

– アプリケーション・プログラムの実行中に PCB に変更が発生するたびに、新規の相対 PCB 番号を判別します。

- コメント・ステートメント

  イメージ・キャプチャー・プログラムを呼び出すと、コメント・ステートメントも作成されます。コメント・ステートメントから以下のものが得られます。

  – IMS がトレースを開始した時刻および日付
  – トレースされている PSB の名前

  またイメージ・キャプチャー・プログラムは、IMS によるエラー検出の対象となるどの呼び出しよりも前に、コメント・ステートメントを作成します。

- 呼び出しステートメント

  イメージ・キャプチャー・プログラムは、アプリケーション・プログラムが出す DL/I 呼び出しまたは EXEC DLI コマンドのそれぞれについて、呼び出しステートメントを作成します。また、トレースを開始する際、および各コミット・ポイントあるいは CHKP 要求の後で、CHKP 呼び出しを生成します。

- 比較ステートメント

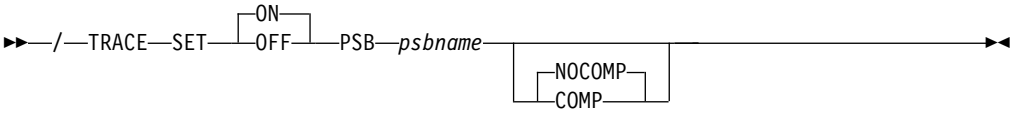
  DLITRACE 制御ステートメントに COMP を指定する場合、イメージ・キャプチャー・プログラムはデータおよび PCB 比較ステートメントを作成します。

オンラインでのイメージ・キャプチャーの実行

オンラインでイメージ・キャプチャー・プログラムを実行する場合、トレース出力は IMS ログ・データ・セットに送られます。オンラインでイメージ・キャプチャー・プログラムを稼働するには、 z/OS コンソールから IMS TRACE コマンドを出してください。

BMP をトレースしている場合、および DFSDDLT0 にトレース結果を使用しようとしている場合、BMP には処理を行っているデータベースへの排他的書き込みアクセスがなければなりません。 アプリケーション・プログラムに排他的アクセスがない場合、 DFSDDLT0 の結果はアプリケーション・プログラムの結果と異なってしまう可能性があります。

次の図は TRACE コマンドの形式を示します。



SET ON|OFF
トレースのオン、オフを切り替えます。

PSB psbname
トレースしたい PSB 名を指定します。各 PSB ごとに別々の TRACE コマンドを出すことにより、複数の PSB を同時にトレースすることができます。

COMP|NOCOMP
DFSDDLT0 で使用されるデータおよび PCB 比較ステートメントを、イメージ・キャプチャー・プログラムに作成させるかどうかを指定します。

バッチ・ジョブとしてのイメージ・キャプチャーの実行

イメージ・キャプチャー・プログラムをバッチ・ジョブとして稼働するには、DFSVSAMP DD データ・セットに DLITRACE 制御ステートメントを使用します。

DLITRACE 制御ステートメントに、以下を指定してください。

- プログラムが出したすべての DL/I 呼び出しをトレースしたいのか、いくつかの呼び出しグループだけをトレースしたいのか。
- 以下のいずれへの出力をトレースしたいのか。

  指定する順次データ・セット

  IMS ログ・データ・セット

  順次データ・セットと IMS ログ・データ・セットの両方

トレースされていプログラムが、 CHKP 呼び出しおよび XRST 呼び出しを出す場合は、 DFSDDLT0 とともにトレース出力を使用する際に、チェックポイントおよび再始動の情報を直接複製することはできません。

トレース出力を使用して、IMS DL/I または DBB バッチ領域で DFSDDLT0 を稼働する場合、その結果はアプリケーション・プログラムの結果と同じになりますが、それはデータベースが更新されていなかった場合だけです。

DFSVSAMP DD データ・セット中の DLITRACE 制御ステートメントの形式については、「*IMS V15* システム定義」のトピック『DL/I 呼び出しイメージ・トレースの定義』を参照してください。

DLITRACE の例

この例では、DLITRACE 制御ステートメントによる、プログラム発行の最初の 14 の DL/I 呼び出しまたはコマンドのトレース、IMS ログ・データ・セットへの出力の送信、および DFSDDLT0 用のデータおよび PCB 比較のステートメントの生成を示しています。

```
//DFSVSAMP DD *
DLITRACE LOG=YES,STOP=14,COMP
/*
```

特殊な JCL 要件

特殊な JCL 要件は次のとおりです。

//IEFRDER DD
ログ・データ・セットの出力が必要な場合、IMS ログ・データ・セットを定義するにはこの DD ステートメントが必要です。

//DFSTROUT DD|anyname
順次データ・セットの出力が必要な場合は、データ・セットを定義するのにこの

DD ステートメントが必要です。 代替 DD 名 (任意の名前) を指定したい場合は、使用する DDNAME パラメーターを DLITRACE 制御ステートメントに指定してください。

JCL ステートメントの DCB パラメーターは、必須ではありません。データ・セットの特性は、以下のとおりです。

- RECFM=F
- LRECL=80

イメージ・キャプチャーを使用する際の注意事項

- トレースされているプログラムが CHKP および XRST 呼び出しを出す場合は、DFSDDLT0 とともにトレース出力を使用する際に、チェックポイントおよび再始動を直接複製することができません。
- トレース出力を使用して IMS DL/I または DBB バッチ領域で DFSDDLT0 を稼働する場合、その結果はデータベースが更新されない限り、アプリケーション・プログラムの結果と同じになります。

ログ・データ・セットからのイメージ・キャプチャー・データのリトリーブ

トレース出力が IMS ログ・データ・セットに送信される場合は、ユーティリティー DFSERA10 と、DL/I 呼び出しトレース出口ルーチン DFSERA50 を使用してリトリーブすることができます。 DFSERA50 は、リトリーブされるイメージ・キャプチャー・プログラム・レコードの非ブロック化、形式設定、および番号付けを行います。 DFSERA50 を使用するには、DFSERA10 入力ストリームに、順次出力データ・セットを定義する DD ステートメントを挿入しなければなりません。 この DD ステートメントのデフォルト DD 名は、TRCPUNCH です。カードは、BLKSIZE=80 を指定してください。

例えば、SYSIN データ・セットに以下の DFSERA10 入力制御ステートメントの例を使用して、ログ・データ・セットからイメージ・キャプチャー・プログラム・データをリトリーブすることができます。

- すべてのイメージ・キャプチャー・プログラム・レコードを印刷する。

```
Column 1       Column 10
OPTION         PRINT OFFSET=5,VALUE=5F,FLDTYP=X
```

- **PSB** 名ごとに選択したイメージ・キャプチャー・プログラム・レコードを印刷する。

```
Column 1       Column 10
OPTION         PRINT OFFSET=5,VALUE=5F,COND=M
OPTION         PRINT OFFSET=25,VLDTYP=C,FLDLEN=8,
               VALUE=psbname, COND=E
```

- イメージ・キャプチャー・プログラム・レコードを形式設定する **(DFSDDLT0** への入力として使用可能な形式に**)**。

```
Column 1       Column 10
OPTION         PRINT OFFSET=5,VALUE=5F,COND=M
OPTION         PRINT EXITR=DFSERA50,OFFSET=25,FLDTYP=C
               VALUE=psbname,FLDLEN=8,DDNAME=OUTDDN,COND=E
```

DFSERA50 が使用する DD ステートメントの名前を指定するのに、DDNAME= パラメーターが使用されます。 OUTDDN DD ステートメントに定義されたデータ・

セットは、デフォルト TRCPUNCH DD ステートメントの代わりに使用されます。この例では、DD は以下のようになります。

```
//OUTDDN DD ...,DCB=(BLKSIZE=80),...
```

CICS プログラムのモニターおよびデバッグの要求

STAT および LOG 要求を使用すると、プログラムをデバッグする際に役立ちます。

- 統計 (STAT) 要求は、データベース統計をリトリーブします。 STAT は、呼び出しレベル・プログラムとコマンド・レベル・プログラムの両方から発行できます。
- LOG (ログ) 要求により、アプリケーション・プログラムはシステム・ログにレコードを書き込むことができます。バッチ・プログラムのコマンドまたは呼び出しとして、LOG を出すことができます。この場合、レコードは IMS ログに書き込まれます。 DBCTL 環境のオンライン・プログラムの呼び出しまたはコマンドとして、 LOG を出すことができます。この場合、レコードは DBCTL ログに書き込まれます。

CICS プログラムの異常終了時の処置

プログラムが異常終了したときに、必ずいくつかの処理をとることができれば、問題の検出および修正の作業を単純化することができます。

1 つ目は、異常終了したプログラムの状況についてできるだけ多くの情報を記録できることです。 2 つ目は、確かな初期設定エラーおよび実行エラーについて検査できることです。

CICS 異常終了後の推奨処置

多くの場所に、プログラムが異常終了した場合の処置に関する指針があります。ここでは、共通に使用できる指針をいくつか提案します。

- エラー状態を文書化して、調査および訂正に役立ててください。役立つ情報のいくつかを、以下に示します。
  - プログラムの PSB 名
  - プログラムが処理していたトランザクション・コード (オンライン・プログラムのみ)
  - 処理されていた入力画面のテキスト (オンライン・プログラムのみ)
  - 呼び出し機能
  - 端末装置 ID (オンライン・プログラムのみ)
  - PCB あるいは DIB の内容
  - 問題が発生した際の入出力域の内容
  - データベース要求が実行されていた場合、(もしあれば) その要求が使用する SSA あるいは SEGMENT および WHERE オプション
  - 日時
- プログラムでエラーが発生した場合、すべての必要なエラー情報を標準エラー・ルーチンに渡すことができます。

- またオンライン・プログラムは、プログラム終了についての情報を含むメッセージを、マスター端末宛先 (CSMT) およびアプリケーション端末オペレーターに送信する必要があります。
- LOG 要求を出してシステム・ログにメッセージを送信できます。

CICS の異常終了の診断

テストしているプログラムおよび実行しているプログラムが正しく稼働しない場合、問題を分離する必要があります。問題はプログラミング・エラー (例えば、要求したうちの一つをコーディングした方法のエラー) から、システム問題までのものである可能性があります。このセクションでは、アプリケーション・プログラマーを対象に、プログラムが実行に失敗する、異常終了する、または誤った結果を生成するという場合に取ることができる手順についてのいくつかのガイドラインを示しています。

CICS 初期化エラー

プログラムが制御を受け取る前に、IMS は、アプリケーション・プログラムが使用している PSB および DBD を正しくロードし、初期設定しなければなりません。この領域に問題があると、システム・プログラマーまたは DBA (あるいは同等の資格の専門家) に修正を依頼することがしばしば必要になります。アプリケーション・プログラマーが行えるのは、DBD、PSB、および生成された制御ブロックに最近変更があったかどうかを見つけ出すことです。

CICS 実行時エラー

初期設定エラーがない場合には、プログラムの以下の点を検査してください。

1. コンパイラーからの出力。すべてのエラー・メッセージが解決されているかどうかを確かめてください。
2. バインダーからの出力。
   - すべての外部参照が解決されていますか?
   - すべての必要なモジュールが組み込まれていましたか?
   - 言語インターフェース・モジュールが正しく組み込まれていましたか?
   - 正しいエントリー・ポイントが指定されていますか (バッチ・プログラムの場合のみ) ?
3. 使用中の JCL。
   - データベースを含むファイルを説明した情報は正確ですか? 正しくない場合、DBA に相談してください。
   - 正しい形式で DL/I パラメーター・ステートメントを組み込みましたか? (バッチ・プログラムの場合のみ)
   - EXEC ステートメントに領域サイズ・パラメーターを組み込みましたか? IMS およびプログラムに必要なストレージに対して、十分な大きさの領域あるいは区画を指定しましたか (バッチ・プログラムの場合のみ) ?
4. 呼び出しレベル・プログラム:
   - PCB マスクのフィールドを正しく宣言しましたか?
   - プログラムがアセンブラー言語プログラムである場合には、レジスターを正しく保管あるいは復元しましたか? 入力点で PCB アドレスのリストを保管

しましたか? どの DL/I 呼び出しを出すよりも前に、レジスター 1 は、フルワードのパラメーター・リストを指示しましたか?

- COBOL for z/OS および PL/I for MVS and VM について、DL/I 呼び出しの引数に対して使用しているリテラルは、予想どおりの結果を出しましたか? 例えば、PL/I for MVS and VM では、パラメーター・カウントはフルワードの代わりにハーフワードとして生成されていますか? また機能コードは、4 バイトの要求されたフィールドを作成していますか?
- できるだけ多くの PCB を使用して、プログラム内で間違った結果を出しているのは何かを判別してください。

5. コマンド・レベル・プログラム:
   - ISRT あるいは REPL コマンドに、 FROM オプションを使用しましたか? 使用していない場合、データはデータベースに転送されません。
   - 変換プログラム・メッセージ中にエラー・メッセージがないかを検査してください。

第 10 章 アプリケーション・プログラムの文書化

お客様の多くは、プログラム文書化用の標準を確立しています。お客様でもこの確立された標準が認識されていることをご確認ください。

他のプログラマーのための文書

プログラムの文書化は、プロジェクトの最後に行うようなものではありません。プログラムの構成時およびコーディング時にそのプログラムに関する情報を記録しておくと、その文書は他のプログラマーにとってはるかに詳細で、一層役立つものになります。プログラムを使用して作業する人にとって有効だと思われることは、どんな情報でも提供してください。

この情報を記録するのは、あるコマンド、オプション、呼び出し構造、およびコマンド・コードを選択した理由を、プログラムの保守担当者に知ってもらうためです。例えば、DBA が何らかの方法でデータベースを再編成しようとした場合は、プログラムがデータにアクセスする方法およびその理由に関する情報が役立ちます。

他のプログラマーのために記録できる情報には、以下のものが含まれます。

- プログラムのフローチャートおよび疑似コード
- コード検査から出たプログラムに関するコメント
- プログラム・フローの書き込み説明
- 使用した呼び出しシーケンスを選択した理由に関する以下のような情報
  - DFSDDLT0 を使用して呼び出しシーケンスをテストしましたか?
  - 同じ結果を出す呼び出しの組み合わせが複数ある場合に、なぜ使用したシーケンスを選択したのですか?
  - 他のシーケンスは何でしたか? DFSDDLT0 を使用して、そのシーケンスをテストしましたか?
- プログラムの構造化中、あるいはコーディング中に起こった問題すべて
- プログラムのテスト中に起こった問題すべて
- プログラム内で変更してはいけないものに関する警告

この情報はすべて、プログラムの構造化およびコーディングに関連しています。また、エンド・ユーザー向けの文書をプログラマー向けの文書にも組み込んでください。

最後に、必要とする詳細のレベル、およびプログラムの文書化に最適な形式を決定する必要があります。以下の文書化のガイドラインは、提案として示すものです。

エンド・ユーザー用の文書

アプリケーションの設計の文書化に加えて、プログラムの使用方法に関する情報を記録する必要もあります。

ユーザーが必要とする情報の量、およびその中でユーザーが提供する必要がある情報の量は、プログラムのユーザーと、プログラムのタイプによって異なります。

少なくとも、プログラムのユーザーのために以下の情報を記録してください。

- プログラムを使用する際にユーザーに必要な事柄。以下に例を示します。
  - オンライン・プログラムのパスワードがあるかどうか。
  - バッチ・プログラムの場合は、必須な JCL は何か。
- ユーザーがプログラムに対して提供する必要がある入力データ。以下に例を示します。
  - MPP の場合、画面を最初に形式設定するためにユーザーが入力する MOD 名は何か。
  - CICS オンライン・プログラムの場合、ユーザーが入力しなければならない、CICS トランザクション・コードは何か。予定されている端末入力は何か。
  - バッチ・プログラムの場合、入力形式は、テープか、あるいはディスク・データ・セットか。その入力は元々、直前のジョブから出力されたものか。
- プログラムの出力の内容と形式。以下に例を示します。
  - 報告書の場合は、形式を表示するか、サンプル・リストを示してください。
  - オンラインのアプリケーション・プログラムについては、画面がどのように見えるのかを示してください。
- オンライン・プログラムについては、ユーザーが判断を下す必要がある場合、各判断に含まれる事柄を説明してください。選択項目およびデフォルトを示してください。

プログラムを使用するユーザーが端末装置に精通していない場合、前述の事項に加えて、何らかのユーザーの手引きが必要になります。例えばこれらの手引きでは、端末装置の使用方法およびそのプログラムでできることを明示的に指示してください。これらの手引きには、タスクまたはプログラムが異常終了した場合に取るべき処置、そのプログラムを再始動すべきかどうか、またはデータベースのリカバリーを行う必要があるかどうかについての説明が記載されている必要があります。この種の情報を提供する責任がなくても、この情報に責任のある人々に、アプリケーションに固有な何らかの情報を知らせてください。

第 2 部 IMS DB 用のアプリケーション・プログラミング

IMS では、IMS データベースにアクセスするアプリケーション・プログラムの作成がサポートされています。

第 11 章 IMS DB 用のアプリケーション・プログラムの作成

高水準アセンブラー言語、C 言語、COBOL、Java、Pascal、および PL/I で、IMS DB 内のデータにアクセスするアプリケーション・プログラムを作成できます。

関連概念:

693 ページの『第 38 章 Java 開発用の IMS ソリューションの概要』

プログラミングのガイドライン

プログラムが出す IMS 要求の数、タイプ、および順序は、プログラムの効率に影響を与えます。設計の劣ったプログラムであっても、正しくコーディングされていれば実行が可能です。 IMS では設計の誤りは検出されません。以下のヒントは、アプリケーション・プログラムをできるだけ効率の良い設計で開発するのに役立ちます。

プログラムで出される呼び出しの一般的な順序が決まったならば、順序に関するガイドラインを参照して、改善の余地がないか調べてください。効率の良い順序で要求を出すと、IMS の内部処理の効率が向上します。プログラムを書くときには、このセクションで説明するガイドラインに留意してください。以下のリストは、効率がよく、エラーのないプログラムを作成するのに役立つ、プログラミングのガイドラインを示します。

- 最も単純な呼び出しを使用してください。IMS による検索の範囲を狭くするために、要求を修飾してください。
- 必要なセグメントに IMS が達するためのパスが最も短くなるような要求または要求のシーケンスを使用してください。
- 使用する要求の数をできるだけ少なくしてください。プログラムで DL/I 呼び出しを出すだびに、システムの時間とリソースが使用されます。以下を実行することにより、不必要な呼び出しをなくすことができます。
  - 同じパス内の 2 つ以上のセグメントの置き換え、リトリーブ、または挿入を行う場合には、パス要求を使用してください。これを行うために複数の要求を使用している場合は、不必要な要求を出していることになります。
  - 順序列を変更して、ユーザーのプログラムが、別の入出力域にセグメントを保管し、それ以降にそのセグメントが必要になる時にはその入出力域からセグメントを得られるようにしてください。プログラム実行中に同じセグメントが 2 回以上リトリーブされるような場合には、不必要な要求が出されていることを意味します。
  - GB、GE、および II 状況コードが出される結果となるような、不必要で非生産的な要求を見越して取り除いてください。 例えば、特定のセグメント・タイプについて GN 呼び出しを出し、そのセグメント・タイプのオカレンスがいくつあるかが分かっている場合には、GE 状況コードが戻されるような GN 呼び出しは出さないでください。プログラムがリトリーブしたオカレンスの数を記録しておき、そのセグメント・タイプのオカレンスがすべてリトリーブされたことが分かったときには、別の処理に進んでください。

– 親が存在することを確認するために、親について Get 要求を出す代りに、それぞれの親についての修飾を伴う挿入要求を出してください。IMS が GE 状況コードを戻したときには、少なくとも 1 つの親が存在していないことになります。セグメントを挿入する場合、親セグメントが存在しなければ従属セグメントを挿入することはできません。

- 定期的に更新をコミットしてください。IMS では、コミットされていない更新を持つことができる全機能データベースが一度に 300 個までに制限されています。この制限には、論理的に関連するデータベース、副次索引、および HALDB 区画もカウントされます。コミットされていない更新についての 300 というデータベース制限に近づく最も一般的な理由は、HALDB データベース内の区画数です。PROCOPT 値で BMP アプリケーションによるデータベース内のセグメントの挿入、置換、または削除を許可する場合は、その BMP アプリケーションが、合計で 300 を超えるデータベースと HALDB 区画の更新を、変更をコミットすることなく行わないようにしてください。
- プログラム論理の主要セクションは一緒にしておいてください。 例えば、エラー処理や印刷などを行う場合には、主要セクションの中で分岐して通常の処理を続けるのではなく、プログラムの主要セクション以外の部分にあるエラー・ルーチンや印刷ルーチンなどの条件ルーチンに分岐するようにしてください。
- データの物理的な配置を効率的に使用する呼び出し順を使用してください。 セグメントへのアクセスは、できる限り階層シーケンスに行い、階層内での逆方向への移動を回避してください。
- データベース・レコードは、ルート・セグメントのキー・フィールド順に処理してください。 (HDAM および PHDAM データベースの場合、この順序は使用されるランダム化ルーチンによって異なります。この情報については、DBA にお尋ねください。)
- データベース構造への依存度が高くなる方法で、プログラム論理およびコマンドまたは呼び出しの構造を組み立てることは避けてください。階層の現行構造によっては、プログラムの柔軟性が損なわれます。
- ユーザーのプログラムがロックするセグメント数を最小限にとどめてください。ユーザーのプログラムが使用するそれぞれの PCB について、更新済みセグメントのロックおよび現行データベース・レコードのロックを解放するためにチェックポイントを取る必要が出てくる場合があります。ユーザーのプログラムで PCB が使用される度に、共用レベルまたは更新レベルで現行データベース・レコードがロックされます。このロックが必要なくなった場合は、GU 呼び出しを、「より大」演算子をキー X'FF' (高い値) に使用し、ルート・レベルで修飾して発行することで、新規のロックを獲得せずに現行のロックを解放します。

  データベースの終わりに高い値をおくランダマイザーを使用する場合、および副次索引を使用する場合は、最小化技法を使用しないでください。想定される高い値のキーを超える別のルートを使用する場合、IMS は GE を戻し、アプリケーションが次のステップを決定できるようにします。副次索引は機能しない可能性があります。これは、階層構造が反転するためです。キーは索引内の最後のルートを過ぎていますが、データベース内の最後のルートは過ぎていません。

  獲得 (G) 処理オプションを指定して PCB を使用すると、その PCB については共用レベルでロックされます。これにより、獲得処理オプションを使用する他のプログラムが、同じデータベース・レコードに並行してアクセスできるようにな

ります。更新を許す処理オプション (I、R または D) を指定した PCB を使用すると、その PCB については更新レベルでロックされます。この場合には、他のプログラムは同じデータベース・レコードに並行してアクセスできません。

関連概念:

334 ページの『ユーザー・プログラムの排他使用に対するセグメントの予約』

セグメント検索引数 (SSA)

セグメント検索引数 (SSA) は、IMS が DL/I 呼び出しの処理に使用する情報を指定します。SSA に指定されているフィールドのデータ型に関係なく、SSA はフィールドをバイナリー形式として扱い、バイナリー比較を行います。

1 つ以上の SSA を使用する DL/I 呼び出しを修飾呼び出し といい、SSA を使用しない DL/I 呼び出しを非修飾呼び出し といいます。

非修飾 **SSA**
: セグメント名だけが含まれます。

修飾 **SSA**
: セグメント・オカレンスの名前を指定する 1 つ以上の修飾ステートメントを含みます。修飾ステートメントの代りに、C コマンドとセグメント・オカレンスを連結したキーを使用できます。

SSA を使用してセグメントを名前で選択したり、特定のセグメントに関する検索基準を指定したりできます。特定のセグメントを記述するには、DL/I 呼び出しに修飾ステートメントを追加します。コマンド・コードを使用すると、呼び出しをさらに修飾できます。

非修飾 SSA

非修飾 SSA には、アクセスするセグメント・タイプの名前を指定します。非修飾 SSA では、セグメント名フィールドは 8 バイトであり、その後に 1 バイトのブランクを続けなければなりません。実際のセグメント名が 8 バイトよりも短い場合には、右側にブランクを埋め込む必要があります。非修飾 SSA の例を次に示します。

```
PATIENTbb
```

修飾 SSA

SSA を修飾するには、フィールドまたは仮想子のシーケンス・フィールドのどちらかを使用することができます。修飾された SSA は、アクセスするセグメント・オカレンスを記述します。この記述は修飾ステートメントと呼ばれ、3 つの部分から構成されます。次の表は、修飾された SSA の構造を示しています。

表 *31.* 修飾された *SSA* の構造

SSA コンポーネント	フィールド長
セグメント名	8
(	1
フィールド名	8
比較演算子	2

表 *31.* 修飾された *SSA* の構造 *(*続き*)*

SSA コンポーネント	フィールド長
フィールド値	可変
)	1

修飾ステートメントを使用すると、検索対象の特定セグメント・オカレンスに関する情報を IMS に提供できます。これは、セグメント内のあるフィールドの名前、およびそのフィールドの中から探し出したい値を IMS に対して指定することによって実行します。フィールドと値を関係演算子 (前の表の `R.O.`) により接続し、この関係演算子がどのようにその 2 つを比較するかを IMS に伝えます。例えば、PATNO フィールドが 10460 である PATIENT セグメントにアクセスする場合には、次のような SSA を使用できます。

```
PATIENTb(PATNObb=b10460)
```

あるいは、DL/I 呼び出しがコマンド・コード O を使用する場合、8 バイトのフィールド名の代わりに、4 バイトの開始オフセット位置と 4 バイトのデータ長を使用することができます。開始オフセットは、物理セグメント定義に対する相対値で、1 から始まります。リトリーブ可能な最大長は、データベース・タイプの最大セグメント・サイズで、最小長は 1 です。'*oooollll*' の形式で 2 つのフィールドが指定されます。*oooo* がオフセット位置、*llll* がリトリーブしたいデータの長さです。この方法を使用して、フィールド定義がなくてもデータのリトリーブおよびリトリーブを行うことができます。

修飾ステートメントは括弧で囲みます。 1 番目のフィールドには、セグメントの検索で IMS に使用させるフィールドの名前 (前の表の**フィールド名**) が入ります。 2 番目のフィールドには、関係演算子が入ります。関係演算子には、次のいずれかを使用します。

- 等しい (次のように表します)

  =b

  b=

  EQ
- より大 (次のように表します)

  >b

  b>

  GT
- より小 (次のように表します)

  <b

  b<

  LT
- 以上 (次のように表します)

  >=

  =>

  GE
- 以下 (次のように表します)

<=

=<

LE

- 等しくない (次のように表します)

¬=

=¬

NE

3 番目のフィールド (前の表の**フィールド値**) には、比較値として IMS に使用させる値が入ります。 3 番目のフィールド (**フィールド値**) の長さは、**フィールド名** で指定したフィールドと同じ長さでなければなりません。

1 つの SSA で複数の修飾ステートメントを使用できます。順序フィールドが複数のフィールドから成る場合の仮想論理子セグメントのように、特殊な場合もあります。

仮想論理子のシーケンス・フィールド

一般に、1 つのセグメントに含まれるのは、1 つのシーケンス・フィールドのみです。ただし、仮想論理子セグメント・タイプの場合には、非連続順序フィールドを定義するために複数の FIELD ステートメントを使用できます。

仮想論理子セグメントのシーケンス・フィールドを指定するときに、そのフィールドが連続していない場合には、SSA で指定されるフィールドの長さは、指定のフィールドの長さに後続のすべてのシーケンス・フィールドの長さを加えた値になります。次の図は、非連続シーケンス・フィールドがあるセグメントを示しています。

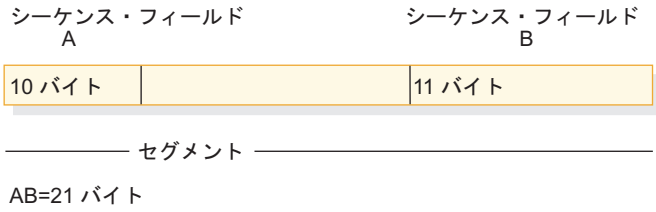


図 47. 非連続シーケンス・フィールドがあるセグメント

最初のシーケンス・フィールドが SSA の「分散した」シーケンス・フィールドに含まれていない場合には、IMS は引数を、シーケンス・フィールドとしてではなくデータ・フィールド仕様として扱います。

関連資料: 仮想論理子セグメントの詳細については、「*IMS V15* データベース管理」を参照してください。

関連概念:

799 ページの『SSAList インターフェースを使用したセグメント検索引数の指定』

SSA のガイドライン

IMS の検索に与える情報が多くなるほど、プログラム内でセグメントを分析および比較するために必要なプログラム論理が少なくてすむため、 SSA を使用するとプログラミングを単純化できます。

SSA を使用しても、特定のセグメントを獲得するために必要な内部論理や入出力などのオーバーヘッドは、必ずしも減少するとは限りません。 SSA を使用せずに特定のセグメントを見つけるために、DL/I 呼び出しを出して、希望するセグメントが見つかるまでキー・フィールドを検査するプログラム論理を組み込むことができます。 DL/I 呼び出しで SSA を使用すると、DL/I 呼び出しを行う回数、およびキー・フィールドを検査するために必要なプログラム論理を減少させることができます。SSA を使用すると、ユーザーの代わりに IMS がこれらのことを実行します。

推奨事項:

- 修飾された呼び出しは、できる限り、修飾された SSA とともに使用してください。 SSA がフィルターとして働き、ユーザーのプログラムが必要としているセグメントだけを戻します。これにより、プログラムが出す呼び出しの数が減少し、パフォーマンスが向上します。また、ユーザーのプログラムの文書化にも役立ちます。特に、挿入呼び出しを使用してセグメントを追加するときには、修飾された SSA は便利です。 SSA を使用することにより、希望する個所にセグメントが挿入されるようになります。
- ルート・セグメントの場合には、可能であれば、キー・フィールドと等号関係演算子を指定してください。「等号」演算子、「以上」演算子、または「より大」演算子を指定したキー・フィールドを使用すると、IMS はルート・セグメントに直接移動します。
- 従属セグメントの場合、ルート・レベルのときほどキー・フィールドは重要ではありませんが、SSA でキー・フィールドを使用することをお勧めします。キー・フィールドと等号演算子を使用すると、IMS はそのキーよりも高いキー値が検出されたときにそのレベルで検索を停止します。それ以外の場合、IMS は、特定のセグメントが存在するかどうかを判別するために、設定された親セグメントのもとにあるセグメント・タイプのすべてのオカレンスを検索しなければなりません。
- キー・フィールド以外のフィールドを使用して何度もセグメントを検索する必要がある場合には、フィールドに副次索引を含めるようにしてください。

例えば、「Ellen Carter」という氏名の患者のレコードを検索する場合を考えてみます。この例の患者セグメントは、キー・フィールドである患者番号、患者名、および患者の住所の 3 つのフィールドから構成されていることに注意してください。患者番号がキー・フィールドになっているため、IMS は患者番号の順に患者セグメントを保管します。「Ellen Carter」という患者に関するレコードを得るための最良の方法は、SSA にこの人の患者番号を指定することです。この人の番号が 09000 である場合、次のような呼び出しと SSA を使用してください。

```
GUbbbbbbPATIENTb(PATNObbb=b09000)
```

プログラムが無効な番号を指定した場合、または誰かがデータベースから Ellen Carter のレコードを削除した場合、IMS はすべての PATIENT オカレンスを検索しなくてもそのセグメントが存在しないことを判別できます。

ただし、番号が分からないために、代わりに名前を指定した場合には、IMS はすべての患者セグメントを検索し、「Ellen Carter」が検出されるまで、あるいは患者セグメントの終わりに達するまで、各患者名フィールドを読み取る必要があります。

関連概念:

337 ページの『第 18 章 副次索引付けおよび論理関係』

複数の修飾ステートメント

修飾ステートメントを使用するときには、データベース内のセグメントのフィールドを比較するためのフィールド値を IMS に提供する以外のことも行えます。すなわち、IMS に比較させたいフィールドの限界を設定するために、いくつかのフィールド値を指定できます。

1 つの呼び出しでは、最大 1024 までの修飾ステートメントを使用できます。

ブール演算子の 1 つで修飾ステートメントを結合します。 IMS に、例えば次のような値を探すように示すことができます。 A より大きくしかも B より小さい値。A と等しいかまたは B より大きい値。ブール演算子には、次のものがあります。

論理 **AND**
: この要求を満足させるセグメントは、論理 AND (* または & でコーディング) で結合された両方の修飾ステートメントを満足させるものでなければなりません。

論理 **OR**
: この要求を満足させるセグメントは、論理 OR (+ または | でコーディング) で結合された修飾ステートメントのうちのいずれを満足させるものであってもかまいません。

さらに、独立 AND と呼ばれるもう 1 つのブール演算子があります。これは、副次索引にのみ使用されます。

複数の修飾ステートメントを満足させるセグメントは、それらの修飾ステートメントのセットを満足させるものでなければなりません。このセットとは、AND によって結び付けられている複数の修飾ステートメントのことをいいます。セグメントがセットを満足させるためには、そのセット内の各修飾ステートメントを満足させる必要があります。それぞれの OR 演算子より、新しい修飾ステートメントのセットが始まります。複数の修飾ステートメントを処理する場合、IMS はそれらを左から右への順で読み取り、その順序でステートメントを処理します。

ルート・セグメントに関して複数の修飾ステートメントを指定した場合、修飾ステートメント内で指名したフィールドによって、呼び出しの条件を満たすために IMS が調べるルートの範囲が影響を受けます。 DL/I は、修飾ステートメントを調べて、受け入れ可能な最小のキー値を判別します。

1 つ以上のセットの中に、「等しい」、「より大きい」、または「より大きいまたは等しい」演算子でキー・フィールドが修飾されているステートメントが 1 つも含まれていない場合、IMS はデータベースの最初のルートから検索を開始し、修飾の条件を満たすルートを探します。

それぞれのセットの中に、「等しい」、「より大きい」、または「より大きいまたは等しい」演算子でキー・フィールドが修飾されているステートメントが少なくと

も 1 つは含まれている場合、IMS はそれらのキーのうち最小のものを検索開始位置として使用します。検索の開始位置を設定した後で、IMS は GN 呼び出しを処理する場合と同じように、データベース内を順番に順方向検索を行って、呼び出しを処理します。IMS は検出したそれぞれのルートを調べ、そのルートが修飾ステートメントのセットの条件を満たせるかどうかを判別します。IMS は、修飾ステートメントを調べて、受け入れ可能な最大のキー値を判別します。

1 つ以上のセットの中に、「等しい」、「以下」、または「より小さい」演算子でキー・フィールドが修飾されているステートメントが 1 つも含まれていない場合、IMS は、最大キー値が存在しないものと判定します。それぞれのセットの中に、「等しい」、「より小さい」、または「以下」演算子でキー・フィールドが修飾されているステートメントが少なくとも 1 つは含まれている場合、IMS はこれらのキーのうちの最大キーを使用して、検索停止位置を判別します。

IMS は、呼び出しの条件と満たすか、データベースの終わりに達するか、あるいは最大値を超えるキー値を検出するまで、検索を続けます。最大キー値がない場合には、IMS が呼び出しの条件を満たすか、あるいはデータベースの終わりに達するまで、検索が続けられます。

例: 次に示すのは、ルート・レベルで使用される SSA の例です。

```
ROOTKEYb
=b10&FIELDBb
b=XYZ+ROOTKEYb
=10&FIELDBb
b
=ABC
```

この場合、最小キーと最大キーは 10 になります。すなわち、IMS はキー 10 から検索を開始し、10 を超える最初のキーを検出したときに検索を停止します。この SSA の条件を満たすには、ROOTKEY フィールドが 10 に等しく、FIELDB が ABC または XYZ のいずれかと等しくなければなりません。

```
ROOTKEYb
=>10&ROOTKEYb
<20
```

この場合、最小キーは 10 で、最大キーは 20 です。10 から 20 までの範囲内のキーが SSA を満足させることになります。IMS は、20 を超える最初のキーを検出したときに、検索を停止します。

```
ROOTKEYb
10&ROOTKEYb
=<20+ROOTKEYb
=>110&ROOTKEYb
=<120
```

この場合、最小キーは 10 で、最大キーは 120 です。10 から 20 までの範囲のキーと、110 から 120 までの範囲内のキーが呼び出しを満足させることになります。IMS は、120 を超える最初のキーを検出したときに、検索を停止します。IMS は、20 から 110 までの範囲はスキャンしないで、(HIDAM または PHIDAM の検索を使用して) 20 から 110 にスキップします。このように範囲を指定すると、プログラム操作の効率が良くなります。

論理関係の一部である複数の修飾ステートメントのセグメントを使用する場合には、このほかにも考慮事項があります。

関連概念:

338 ページの『副次索引を使用した複数の修飾ステートメント』

複数の修飾ステートメントの使用法の例

ここでは、複数の修飾ステートメントの使用法を例示します。

例えばサンプルの医療データベースから、以下の質問に答えを出したいとします。

> 1992 年中に患者番号 04120 の患者が来院したか?

この質問への答えを得るには、患者名に加えてさらに多くの情報を IMS に付与する必要があります。この場合、IMS に、ILLNESS セグメントからそれらの患者を検索し、それぞれを読み取り、日付が 1992 年内のものを返すように指示します。そのために、次のような呼び出しを出します。

```
GU   PATIENTb(PATNObbbEQ04120)
     ILLNESSb(ILLDATEb>=19920101&ILLDATEb<=19921231)
```

すなわち、患者番号 04120 のもとにある ILLNESS セグメント・オカレンスのうち、1992 年 1 月 1 日以降で、しかも 1992 年 12 月 31 日以前の日付をもつもの (AND 結合子で両方の修飾を結び付けます) を、IMS に戻させる必要があります。次の質問に答える必要があるものとします。

> 1992 年の 1 月中または 1992 年の 7 月中に Judy Jennison が来院したか?彼女の患者番号は 05682 である。

次のような SSA を使用した GU 呼び出しを出すこともできます。

```
GU   PATIENTb(PATNObEQ05682)
     ILLNESSb(ILLDATEb>=19920101&ILLDATEb<=19920131|ILLDATEb>=19920701&ILLDATEb<=19920731)
```

この要求を満足させるためには、ILLDATE の値が 2 つのセットのうちのいずれかを満足させる必要があります。IMS は、 1992 年の 1 月または 1992 年 7 月のいずれかに関する ILLNESS セグメント・オカレンスがあれば、それを戻します。

HDAM、PHDAM、DEDB 用の複数の修飾ステートメント

HDAM (階層直接アクセス方式)、PHDAM (区分 HDAM)、または DEDB (高速処理データベース) 編成の場合、ランダム化出口ルーチンは、通常、ルート・キーを昇順キー配列では保管しません。これらの編成でも、IMS は、最小キー値および最大キー値を判別します。この最小キー値がランダム化出口ルーチンに渡され、このルーチンで開始アンカー・ポイントが判別されます。

このアンカー以降の最初のルートが検索の開始点になります。IMS が最大キー値を超えるキーを検出すると、IMS は GE 状況コードを戻して検索を終了します。ランダム化ルーチンによるランダム化の結果、キーがキーの昇順で保管されるようになっている場合、キーの範囲を指定した呼び出しを出すと、その範囲内のすべてのキーが戻されます。ただし、ランダム化ルーチンによってキー・シーケンスにランダム化されていない場合には、この呼び出しで、要求された範囲内のキーがすべて戻されるとは限りません。 したがって、HDAM、PHDAM、DEDB 編成の場合、キー値の範囲を指定した呼び出しは、キーが昇順になっている場合にのみ使用してください。

推奨事項:

- HDAM、PHDAM、DEDB 編成の場合、キー値の範囲を指定した呼び出しは、キーが昇順になっている場合にのみ使用してください。
- HDAM、PHDAM、または DEDB 編成の場合には、ルート・レベルでの値の範囲を指定した呼び出しを使用しないでください。

推奨されてはいませんが、GN/GHN データベース呼び出しを行う際に、コマンド・コード A および G を使用してデータベースの逐次探索を実行できます。コマンド・コード A は、位置情報をクリアし、呼び出しがデータベースの先頭から開始できるようにします。コマンド・コード G は、値の範囲をルート・レベルで指定する SSA と共に使用すると、ランダム化を抑止してデータベースの逐次探索が行われるようにします。返されるセグメントは、ランダム化の方法によっては順次方式で配置されていない場合があります。

データベースを順次検索するには、以下のセグメント検索引数 (SSA) を、ルート・レベルで値の範囲を指定する SSA と一緒に使用します。

```
key field > hex zeros & key field < all f's key
```

返されるセグメントは、ランダム化の方法によっては順次方式で配置されていない場合があります。

HDAM または PHDAM データベースの詳細については、 *IMS V15* データベース管理 を参照してください。

SSA とコマンド・コード

SSA は、また、1 つ以上のコマンド・コードを含むことができます。コマンド・コードは DL/I 呼び出しの機能を変更もしくは拡張します。

コマンド・コードについては、「*IMS V15* アプリケーション・プログラミング *API*」のトピック『DL/I 呼び出し用の汎用コマンド・コード』を参照してください。

IMS は、常にパス中最低のセグメントを入出力域に返します。SSA に D というコマンド・コードをコーディングすると、IMS はその SSA によって記述されたセグメントも戻します。D コマンド・コードを使用する呼び出しは、パス *CALL* と呼ばれます。

例えば、次の図に示す階層構造で、セグメント F と、F に至るパス内のすべてのセグメントをリトリーブするために、プログラムに GU 呼び出しの D コマンド・コードをコーディングする場合を考えてみます。

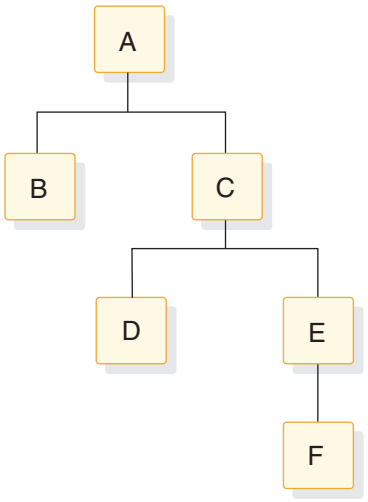


図 *48. D* コマンド・コードの例

この呼び出しの機能とその SSA は、次のようになります。

```
GU   Abbbbbbb
     *D
     Cbbbbbbb
     *D
     Ebbbbbbb
     Fbbbbbbb
```

コマンド・コードは 1 文字です。 SSA 内のセグメント名フィールドのあとに、このコマンド・コードをコーディングしてください。セグメント名フィールドとコマンド・コードは、次の表で示すとおりにアスタリスクで区切ってください。

表 *32.* コマンド・コードを使用した非修飾 *SSA*

SSA コンポーネント	フィールド長
セグメント名	8
*	1
コマンド・コード	可変
b	1

コマンド・コードは、修飾された SSA でも、非修飾 SSA でも使用できます。ただし、MSDB 呼び出しでコマンド・コードを使用することはできません。コマンド・コードの後に修飾ステートメントを続けない場合には、1 バイトのブランクを設ける必要があります。コマンド・コードの後に修飾ステートメントを続ける場合には、ブランクは使用しないでください。その代わりに、次の表に示すように、コマンド・コードの後に修飾ステートメントの左括弧を続けてください。

表 *33.* コマンド・コードを使用した修飾 *SSA*

SSA コンポーネント	フィールド長
セグメント名	8
*	1
コマンド・コード	可変
(	1

表 33. コマンド・コードを使用した修飾 SSA (続き)

SSA コンポーネント	フィールド長
フィールド名、あるいは O コマンド・コードが指定されている場合は、フィールド名か、フィールドの位置と長さ	8
関係演算子 (R.O.)	2
フィールド値	可変
)	1

セグメント内のフィールドの位置、および探しているフィールドの値を IMS に指定することで、フィールドの位置と値が関係演算子によって関係付けられます。関係演算子は、この 2 つをどのように比較するかを IMS に伝えます。フィールドの位置は、DBD で定義されている検索可能なフィールド名で指定することも、コマンド・コード O を使用した場合の位置および長さで指定することも可能です。

コマンド・コードを使用して DEDB 内のサブセット・ポインターを管理する場合には、コマンド・コードの直後にサブセット・ポインターの番号をコーディングしてください。サブセット・ポインターは、同じ親のもとにあるセグメント・オカレンスのチェーンを 2 つ以上のグループ、すなわちサブセットに分割するための手段です。どのセグメント・タイプについても、8 つまでのサブセット・ポインターを定義できます。アプリケーション・プログラムを使用して、これらのサブセット・ポインターを管理できます。

関連概念:

380 ページの『サブセット・ポインター・コマンド・コードを使用した高速機能 DEDB の処理』

DL/I 呼び出しとデータ域のコーディングの考慮事項

プログラムに関する設計上の決定をすべて行った後で、コーディングによってその決定が具体化されます。プログラムの設計と処理論理を知る以外に、プログラムが処理するデータ、プログラムが参照する PCB、およびプログラムが処理する階層内のセグメント形式を知っておく必要があります。

次のリストをチェックリストとして利用し、脱落している情報がないかを調べることができます。データ、アプリケーション・プログラムで使用される IMS オプション、セグメント・レイアウト、およびアプリケーション・プログラムのデータ構造に関する情報が脱落している場合には、インストール先における DBA あるいはそれに相当するスペシャリストからその情報を得てください。インストール先で確立されているプログラミングの標準と規則にも留意してください。

プログラムの設計の考慮事項

- プログラムで使用する呼び出しの順序
- 各呼び出しの形式
  - 呼び出しに SSA が含まれるか
  - 含まれる場合には、SSA は修飾されているか非修飾か
  - 呼び出しにコマンド・コードが含まれるか

- プログラムの処理論理
- 各呼び出しのあとで状況コードをチェックするためにプログラムが使用するルーチン
- プログラムが使用するエラー・ルーチン

チェックポイントの考慮事項

- 使用するチェックポイント呼び出しのタイプ (基本またはシンボリック)
- 基本チェックポイント呼び出しであるか、あるいはシンボリック・チェックポイント呼び出しであるかにかかわらず、各チェックポイント呼び出しに割り当てられる識別番号
- シンボリック・チェックポイント呼び出しを使用する場合には、プログラムのどの区域のチェックポイントを取るのか

セグメントの考慮事項

- そのセグメントが固定長か可変長か
- セグメントの長さ (可変長セグメントの場合には最大長)
- 各セグメントに入るフィールドの名前
- このセグメントにキー・フィールドがあるかどうか。キー・フィールドがある場合、固有か非固有か。キー・フィールドがない場合には、どのような順序付け規則が定義されているのか。 (セグメントのキー・フィールドは、DBD 内の FIELD ステートメントの SEQ キーワードで定義されます。順序付け規則は、DBD 内の SEGM ステートメントの RULES キーワードで定義されます。)
- セグメントのフィールド・レイアウト
  - 各フィールドのバイト位置
  - 各フィールドの長さ
  - 各フィールドの形式

データ構造の考慮事項

- プログラムが処理するデータ構造は、DB PCB で定義されています。プログラムが参照するすべての PCB は、アプリケーション・プログラムの PSB の一部です。PSB 内で PCB がどのような順序で定義されているのかを知っておく必要があります。
- プログラムが処理する各データ構造のレイアウト
- それぞれのデータ構造ごとに、複数位置付けまたは単一位置付けが指定されているかどうか。これは、PSB の生成時に PCB ステートメントの POS キーワードで指定されます。
- 複数 DB PCB を使用するデータ構造があるかどうか

CICS DL/I 呼び出し側プログラムを実行する準備

CICS DL/I 呼び出し側プログラムを実行する前に、いくつかのステップを行わなければなりません。

該当する CICS 参照情報を参照してください。

- CICS オンライン・プログラムの変換、コンパイル、およびバインドについては、「*CICS Transaction Server for z/OS CICS* システム定義ガイド」のアプリケーション・プログラムのインストールに関する説明を参照してください。
- CICS オンライン・プログラムで使用するコンパイラー・オプションと、DL/I 呼び出しを使用する CICS オンライン COBOL プログラムを Enterprise COBOL に変換する際の CICS の考慮事項については、「*CICS Transaction Server for z/OS CICS* アプリケーション・プログラミング・ガイド」を参照してください。

DL/I 呼び出しとデータ域のコーディング方法の例

DL/I 呼び出しおよびデータ域は、アセンブラー言語、C、COBOL、Pascal、Java、および PL/I でコーディングできます。

アセンブラー言語でのバッチ・プログラム・コーディング

次のサンプル・コードは、IMS データベースにアクセスする IMS プログラムをアセンブラー言語で作成する方法を示しています。

プログラムの右側の数字は、このプログラムの後の注の番号に対応しています。この種のプログラムは、バッチ・プログラム、またはバッチ指向 BMP として実行できます。

アセンブラー言語プログラムのサンプル

```
PGMSTART CSECT                                                          NOTES
*               EQUATE REGISTERS                                         1
*   USEAGE OF REGISTERS
R1       EQU   1              ORIGINAL PCBLIST ADDRESS
R2       EQU   2              PCBLIST ADDRESS1
R5       EQU   5              PCB ADDRESSS
R12      EQU   12             BASE ADDRESS
R13      EQU   13             SAVE AREA ADDRESS
R14      EQU   14
R15      EQU   15
*
         USING PGMSTART,R12   BASE REGISTER ESTABLISHED                  2
         SAVE  (14,12)        SAVE REGISTERS
         LR    12,15          LOAD REGISTERS
         ST    R13,SAVEAREA+4 SAVE AREA CHAINING
         LA    R13,SAVEAREA   NEW SAVE AREA
         USING PCBLIST,R2     MAP INPUT PARAMETER LIST
         USING PCBNAME,R5     MAP DB PCB
         LR    R2,R1          SAVE INPUT PCB LIST IN REG 2
         L     R5,PCBDETA     LOAD DETAIL PCB ADDRESS
         LA    R5,0(R5)       REMOVE HIGH ORDER END OF LIST FLAG         3
         CALL  ASMTDLI,(GU,(R5),DETSEGIO,SSANAME),VL                     4
*
*
         L     R5,PCBMSTA     LOAD MASTER PCB ADDRESS
         CALL  ASMTDLI,(GHU,(R5),MSTSEGIO,SSAU),VL                       5
*
*
         CALL  ASMTDLI,(GHN,(R5),MSTSEGIO),VL                            6
*
*
         CALL  ASMTDLI,(REPL,(R5),MSTSEGIO),VL
*
*
         L     R13,4(R13)     RESTORE SAVE AREA
```

```
         RETURN (14,12)     RETURN BACK                                    7
*
*     FUNCTION CODES USED
*
GU       DC    CL4'GU'
GHU      DC    CL4'GHU'
GHN      DC    CL4'GHN'
REPL     DC    CL4'REPL'                                                   8
*
*       SSAS
*
SSANAME  DS    0C
         DC    CL8'ROOTDET'
         DC    CL1'('
         DC    CL8'KEYDET'                                                 9
         DC    CL2' ='
NAME     DC    CL5'  '
         DC    C')'
*
SSAU     DC    CL9'ROOTMST'*
MSTSEGIO DC    CL100' '
DETSEGIO DC    CL100' '
SAVEAREA DC    18F'0'
*                                                                         10
PCBLIST  DSECT
PCBIO    DS    A              ADDRESS OF I/O PCB
PCBMSTA  DS    A              ADDRESS OF MASTER PCB
PCBDETA  DS    A              ADDRESS OF DETAIL PCB                       11
*
PCBNAME  DSECT
DBPCBDBD DS    CL8            DBD NAME
DBPCBLEV DS    CL2            LEVEL FEEDBACK
DBPCBSTC DS    CL2            STATUS CODES
DBPCBPRO DS    CL4            PROC OPTIONS
DBPCBRSV DS    F              RESERVED
DBPCBSFD DS    CL8            SEGMENT NAME FEEDBACK
DBPCBMKL DS    F              LENGTH OF KEY FEEDBACK
DBPCBNSS DS    F              NUMBER OF SENSITIVE SEGMENTS IN PCB
DBPCBKFD DS    C              KEY FEEDBACK AREA
         END   PGMSTART
```

注:

1. アセンブラー言語プログラムへのエントリー・ポイントは、任意の名前にできます。また、どの呼び出しでも、ASMTDLI の代わりに CBLTDLI を使用できます。
2. IMS がアプリケーション・プログラムに制御を渡すときには、レジスター 1 には、可変長フルワード・パラメーター・リストのアドレスが入っています。このリストの各ワードには、アプリケーション・プログラムが保管する必要のある PCB のアドレスが入っています。パラメーター・リストの最後のワードの最高位バイトの 0 ビットは、リストの終わりを示すために値 1 に設定されます。アプリケーション・プログラムは、その後 DL/I 呼び出しを実行する時に、これらのアドレスを使用します。
3. プログラムは、DETAIL DB PCB のアドレスをロードします。
4. プログラムは、修飾した SSA (SSANAME) を使用して DETAIL データベースに対する GU 呼び出しを出します。
5. プログラムは、HALDB マスター PCB のアドレスをロードします。
6. プログラムが出す次の 3 つの呼び出しは、HALDB マスターを対象としています。最初の呼び出しは、非修飾 SSA を使用する GHU 呼び出しです。2 番目

の呼び出しは、非修飾の GHN 呼び出しです。REPL 呼び出しは、GHN 呼び出しでリトリーブされたセグメントを MSTSEGIO 域内のセグメントによって置き換えます。

サンプル状況コード・エラー・ルーチンに対する呼び出しを除き、アセンブラー言語の DL/I 呼び出しでは、VL パラメーターの代わりに *parmcount* パラメーターを使用できます。

7. RETURN ステートメントは IMS レジスターをロードし、IMS に制御を戻します。
8. 呼び出し機能は 4 文字の定数として定義されます。
9. プログラムは SSA の各部分を個別に定義し、SSA のフィールドを修正できるようにします。
10. プログラムでは、リトリーブまたは挿入される最長セグメント (あるいは、プログラムが D コマンド・コードを使用する場合には、最長のセグメント経路) を収めるのに十分な大きさの入出力域を定義する必要があります。このプログラムの入出力域は、いずれも 100 バイトになっています。
11. 各 PCB にはフルワードを定義しなければなりません。アセンブラー言語プログラムは、DB PCB 基底アドレスを使用することにより、DL/I 呼び出しが出された後の状況コードにアクセスできます。

    この例は、I/O PCB がアプリケーション・プログラムに渡されたものと想定しています。プログラムがバッチ・プログラムの場合には、I/O PCB が含まれるように、PSBGEN の PSBGEN ステートメントで CMPAT=YES を指定しなければなりません。バッチ・プログラムではシステム・サービス呼び出しを出すために I/O PCB が必要となるため、常に CMPAT=YES を指定するようにしてください。

制約事項: IMS 言語インターフェース・モジュール (DFSLI000) を、コンパイル済みのアセンブラー言語プログラムとバインドする必要があります。

アセンブラー言語での CICS オンライン・プログラム・コーディング

アセンブラー言語で書かれた次のサンプル・コードは、UIB のアドレス可能度の定義方法および設定方法を示しています。

プログラムの右側の数字は、このプログラムのあとの注の番号と対応しています。このプログラムは、DBCTL を使用して CICS 環境で実行できます。

呼び出しレベルのアセンブラー言語プログラム (CICS オンライン) のサンプル

```
PGMSTART DSECT                                          注
UIBPTR   DS    F
IOAREA   DS    0CL40                                    1
AREA1    DS    CL3
AREA2    DS    CL37
         DLIUIB
         USING UIB,8                                    2
PCBPTRS  DSECT
*        PSB ADDRESS LIST
PCB1PTR  DS    F
```

```
PCB1     DSECT
         USING PCB1,6                                              3
DBPC1DBD DS    CL8
DBPC1LEV DS    CL2
DBPC1STC DS    CL2
DBPC1PRO DS    CL4
DBPC1RSV DS    F
DBPC1SFD DS    CL8
DBPC1MKL DS    F
DBPC1NSS DS    F
DBPC1KFD DS    0CL256
DBPC1NM  DS    0CL12
DBPC1NMA DS    0CL14
DBPC1NMP DS    CL17
ASMUIB   CSECT
         B     SKIP
PSBNAME  DC    CL8'ASMPSB'
PCBFUN   DC    CL4'PCB'
REPLFUN  DC    CL4'REPL'
TERMFUN  DC    CL4'TERM'
GHUFUN   DC    CL4'GHU'
SSA1     DC    CL9'AAAA4444'
GOODRC   DC    XL1'00'
GOODSC   DC    CL2'  '
SKIP     DS    0H                                                  4
*              SCHEDULE PSB AND OBTAIN PCB ADDRESSES
         CALLDLI ASMTDLI,(PCBFUN,PSBNAME,UIBPTR)
         L     8,UIBPTR                                            5
         CLC   UIBFCTR,X'00'
         BNE   ERROR1
*              GET PSB ADDRESS LIST
         L     4,UIBPCBAL
         USING PCBPTRS,4
*              GET ADDRESS OF FIRST PCB IN LIST
         L     6,PCB1PTR
*              ISSUE DL/I CALL: GET A UNIQUE SEGMENT
         CALLDLI ASMTDLI,(GHUFUN,PCB1,IOAREA,SSA1)                 6
         CLC   UIBFCTR,GOODRC
         BNE   ERROR2
         CLC   DBPC1STC,GOODSC
         BNE   ERROR3                                              7
*              PERFORM SEGMENT UPDATE ACTIVITY
         MVC   AREA1,.......
         MVC   AREA2,.......
*              ISSUE DL/I CALL: REPLACE SEGMENT AT CURRENT POSITION
         CALLDLI ASMTDLI,(REPLFUN,PCB1,IOAREA,SSA1)                8
         CLC   UIBFCTR,GOODRC
         BNE   ERROR4
         CLC   DBPC1STC,GOODSC
         B     TERM
ERROR1   DS    0H
*              INSERT ERROR DIAGNOSTIC CODE
         B     TERM
ERROR2   DS    0H
*              INSERT ERROR DIAGNOSTIC CODE
         B     TERM
ERROR3   DS    0H
*              INSERT ERROR DIAGNOSTIC CODE
         B     TERM
ERROR4   DS    0H
*              INSERT ERROR DIAGNOSTIC CODE
ERROR5   DS    0H
*              INSERT ERROR DIAGNOSTIC CODE
         B     TERM
TERM     DS    0H
*              RELEASE THE PSB
```

```
        CALLDLI ASMDLI, (TERMFUN)
        EXEC CICS RETURN
        END   ASMUIB                                          9,10
```

注:

1. プログラムでは、リトリーブまたは挿入される最長セグメント (あるいは、プログラムが D コマンド・コードを使用する場合には、最長のセグメント経路) を収めるのに十分な大きさの入出力域を定義する必要があります。
2. DLIUIB ステートメントは、UIB DSECT をコピーします。
3. 各 DB PCB にはフルワードを定義しなければなりません。アセンブラー言語プログラムは、DB PCB 基底アドレスを使用することにより、DL/I 呼び出しが出された後の状況コードにアクセスできます。
4. これは非修飾 SSA です。修飾された SSA の場合には、SSA の各部分を個別に定義し、プログラムが SSA のフィールドを修正できるようにしてください。
5. この呼び出しは、PSB をスケジュールし、PSB アドレスを獲得します。
6. この呼び出しは、データベースからセグメントをリトリーブします。

   CICS オンライン・アセンブラー言語プログラムでは、呼び出しステートメントの代わりに CALLDLI マクロを使用して DL/I データベースにアクセスします。このマクロは、呼び出しステートメントに類似しています。このマクロの形式は、次のとおりです。

   ```
   CALLDLI ASMTDLI,(function,PCB-name,ioarea, SSA1,...SSAn),VL
   ```
7. CICS オンライン・プログラムでは、DB PCB 内の状況コードをチェックする前に、UIB に戻された戻りコードをチェックする必要があります。
8. REPL 呼び出しは、最後の Get Hold 呼び出しでリトリーブされたセグメント内のデータを置き換えます。このセグメント内のデータは、呼び出しで参照した入出力域の内容によって置き換えられます。
9. この呼び出しによって PSB が解放されます。
10. RETURN ステートメントは IMS レジスターをロードし、IMS に制御を戻します。

関連資料: CICS アプリケーション・プログラムのインストールについて詳しくは、「*CICS Transaction Server for z/OS CICS* アプリケーション・プログラミング・リファレンス」を参照してください。

関連資料:

270 ページの『UIB の指定 (CICS オンライン・プログラムのみ)』

C 言語でのバッチ・プログラム・コーディング

次のサンプル・コードは、IMS データベースにアクセスする IMS プログラムを C 言語で作成する方法を示しています。

プログラムの右側の数字は、このプログラムのあとの注の番号と対応しています。

C 言語プログラムのサンプル

```
#pragma runopts(env(IMS),plist(IMS))                                    NOTES
#include <ims.h>
#include <stdio.h>                                                       1
main() {                                                                 2
/*                                                          */
/*             descriptive statements                       */
/*                                                          */
 IO_PCB_TYPE *IO_PCB = (IO_PCB_TYPE*)PCBLIST[0];
 struct {PCB_STRUCT(10)} *mast_PCB   = __pcblist[1];
struct {PCB_STRUCT(20)} *detail_PCB = __pcblist[2];                      3
const static char func_GU[4]   = "GU  ";
const static char func_GN[4]   = "GN  ";
const static char func_GHU[4]  = "GHU ";
const static char func_GHN[4]    = "GHN ";
const static char func_GNP[4]    = "GNP ";                               4
const static char func_GHNP[4] = "GHNP";
const static char func_ISRT[4] = "ISRT";
const static char func_REPL[4] = "REPL";
const static char func_DLET[4] = "DLET";
char qual_ssa[8+1+8+2+6+1+1]; /* initialized by sprintf                 5
                              /*below.  See the  */
                              /*explanation for  */
                              /*sprintf in note 7 for the */
                              /*meanings of 8,1,8,2,6,1 ——*/
                              /*the final 1 is for the    */
                              /*trailing '¥0' of string */
static const char unqual_ssa[]= "NAME     ");
                              /* 12345678_ */
struct {
          ——
          ——
          ——
       } mast_seg_io_area;

struct {
          ——
          ——                                                             6
          ——
       } det_seg_io_area;
/*                                                          */
/*        Initialize the qualifier                          */
/*                                                          */

    sprintf(qual_ssa,
           "8.8s(8.8s6.6s)",
           "ROOT", "KEY", "=", "vvvvv");                                 7
/*                                                          */
/*           Main part of C batch program                   */
/*                                                          */
     ctdli(func_GU, detail_PCB,
          &det_seg_io_area,qual_ssa);                                    8

     ctdli(func_GHU, mast_PCB,
          &mast_seg_io_area,qual_ssa);                                   9

     ctdli(func_GHN, mast_PCB,
          &mast_seg_io_area);                                            10

     ctdli(func_REPL, mast_PCB,
          &mast_seg_io_area;                                             11
}                                                                        12
```

注:

1. IMS のもとで呼び出されると、**env(IMS)** は適切な操作環境を設定し、**plist(IMS)** は適切なパラメーター・リストを設定します。**ims.h** ヘッダー・ファイルには、PCB レイアウト、**__pcblist**、および **ctdli** ルーチンの宣言が入っています。PCB レイアウトは、プログラムが使用する PCB のマスクを構造として定義します。プログラムはこれらの定義を使用して、PCB 内のフィールドをチェックできます。

   stdio.h ヘッダー・ファイルには、(SSA の構築に使用される) **sprintf** の宣言が含まれます。
2. IMS は、アプリケーション・プログラムの PSB をロードした後、このエントリー・ポイントを介してアプリケーション・プログラムに制御を渡します。
3. C の実行時に **__pcblist** の値がセットされます。 PCB を参照する順序は、PSB で定義されたときの順序どおりでなければなりません。 (実際に必要なキー長に応じて、「10」と「20」を除いた他の値を使用できます。) これらの宣言は、次のようなマクロを使用して行われます。

   ```
   #define IO_PCB (IO_PCB_TYPE *) (__pcblist[0])
   #define mast_PCB   (__pcblist[1])
   #define detail_PCB (__pcblist[2])
   ```

   この例は、I/O PCB がアプリケーション・プログラムに渡されたものと想定しています。プログラムがバッチ・プログラムの場合には、I/O PCB が含まれるように、PSBGEN の PSBGEN ステートメントで CMPAT=YES を指定しなければなりません。バッチ・プログラムではシステム・サービス呼び出しを出すために I/O PCB が必要となるため、常に CMPAT=YES をバッチ・プログラムについて指定する必要があります。
4. これらの各区域は、バッチ・プログラムで使用された呼び出し機能のうちいずれかを定義します。各文字ストリングは、各機能に割り立てられた値をもつ 4 文字の英数字として定義されます。 (**[4]** を指定しなかった場合には、各定数のために 5 バイトが予約されます。) 他の定数も同様に定義できます。また、標準定義をソース・ライブラリーに保管させ、**#include** ディレクティブを使用してそれらの定義を組み込むこともできます。

   あるいは、マクロを使ってこれらを定義することもできますが、各ストリングのトレーラーにヌル (**'¥0'**) が付きます。
5. SSA は、ストリングに書き込まれます (注 7 を参照)。 COBOL、PL/I、または Pascal の場合と同じように構造を定義することもできますが、**sprintf** を使用したほうが便利です。 (C ストリングの末尾にはヌルがあり、これは IMS に渡せないことを忘れないでください。)ストリングは末尾にヌルを含み、それが IMS によって無視されるため、ストリング長は IMS の要求より 1 バイトだけ大きくなります。また、大括弧内の数字は、これらの長さに等しいフィールドが SSA 内に 6 つあるものと想定したものです。
6. データベースとの間でセグメントの受け渡しを行うために使用される入出力域が、構造として定義されます。
7. **sprintf** 関数は、SSA に記入を行うために使用されます。「**%-8.8s**」という書式は、「ちょうど 8 桁の、左寄せされたストリング」を意味します。
   「**%2.2s**」という書式は、「ちょうど 2 桁の、右寄せされたストリング」を意味します。

ROOT の部分と **KEY** の部分は変化しないため、これは次のようにコーディングすることもできます。

```
sprintf(qual_ssa,
        "ROOT    (KEY      =%-6.6s)", "vvvvv");
      /* 12345678 12345678 */
```

8. この呼び出しは、データベースからデータをリトリーブします。これには修飾された SSA が含まれています。修飾された SSA を使用する呼び出しを出す前に、SSA のデータ・フィールドを初期設定してください。非修飾 SSA を使用する呼び出しを出す前に、セグメント名フィールドを初期設定してください。COBOL、PL/I、および Pascal のインターフェース・ルーチンとは異なり、**ctdli** もその結果として状況コードを戻します。(ブランクは 0 に変換されます。) したがって、次のようにコーディングできます。

```
switch (ctdli(....)) {
    case 0: ... /* everything ok */

            break;
    case 'AB': ....

            break;
    case 'IX': ...

            break;
    default:
}
```

C プログラムの DL/I 呼び出しでは、PCB ポインターだけを渡すことができます。

9. これも、修飾された SSA を使用する呼び出しです。
10. この呼び出しは、データベースからデータをリトリーブする非修飾呼び出しです。これは Get Hold 呼び出しであるため、この後に `REPL` または `DLET` を出すことができます。
11. `REPL` 呼び出しは、最後の Get Hold 呼び出しでリトリーブされたセグメント内のデータを置き換えます。このセグメント内のデータは、呼び出しで参照した入出力域の内容によって置き換えられます。
12. (**return** ステートメントまたは **exit** 呼び出しによって) **main** ルーチンが終わると、IMS に制御が戻されます。

制約事項**:** IMS は、IMS と C 言語の間のインターフェースである、言語インターフェース・モジュール (DFSLI000) を提供しています。このモジュールは、バインド時にアプリケーション・プログラムによって使用できるようになっていなければなりません。

COBOL でのバッチ・プログラム・コーディング

次のサンプル・コードは、IMS データベースにアクセスする IMS プログラムを COBOL で作成する方法を示しています。

プログラムの右側の数字は、このプログラムのあとの注の番号と対応しています。この種のプログラムは、バッチ・プログラム、またはバッチ指向 BMP として実行できます。

COBOL プログラムのサンプル

```
Identification Division.
Program-ID. BATCOBOL.
Environment Division.
Data Division.
Working-Storage Section.
  01 Func-Codes.
     05 Func-GU          Picture XXXX Value 'GU  '.
     05 Func-GHU         Picture XXXX Value 'GHU '.
     05 Func-GN          Picture XXXX Value 'GHN '.
     05 Func-GHN         Picture XXXX Value 'GHN '.
     05 Func-GNP         Picture XXXX Value 'GNP '.
     05 Func-GHNP        Picture XXXX Value 'GHNP'.
     05 Func-REPL        Picture XXXX Value 'REPL'.
     05 Func-ISRT        Picture XXXX Value 'ISRT'.
     05 Func-DLET        Picture XXXX Value 'DLET'.
     05 Parmcount        Picture S9(5) Value +4 Comp-5.
  01 Unqual-SSA.
     05 Seg-Name         Picture X(08) Value  '        '.
     05 Filler           Picture X     Value  ' '.
  01 Qual-SSA-Mast.
     05 Seg-Name-M       Picture X(08) Value 'ROOTMast'.
     05 Begin-Paren-M    Picture X     Value '('.
     05 Key-Name-M       Picture X(08) Value 'KeyMast '.
     05 Kel-Oper-M       Picture X(05) Value ' ='.
     05 Key-Value-M      Picture X(06) Value 'VVVVVV'.
     05 End-Paren-M      Picture X     Value ')'.
  01 Qual-SSA-Det.
     05 Seg-Name-D       Picture X(08) Value 'ROOTDET '.
     05 Begin-Paren-D    Picture X     Value '('.
     05 Key-Name-D       Picture X(08) Value 'KEYDET  '.
     05 Rel-Oper-D       Picture X(05) Value ' ='.
     05 Key-Value-D      Picture X(06) Value 'VVVVVV'.
     05 End-Paren-D      Picture X     Value ')'.
  01 Det-Seg-In.
     05 Data1            Picture X.
     05 Data2            Picture X.
 01 Mast-Seg-In.
     05 Data1            Picture X.
     05 Data2            Picture X.
linkage section.
  01 IO-PCB.
     05 Filler           Picture X(10).
     05 IO-Status-Code   Picture XX.
     05 Filler           Picture X(20).
  01 DB-PCB-Mast.
     05 Mast-Dbd-Name    Picture X(8).
     05 Mast-Seg-Level   Picture XX.
     05 Mast-Status-Code Picture XX.
     05 Mast-Proc-Opt    Picture XXXX.
     05 Filler           Picture S9(5) Comp-5.
     05 Mast-Seg-Name    Picture X(8).
     05 Mast-Len-KFB     Picture S9(5) Comp-5.
     05 Mast-Nu-Senseg   Picture S9(5) Comp-5.
     05 Mast-Key-FB      Picture X(256).
  01 DB-PCB-Detail.
     05 Det-Dbd-Name     Picture X(8).
     05 Det-Seg-Level    Picture XX.
     05 Det-Status-Code  Picture XX.
     05 Det-Proc-Opt     Picture XXXX.
     05 Filler           Picture S9(5) Comp-5.
     05 Det-Seg-Name     Picture X(8).
     05 Det-Len-KFB      Picture S9(5) Comp-5.
     05 Det-Nu-Senseg    Picture S9(5) Comp-5.
     05 Det-Key-FB       Picture X(256).
```

```
Procedure Division using IO-PCB DB-PCB-Mast DB-PCB-Detail.
    Call 'CBLTDLI' using Func-GU DB-PCB-Detail
          Det-seg-in Qual-SSA-Det.
     .
     .
    Call 'CBLTDLI' using Parmcount Func-ghu DB-PCB-Mast
          Mast-seg-in Qual-SSA-Mast.
     .
     .
    Call 'CBLTDLI' using Func-GHN DB-PCB-Mast
          Mast-seg-in.
     .
     .
    Call 'CBLTDLI' using Func-REPL DB-PCB-Mast
          Mast-seg-in.
     .
     .
    Goback.
```

注:

1. プログラムが定義する各 DL/I 呼び出し機能を、77 レベルまたは 01 レベルの作業用ストレージ項目で定義します。各ピクチャー文節は、 4 文字の英数字として定義され、各機能に割り当てられた値が入ります。オプションの *parmcount* フィールドを含めたい場合には、各タイプの呼び出しごとにカウント値を初期設定できます。また、COBOL COPY ステートメントを使用してこれらの標準的記述をプログラムに含めることもできます。
2. 非修飾 SSA のために 9 バイト域が設定されます。非修飾 SSA が必要な呼び出しを出す前に、プログラムはセグメント名をこの区域に移します。複数の SSA が必要な呼び出しの場合には、追加の区域を定義する必要があります。
3. 01 レベルの作業用ストレージ項目で、アプリケーション・プログラムが使用するそれぞれの修飾された SSA を定義します。フィールド値が異なるため、修飾された SSA は別々に定義しなければなりません。
4. 01 レベルの作業用ストレージ項目で、データベースとの間でセグメントを受け渡すために使用する入出力域を定義します。02 レベルの項目でさらに入出力域を定義できます。各セグメント・タイプごとに別個の入出力域を使用することも、1 つの入出力域を定義してそれをすべてのセグメントに使用することもできます。01 レベル下のサブ項目でさらに入出力域を定義できます。各セグメント・タイプごとに別個の入出力域を使用することも、1 つの入出力域を定義してそれをすべてのセグメントに使用することもできます。
5. 01 レベルのリンケージ・セクション項目で、プログラムが必要とする各 PCB のマスクを定義します。 DB PCB は入力用データベースと出力用データベースの両方を表します。各 DL/I 呼び出しを出した後で、プログラムはこのリンケージを介して状況コードをチェックします。 DB PCB 内の各フィールドを定義して、プログラムでそれらのフィールドを参照できるようにします。
6. これが、バッチ・プログラムにおける標準の手続き部ステートメントです。 IMS は、アプリケーション・プログラムの PSB をロードした後、アプリケーション・プログラムに制御を渡します。PSB には、その PSB で定義されているすべての PCB が含まれています。手続き部ステートメントに USING がコーディングされ、その PCB の 1 つ 1 つを、プログラムがリンケージ・セクションで PCB マスクを定義するのに使った名前で参照しています。 PCB は、PSB で定義された順にリストする必要があります。

前述のサンプル・コードは、I/O PCB がアプリケーション・プログラムに渡されたものと想定しています。プログラムがバッチ・プログラムの場合には、I/O PCB が含まれるように、PSBGEN の PSBGEN ステートメントで CMPAT=YES を指定しなければなりません。バッチ・プログラムではシステム・サービス呼び出しを出すために I/O PCB が必要となるため、常に CMPAT=YES をバッチ・プログラムについて指定する必要があります。

入り口 DLITCBL ステートメントは主プログラムだけで使用されます。呼び出されるプログラムでは使用しないでください。

7. この呼び出しは、修飾された SSA を使用してデータベースからデータをリトリーブします。この呼び出しを出す前に、プログラムは、リトリーブする特定のセグメントを指定するために SSA のキーまたはデータ値を初期設定しなければなりません。プログラムは、この呼び出しを出した直後に、呼び出しで参照された DB PCB 内の状況コードを検査する必要があります。サンプル状況コード・エラー・ルーチンに対する呼び出しを除き、 COBOL プログラムの DL/I 呼び出しには *parmcount* パラメーターを含めることができます。ただし、これは COBOL で必須ではありません。
8. これも、修飾された SSA を含むリトリーブ呼び出しです。
9. これは、非修飾リトリーブ呼び出しです。
10. この REPL 呼び出しは、最後の Get Hold 呼び出しでリトリーブされたセグメントを置き換えます。このセグメントは、呼び出しで参照された入出力域 (MAST-SEG-IN) の内容によって置き換えられます。
11. 処理が終了すると、プログラムは GOBACK ステートメントを出します。

関連資料: これらのプロシージャーの使用方法については、「*IMS V15* システム定義」を参照してください。

IMS 言語インターフェース・モジュールへの COBOL コードのバインド

IMS は言語インターフェース・モジュール (DFSLI000) を提供します。プログラムをコンパイルした後で、このモジュールをバッチ・プログラムとバインドしなければなりません。このモジュールにより、IMS に対する共通インターフェースが提供されます。

IMS 提供のプロシージャー (IMSCOBOL または IMSCOBGO) を使用すると、IMS は言語インターフェースをアプリケーション・プログラムとバインドします。IMSCOBOL は、ユーザー・プログラムのコンパイルとバインドを行う、2 つのステップからなるプロシージャーです。 IMSCOBGO は、ユーザー・コンパイルをコンパイルし、バインドし、IMS バッチ領域で実行する、3 ステップのプロシージャーです。

COBOL での CICS オンライン・プログラム・コーディング

次のサンプル・コードは、Enterprise COBOL で書かれたスケルトン・オンライン・プログラムです。これらのプログラムは、UIB の定義方法および UIB に対するアドレス可能度を設定する方法を例示しています。

プログラムの右側の数字は、このプログラムの後の注の番号と対応しています。この種のプログラムは、DBCTL を使用して CICS 環境で実行できます。

CICS で実行可能な COBOL プログラムのサンプル

```
Identification Division.
       Program-ID. CBLUIB.
       Environment Division.
       Data Division.
       Working-Storage Section.
         01 Func-Codes.
            05  Psb-Name          Picture X(8) Value 'CBLPSB  '.
            05  Func-PCB          Picture X(4) Value 'PCB '.
            05  Func-TERM         Picture X(4) Value 'TERM'.                  1
            05  Func-GHU          Picture X(4) Value 'GHU '.
            05  Func-REPL         Picture X(4) Value 'REPL'.
            05  SSA1              Picture X(9) Value 'AAAA4444 '.
            05  Success-Message   Picture X(40).
            05  Good-Status-Code  Picture XX Value '  '.                     2
            05  good-return-code  Picture X Value low-Value.
       01  Message0.
           05  Message1           Picture X(38).                             3
           05  Message2           Picture XX.
       01  Dli-IO-Area.
           05  Area1              Picture X(3).
           05  Area2              Picture X(37).
             Procedure Division.
      * Schedule the psb and address the uib
           Call 'CBLTDli' using Func-PCB Psb-Name                            4
                address of Dliuib.
           If Uibfctr is not equal low-Values then
      * Insert error diagnostic code
              Exec CICS return end-exec
           End-if.
           Set address of pcb-addresses to pcbaddr.
      * Issue DL/I Call: get a unique segment
           Set address of pcb1 to pcb-address-list(1).
           Call 'CBLTDli' using Func-GHU Pcb1                                5
                Dli-io-area ssa1.
           If uibfctr is not equal good-return-code then
      * Insert error diagnostic code                                          6
              Exec CICS return end-Exec
           End-if.
           If pcb1-status-code is not equal good-status-code then
      * Insert error diagnostic code
              Exec CICS return end-Exec
           End-if.
      *    Perform segment update activity
           Move 'aaa' to area1.
           Move 'bbb' to area2.
      *    Issue DL/I Call: replace segment at current position             7
           Call 'CBLTDli' using Func-REPL Pcb1
                Dli-io-area ssa1
           If uibfctr is not equal good-return-code then
      * Insert error diagnostic code
              Exec CICS return end-Exec
           End-if.
           If pcb1-status-code is not equal good-status-code then
      * Insert error diagnostic code
              Exec CICS return end-Exec
           End-if.
      * Release the psb
           Call 'CBLTDli' using Func-TERM.
      * Other application Function                                           8,9
           Exec CICS return end-Exec.
           Goback.
```

注:

1. プログラムが定義する各 DL/I 呼び出し機能を、77 レベルまたは 01 レベルの作業用ストレージ項目で定義します。各ピクチャー文節は、 4 文字の英数字として定義され、各機能に割り当てられた値が入ります。オプションの *parmcount* フィールドを含めたい場合には、各タイプの呼び出しごとにカウント値を初期設定できます。また、COBOL COPY ステートメントを使用してこれらの標準的記述をプログラムに含めることもできます。
2. 非修飾 SSA のために 9 バイト域が設定されます。非修飾 SSA を必要とする呼び出しを出す前に、プログラムは、セグメント名を使用してこの区域を初期設定するか、あるいはそのセグメント名をこの区域に移すことができます。複数の SSA が必要な呼び出しの場合には、追加の区域を定義する必要があります。
3. 01 レベルの作業用ストレージ項目で、データベースとの間でセグメントを受け渡すために使用する入出力域を定義します。01 レベル下のサブ項目でさらに入出力域を定義できます。各セグメント・タイプごとに別個の入出力域を使用することも、1 つの入出力域を定義してそれをすべてのセグメントに使用することもできます。
4. リンケージ・セクションで PCB レイアウトが 1 つ定義されます。PCB-ADDRESS-LIST の定義が *n* 回繰り返されます。ただし、*n* は PSB 内の PCB の数以上の値です。
5. `PCB` 呼び出しは、ユーザーのプログラムで使用するために PSB をスケジュールします。DLIUIB パラメーターのアドレスによって DLIUIB のアドレスが戻されます。
6. この非修飾 `GHU` 呼び出しは、データベースからセグメントをリトリーブし、呼び出しで参照されている入出力域に入れます。この呼び出しを出す前に、プログラムは、リトリーブする特定のセグメントを指定するために SSA のキーまたはデータ値を初期設定しなければなりません。
7. CICS オンライン・プログラムでは、DB PCB 内の状況コードを検査する前に、UIB に戻された戻りコードを検査する必要があります。
8. `REPL` 呼び出しは、最後の Get Hold 呼び出しでリトリーブされたセグメントを、プログラムが入出力域に入れたデータによって置き換えます。
9. `TERM` 呼び出しは、プログラムが以前にスケジュールした PSB を終了させます。この呼び出しはオプションであり、処理を続行する前に同期点をとる必要がある場合にのみ出されます。処理が終了すると、プログラムは EXEC CICS RETURN ステートメントを出します。これが最高位レベルの CICS プログラムからの戻りである場合には、 CICS によって `TERM` 呼び出しと同期点が内部的に生成されます。

CICS オンラインでの呼び出しレベル OS/VS COBOL プログラムのサンプル (Enterprise COBOL では廃止)

```
       Identification Division.                                    注
    Program-ID. CBLUIB.
       Environment Division.
       Data Division.
       Working-Storage Section.
         01 Func-Codes.
            05  Psb-Name          Picture X(8) Value 'CBLPSB  '.    1
            05  Func-PCB          Picture X(4) Value 'PCB '.
            05  Func-TERM         Picture X(4) Value 'TERM'.
            05  Func-GHU          Picture X(4) Value 'GHU '.
            05  Func-REPL         Picture X(4) Value 'REPL'.
            05  SSA1              Picture X(9) Value 'AAAA4444 '.   2
```

```
      05  Success-Message   Picture X(40).
      05  Good-Status-Code  Picture XX   Value '  '.
      05  Good-Return-Code  Picture X    Value low-Value.
 01  Message0.
     05  Message1           Picture X(38).
     05  Message2           Picture XX.
 01  Dli-IO-Area.                                                    3
     05  Area1              Picture X(3).
     05  Area2              Picture X(37).
 Linkage Section.                                                    4
 01  BllCells.
     05 FIller              Picture S9(8) Comp-5.
     05 Uib-Ptr             Picture S9(8) Comp-5.
     05 B-Pcb-Ptrs          Picture S9(8) Comp-5.
     05 Pcb1-Ptr            Picture S9(8) Comp-5.
     Copy DliUib.                                                  5,6
 01  Overlay-Dliuib Redefines Dliuib.
     05  Pcbaddr usage is pointer.
     05  Filler             Picture XX.
 01  Pcb-Ptrs.
     05 B-Pcb1-Ptr          Picture 9(8)  Comp-5.
 01  Pcb1.                                                           7
     05 Pcb1-Dbd-Name       Picture X(8).
     05 Pcb1-Seg-Level      Picture XX.
     05 Pcb1-Status-Code    Picture XX.
     05 Pcb1-PROC-OPT       Picture XXXX.
     05 FIller              Picture S9(5) Comp-5.
     05 Pcb1-Seg-Name       Picture X(8).
     05 Pcb1-Len-KFB        Picture S9(5) Comp-5.
     05 Pcb1-NU-ENSeg       Picture S9(5) Comp-5.
     05 Pcb1-KEY-FB         Picture X(256).
 Procedure Division.                                                 8
     Call 'CBLTDLI' using Func-PCB Psb-Name Uib-ptr.
     If Uibfctr is not equal low-values then
*       Insert error diagnostic Code
        Exec CICS Return end-Exec
     End-if.
     Move Uibpcbal to B-Pcb-Ptrs.
     Move B-Pcb1-Ptr to Pcb1-Ptr.

*       Issue DL/I Call: get a unique segment                       9
     Call 'CBLTDLI' using Func-GHU Pcb1
         Dli-io-area ssa1.
     Service reload Uib-ptr
     If Uibfctr is not equal Good-Return-Code then                 10
*       Insert error diagnostic Code
        Exec CICS Return end-Exec
     End-if.

     If Pcb1-Status-Code is not equal Good-Status-Code then
*       Insert error diagnostic Code
        Exec CICS Return end-Exec
     End-if.


*       Perform segment update activity
     Move 'aaa' to area1.
     Move 'bbb' to area2.
*       Issue DL/I Call: replace segment at current position       11
     Call 'CBLTDLI' using Func-REPL Pcb1
          Dli-io-area ssa1.
     If Uibfctr is not equal Good-Return-Code then
*       Insert error diagnostic Code
        Exec CICS Return end-Exec
     End-if.

     If Pcb1-Status-Code is not equal Good-Status-Code then
```

```
*       Insert error diagnostic Code
        Exec CICS Return end-Exec
     End-if.

*    Release the PSB
     Call 'CBLTDLI' using Func-TERM.                          12,13
       Exec CICS Return end-Exec.
```

注:

1. プログラムが定義する各 DL/I 呼び出し機能を、77 レベルまたは 01 レベルの作業用ストレージ項目で定義します。各ピクチャー文節は、 4 文字の英数字として定義され、各機能に割り当てられた値が入ります。オプションの *parmcount* フィールドを含めたい場合には、各タイプの呼び出しごとにカウント値を初期設定できます。また、COBOL COPY ステートメントを使用してこれらの標準的記述をプログラムに含めることもできます。
2. 非修飾 SSA のために 9 バイト域が設定されます。非修飾 SSA を必要とする呼び出しを出す前に、プログラムは、セグメント名を使用してこの区域を初期設定するか、あるいはそのセグメント名をこの区域に移すことができます。複数の SSA が必要な呼び出しの場合には、追加の区域を定義する必要があります。
3. 01 レベルの作業用ストレージ項目で、データベースとの間でセグメントを受け渡すために使用する入出力域を定義します。02 レベルの項目でさらに入出力域を定義できます。各セグメント・タイプごとに別個の入出力域を使用することも、1 つの入出力域を定義してそれをすべてのセグメントに使用することもできます。
4. リンケージ・セクションは、アプリケーション・プログラムの作業用ストレージの外部にあるストレージのアドレスが入るパラメーター・リストへのアドレッシングを可能にするために、このタイプの定義で始めなければなりません。最初の 02 レベル定義は、リスト内の他のフィールドへのアドレッシングを可能にするために、CICS によって使用されます。リンケージ・セクションでは、他の 02 レベルの名前と 01 レベルのデータ定義との間に 1 対 1 の対応が成り立ちます。
5. COPY DLIUIB ステートメントが展開されます。
6. UIB は、PCB アドレスが入っている区域のアドレスを戻します。実際の PCB アドレスを得るためには、PCB ポインターの定義が必要です。この区域内のアドレスは変更しないでください。
7. PCB はリンケージ・セクションで定義されます。
8. PCB 呼び出しは、ユーザーのプログラムで使用するために PSB をスケジュールします。
9. この非修飾 GHU 呼び出しは、データベースからセグメントをリトリーブし、呼び出しで参照されている入出力域に入れます。この呼び出しを出す前に、プログラムは、リトリーブする特定のセグメントを指定するために SSA のキーまたはデータ値を初期設定しなければなりません。
10. CICS オンライン・プログラムでは、DB PCB 内の状況コードを検査する前に、UIB に戻された戻りコードを検査する必要があります。
11. REPL 呼び出しは、最後の Get Hold 呼び出しでリトリーブされたセグメントを、プログラムが入出力域に入れたデータによって置き換えます。

12. TERM 呼び出しは、プログラムが以前にスケジュールした PSB を終了させます。この呼び出しはオプションであり、処理を続行する前に同期点をとる必要がある場合にのみ出されます。
13. 処理が終了すると、プログラムは EXEC CICS RETURN ステートメントを出します。これが最高位レベルの CICS プログラムからの戻りである場合には、CICS によって TERM 呼び出しと同期点が内部的に生成されます。

関連資料: アプリケーション・プログラムのインストールについて詳しくは、「*CICS Transaction Server for z/OS CICS* アプリケーション・プログラミング・ガイド」を参照してください。

関連資料:

270 ページの『UIB の指定 (CICS オンライン・プログラムのみ)』

Java でのプログラム・コーディング

IMS は Java プログラミング言語を使用したアプリケーション開発にサポートを提供します。

Java 開発用の IMS ソリューションのドライバーおよびリソース・アダプターを使用して、IMS データベースへのアクセスや IMS トランザクションの処理を実行する Java アプリケーションを作成できます。

関連概念:

693 ページの『第 38 章 Java 開発用の IMS ソリューションの概要』

Pascal でのバッチ・プログラム・コーディング

以下のコード・サンプルは、Pascal で書かれたスケルトン・バッチ・プログラムです。このプログラムは、Pascal で書かれた IMS プログラムの各部分が整合して機能する様子を示しています。プログラムの右側の数字は、このプログラムの後の注の番号と対応しています。

制約事項: Pascal は、CICS ではサポートされません。

```
segment PASCIMS;                                                NOTES
                                                                    1
  type                                                              2
    CHAR2 = packed array [1..2] of CHAR;
    CHAR4 = packed array [1..4] of CHAR;
    CHAR6 = packed array [1..6] of CHAR;
    CHARn = packed array [1..n] of CHAR;
                DB_PCB_TYPE = record                                3
                DB_NAME : ALFA;
                DB_SEG_LEVEL : CHAR2;
                DB_STAT_CODE : CHAR2;
                DB_PROC_OPT  : CHAR4;
                FILLER       : INTEGER;
                DB_SEG_NAME  : ALFA;
                DB_LEN_KFB   : INTEGER;
                DB_NO_SENSEG : INTEGER;
                DB_KEY_FB    : CHARn;
              end;
  procedure PASCIMS (var SAVE: INTEGER;                             4
                     var DB_PCB_MAST: DB_PCB_TYPE;
                      var DB_PCB_DETAIL : DB_PCB_TYPE);
                REENTRANT;
  procedure PASCIMS;
  type                                                              5
```

```
  QUAL_SSA_TYPE  = record
                             SEG_NAME           : ALFA;
                  SEQ_QUAL     : CHAR;
                  SEG_KEY_NAME : ALFA;
                  SEG_OPR      : CHAR2;
                  SEG_KEY_VALUE: CHAR6;
                  SEG_END_CHAR : CHAR;
                 end;
  MAST_SEG_IO_AREA_TYPE = record
                            (* Field declarations *)
                           end;
   DET_SEG_IO_AREA_TYPE = record
                            (* Field declarations *)
                          end;
var                                                              6
  MAST_SEG_IO_AREA : MAST_SEG_IO_AREA_TYPE;
  DET_SEG_IO_AREA : DET_SEG_IO_AREA_TYPE;
const                                                            7
   GU   = 'GU  ';
   GN   = 'GN  ';
  GHU  = 'GHU ';
  GHN  = 'GHN ';
  GHNP = 'GHNP';
  ISRT = 'ISRT';
  REPL = 'REPL';
  DLET = 'DLET';
  QUAL_SSA = QUAL_SSA_TYPE('ROOT','(','KEY',' =',
                           'vvvvv',')');
  UNQUAL_SSA = 'NAME     ';
 procedure PASTDLI; GENERIC;                                      8
begin
  PASTDLI(const GU,                                              9
          var   DB_PCB_DETAIL;
          var   DET_SEG_IO_AREA;
          const QUAL_SSA);
  PASTDLI(const GHU,                                             10
          var   DB_PCB_MAST,
          var   MAST_SEG_IO_AREA,
          const QUAL_SSA);
  PASTDLI(const GHN,                                             11
          var   DB_PCB_MAST,
          var   MAST_SEG_IO_AREA);
  PASTDLI(const REPL,                                            12
          var   DB_PCB_MAST,
          var   MAST_SEG_IO_AREA);
end;
                                                                 13
```

注:

1. Pascal のコンパイル単位の名前を定義します。
2. ユーザーのプログラムで使用される PCB のために必要なデータ・タイプを定義します。
3. ユーザーのプログラムで使用される PCB データ・タイプを定義します。
4. IMS で呼び出される REENTRANT プロシージャーのためのプロシージャー見出しを宣言します。パラメーター・リストの最初のワードは INTEGER にしてください。 INTEGER は VS Pascal 用の予約語です。パラメーターの残りの部分は、IMS から受け取った PCB のアドレスです。
5. SSA および入出力域のために必要なデータ・タイプを定義します。
6. 入出力域に使用される変数を宣言します。

7. PASTDLI DL/I 呼び出しで使用される、機能コードや SSA などの定数を定義します。
8. GENERIC ディレクティブを使用して IMS インターフェース・ルーチンを宣言します。GENERIC は、複数のパラメーター・リスト・フォーマットが使用できる外部ルーチンを識別します。 GENERIC ルーチンのパラメーターは、ルーチンが呼び出されるときに初めて「宣言されます」。
9. この呼び出しは、データベースからデータをリトリーブします。これには修飾された SSA が含まれています。修飾された SSA を使用する呼び出しを出す前に、SSA のデータ・フィールドを初期設定してください。非修飾 SSA を使用する呼び出しを出す前に、セグメント名フィールドを初期設定してください。
10. これも、修飾された SSA で修飾された呼び出しです。
11. この呼び出しは、データベースからデータをリトリーブする非修飾呼び出しです。これは Get Hold 呼び出しであるため、その後に続けて REPL 呼び出しまたは DLET 呼び出しを出すことができます。
12. REPL 呼び出しは、最後の Get Hold 呼び出しでリトリーブされたセグメント内のデータを置き換えます。このデータは、呼び出しで参照した入出力域の内容によって置き換えられます。
13. PASCIMS プロシージャーから抜け出して IMS に制御を戻します。RETURN ステートメントをコーディングすることにより、別の点から抜け出すこともできます。

制約事項: プログラムをコンパイルした後で、このプログラムを IMS 言語インターフェース・モジュール DFSLI000 とバインドしなければなりません。

PL/I でのバッチ・プログラム・コーディング

次のサンプル・コードは、PL/I で書かれたスケルトン・バッチ・プログラムです。これは、PL/I で書かれた IMS プログラムの各部分が整合して機能する様子を示しています。

プログラムの右側の数字は、このプログラムの後の注の番号と対応しています。この種のプログラムは、バッチ・プログラム、またはバッチ指向 BMP として実行できます。

制約事項: IMS アプリケーション・プログラムでは、PL/I マルチタスキングを使用することはできません。これは、マルチタスキングを行う場合には、すべてのタスクが PL/I 制御タスクのサブタスクとして機能するためです。

PL/I プログラムのサンプル

```
/*                                                  */                  NOTES
/*               ENTRY POINT                        */
/*                                                  */
DLITPLI: PROCEDURE (IO_PTR_PCB,DB_PTR_MAST,DB_PTR_DETAIL)                  1
             OPTIONS (MAIN);
/*                                                  */
/*            DESCRIPTIVE STATEMENTS                */
/*                                                  */
DCL  IO_PTR_PCB POINTER;
DCL  DB_PTR_MAST POINTER;
DCL  DB_PTR_DETAIL POINTER;
DCL  FUNC_GU     CHAR(4)    INIT('GU  ');                                  2
```

```
DCL  FUNC_GN     CHAR(4)    INIT('GN  ');
DCL  FUNC_GHU    CHAR(4)    INIT('GHU ');
DCL  FUNC_GHN    CHAR(4)    INIT('GHN ');
DCL  FUNC_GNP    CHAR(4)    INIT('GNP ');
DCL  FUNC_GHNP   CHAR(4)    INIT('GHNP');
DCL  FUNC_ISRT   CHAR(4)    INIT('ISRT');
DCL  FUNC_REPL   CHAR(4)    INIT('REPL');
DCL  FUNC_DLET   CHAR(4)    INIT('DLET');
DCL 1    QUAL_SSA           STATIC UNALIGNED,                          3
         2   SEG_NAME       CHAR(8) INIT('ROOT    '),
         2   SEG_QUAL       CHAR(1) INIT('('),
         2   SEG_KEY_NAME   CHAR(8) INIT('KEY    '),
         2   SEG_OPR        CHAR(2) INIT(' ='),
         2   SEG_KEY_VALUE  CHAR(6) INIT('vvvvv'),
         2   SEG_END_CHAR   CHAR(1) INIT(')');
DCL 1    UNQUAL SSA         STATIC UNALIGNED,
         2   SEG_NAME_U     CHAR(8) INIT('NAME    '),
         2   BLANK          CHAR(1) INIT(' ');
DCL 1   MAST_SEG_IO_AREA,                                              4
         2 ——
         2 ——
         2 ——
DCL 1   DET_SEG_IO_AREA,
         2 ——
         2 ——
         2 ——
DCL 1    IO_PCB               BASED (IO_PTR_PCB),                      5
         2   FILLER           CHAR(10),
         2   STAT             CHAR(2);
DCL 1    DB_PCB_MAST          BASED (DB_PTR_MAST),
         2   MAST_DB_NAME     CHAR(8),
         2   MAST_SEG_LEVEL   CHAR(2),
         2   MAST_STAT_CODE   CHAR(2),
         2   MAST_PROC_OPT    CHAR(4),
         2   FILLER           FIXED BINARY (31,0),
         2   MAST_SEG_NAME    CHAR(8),
         2   MAST_LEN_KFB     FIXED BINARY (31,0),
         2   MAST_NO_SENSEG   FIXED BINARY (31,0),
         2   MAST_KEY_FB      CHAR(*);
DCL 1    DB_PCB_DETAIL        BASE (DB_PTR_DETAIL),
         2   DET_DB_NAME      CHAR(8),
         2   DET_SEG_LEVEL    CHAR(2),
         2   DET_STAT_CODE    CHAR(2),
         2   DET_PROC_OPT     CHAR(4),
         2   FILLER           FIXED BINARY (31,0),
         2   DET_SEG_NAME     CHAR(8),
         2   DET_LEN_KFB      FIXED BINARY (31,0),
         2   DET_NO_SENSEG    FIXED BINARY (31,0),
         2   DET_KEY_FB       CHAR(*);
DCL      THREE    FIXED BINARY     (31,0)   INITIAL(3);                6
DCL      FOUR     FIXED BINARY     (31,0)   INITIAL(4);
DCL      FIVE     FIXED BINARY     (31,0)   INITIAL(5);
DCL      SIX      FIXED BINARY     (31,0)   INITIAL(6);
/*                                                          */
/*            MAIN PART OF PL/I BATCH PROGRAM               */
/*                                                          */
CALL PLITDLI (FOUR,FUNC_GU,DB_PCB_DETAIL,DET_SEG_IO_AREA, QUAL_SSA);   7
   IF DET_STAT_CODE = GOOD_STATUS_CODE THEN DO;
      CALL PLITDLI (FOUR,FUNC_GHU,DB_PCB_MAST,MAST_SEG_IO_AREA,QUAL_SSA);  8
      IF MAST_STAT_CODE = GOOD_STATUS_CODE THEN DO;
         CALL PLITDLI (THREE,FUNC_GHN,DB_PCB_MAST,MAST_SEG_IO_AREA);   9
         IF MAST_STAT_CODE = GOOD_STATUS_CODE THEN DO;
            CALL PLITDLI (THREE,FUNC_REPL,DB_PCB_MAST,MAST_SEG_IO_AREA);  10
            IF MAST_STAT_CODE ^= GOOD_STATUS_CODE THEN DO;
               /* INSERT REPLACE DIAGNOSTIC MESSAGE */
          END;
   END;
```

```
         ELSE DO;
            /* INSERT GHN DIAGNOSTIC MESSAGE */
   END;
         END;
      ELSE DO;
         /* INSERT GHU DIAGNOSTIC MESSAGE */
        END;
END;
ELSE DO;
   /* INSERT GU DIAGNOSTIC MESSAGE */
END;
RETURN;                                                          11
END DLITPLI;
```

注:

1. IMS は、アプリケーション・プログラムの PSB をロードした後、このエントリー・ポイントを介してアプリケーション・プログラムに制御を渡します。 PL/I プログラムは入り口ステートメントで、PCB 名を渡すのではなく、 PCB を差すポインターを渡す必要があります。入り口ステートメントでは、プログラムが使用する PCB を、PCB マスクの定義に割り当てられた名前でリストします。入り口ステートメントで PCB を参照する順序は、PSB で定義されたときの順序どおりでなければなりません。

   このサンプル・コードは、I/O PCB がアプリケーション・プログラムに渡されたものと想定しています。プログラムがバッチ・プログラムの場合には、I/O PCB が含まれるように、PSBGEN の PSBGEN ステートメントで CMPAT=YES を指定しなければなりません。バッチ・プログラムではシステム・サービス呼び出しを出すために I/O PCB が必要となるため、常に CMPAT=YES をバッチ・プログラムについて指定する必要があります。
2. これらの各区域は、バッチ・プログラムで使用された呼び出し機能のうちいずれかを定義します。各文字ストリングは、各機能に割り立てられた値をもつ 4 文字の英数字として定義されます。同様に他の定数、定義できます。また、標準定義をソース・ライブラリーに保管し、%INCLUDE ステートメントを使用してそれらの定義を組み込むこともできます。
3. 構造定義は、プログラムが使用する各 SSA を定義します。 SSA には、非境界合わせ属性を使用する必要があります。 SSA 文字ストリングは、ストレージ内で連続していなければなりません。各 SSA のデータ・フィールドの値が異なるため、それぞれの修飾された SSA に関する構造は別々に定義するようにしてください。
4. データベースとの間でセグメントを受け渡すために使用される入出力域は、構造として定義されます。
5. レベル 01 の宣言は、プログラムが使用する PCB のマスクを構造として定義しています。プログラムはこれらの定義を使用して、PCB 内のフィールドをチェックできます。
6. このステートメントは、PL/I プログラムから出された DL/I 呼び出しで必要になる *parmcount* を定義しています (ただし、サンプル状況コード・エラー・ルーチンに対する呼び出しでは parmcount は使用できません)。*parmcount* は、呼び出し内でその後に続く、パラメーターの数が入る 4 バイト・フィールドのアドレスです。 *parmcount* が必要になるのは PL/I プログラムの場合だけです。他の言語ではオプショナルです。 *parmcount* 内の値は 2 進数 です。

この例は、呼び出し内で 3 つのパラメーターが後に続いている場合の *parmcount* パラメーターのコーディング方法を示しています。

```
DCL    THREE    FIXED BINARY    (31,0)    INITIAL(3);
```

7. この呼び出しは、データベースからデータをリトリーブします。これには修飾された SSA が含まれています。修飾された SSA を使用する呼び出しを出す前に、SSA のデータ・フィールドを初期設定してください。非修飾 SSA を使用する呼び出しを出す前に、セグメント名フィールドを初期設定してください。それぞれの DL/I 呼び出しを出した後で、状況コードをチェックしてください。

   入り口ステートメントにリストした PCB パラメーターは、PL/I プログラムに対して POINTER データ・タイプとして宣言する必要がありますが、PL/I プログラム内の DL/I 呼び出しでは、PCB 名を渡すことも PCB ポインターを渡すこともできます。
8. これも、修飾された SSA で修飾された呼び出しです。
9. これは、データベースからデータをリトリーブする非修飾呼び出しです。これは Get Hold 呼び出しであるため、この後に `REPL` または `DLET` を出すことができます。
10. `REPL` 呼び出しは、最後の Get Hold 呼び出しでリトリーブされたセグメント内のデータを置き換えます。このデータは、呼び出しで参照した入出力域の内容によって置き換えられます。
11. RETURN ステートメントは、IMS に制御を戻します。

IMS 言語インターフェース・モジュールへの PL/I コードのバインド

IMS は、IMS に対する共通インターフェースを提供する言語インターフェース・モジュール (DFSLI000) を備えています。このモジュールをプログラムにバインドしなければなりません。

IMS 提供のプロシージャー (IMSPLI または IMSPLIGO) を使用すると、IMS が言語インターフェース・モジュールをアプリケーション・プログラムとバインドします。IMSPLI は、ユーザー・プログラムのコンパイルとバインドを行う、2 つのステップからなるプロシージャーです。 IMSPLIGO は、ユーザー・コンパイルをコンパイルし、バインドし、DL/I バッチ領域で実行する、3 ステップのプロシージャーです。これらのプロシージャーの使用方法については、「*IMS V15* システム定義」を参照してください。

PL/I での CICS オンライン・プログラム・コーディング

次のサンプル・コードは、PL/I で書かれたスケルトンの CICS オンライン・プログラムです。これは、UIB のアドレス可能度の定義方法および設定方法を示しています。

プログラムの右側の数字は、このプログラムの後の注の番号と対応しています。この種のプログラムは、DBCTL を使用して CICS 環境で実行できます。

呼び出しレベルの PL/I プログラム (CICS オンライン) のサンプル

```
PLIUIB: PROC OPTIONS(MAIN);                                          NOTES
 DCL PSB_NAME CHAR(8) STATIC INIT('PLIPSB  ');                        1
 DCL PCB_FUNCTION CHAR(4) STATIC INIT('PCB ');
```

```
  DCL TERM_FUNCTION CHAR(4) STATIC INIT('TERM');
  DCL GHU_FUNCTION CHAR(4) STATIC INIT('GHU ');
  DCL REPL_FUNCTION CHAR(4) STATIC INIT('REPL');
  DCL SSA1 CHAR(9) STATIC INIT('AAAA4444 ');                              2
  DCL PARM_CT_1 FIXED BIN(31) STATIC INIT(1);
  DCL PARM_CT_3 FIXED BIN(31) STATIC INIT(3);
  DCL PARM_CT_4 FIXED BIN(31) STATIC INIT(4);
  DCL GOOD_RETURN_CODE BIT(8) STATIC INIT('0'B);
  DCL GOOD_STATUS_CODE CHAR(2) STATIC INIT('  ');
  %INCLUDE DLIUIB;                                                        3
  DCL 1 PCB_POINTERS BASED(UIBPCBAL),                                     4
        2 PCB1_PTR POINTER;
  DCL 1 DLI_IO_AREA,                                                      5
        2 AREA1 CHAR(3),
        2 AREA2 CHAR(37);
  DCL 1 PCB1 BASED(PCB1_PTR),                                             6
        2 PCB1_DBD_NAME CHAR(8),
        2 PCB1_SEG_LEVEL CHAR(2),
        2 PCB1_STATUS_CODE CHAR(2),
        2 PCB1_PROC_OPTIONS CHAR(4),
        2 PCB1_RESERVE_DLI FIXED BIN (31,0),
        2 PCB1_SEGNAME_FB CHAR(8),
        2 PCB1_LENGTH_FB_KEY FIXED BIN(31,0),
        2 PCB1_NUMB_SENS_SEGS FIXED BIN(31,0),
        2 PCB1_KEY_FB_AREA CHAR(17);
      /* SCHEDULE PSB AND OBTAIN PCB ADDRESSES */
CALL PLITDLI (PARM_CT_3,PCB_FUNCTION,PSB_NAME,UIBPTR);                    7
IF UIBFCTR = GOOD RETURN CODE THEN DO;
   /* ISSUE DL/I CALL: GET A UNIQUE SEGMENT */
   CALL PLITDLI (PARM_CT_4,GHU_FUNCTION,PCB1,DLI_IO_AREA,SSA1);           8
   IF UIBFCTR = GOOD_RETURN_CODE& PCB1_STATUS_CODE = GOOD_STATUS_CODE THEN DO; 9
      /* PERFORM SEGMENT UPDATE ACTIVITY */
     AREA1 = ......;
     AREA2 = ......;
    /* ISSUE DL/I: REPLACE SEGMENT AT CURRENT POSITION */
      PLITDLI (PARM_CT_3,REPL_FUNCTION,PCB1,DLI_IO_AREA);                 10
    IF UIBFCTR ^= GOOD_RETURN_CODE
     | PCB1_STATUS_CODE ^= GOOD_STATUS_CODE THEN DO;
     /* INSERT REPL ERROR DIAGNOSTIC CODE */
    END;
   END;
   ELSE DO;
      /* INSERT GHU ERROR DIAGNOSTIC CODE */
   END;
END;
ELSE DO;
   /* ANALYZE UIB PROBLEM */
   /* ISSUE UIB DIAGNOSTIC MESSAGE */
END;
/* RELEASE THE PSB */
CALL PLITDLI(PARM_CT_1,TERM_FUNCTION);                                    11
EXEC CICS RETURN;                                                         12
END PLIUIB;
```

注:

1. これらの各区域は、プログラムが使用する DL/I 呼び出し機能を定義しています。各文字ストリングは 4 文字の英数字として定義され、各機能に割り立てられた値が入ります。他の定数も同様に定義できます。標準定義をソース・ライブラリーに保管させ、%INCLUDE ステートメントを使用してそれらの定義を組み込むこともできます。
2. 構造定義は、プログラムが使用する各 SSA を定義します。 SSA には、非境界合わせ属性を使用する必要があります。 SSA 文字ストリングは、ストレー

ジ内で連続していなければなりません。複数の SSA が必要な呼び出しの場合には、追加の区域を定義する必要があります。

3. %INCLUDE DLIUIB ステートメントが展開されます。
4. UIB は、PCB アドレスが入っている区域のアドレスを戻します。実際の PCB アドレスを得るためには、PCB ポインターの定義が必要です。この区域内のアドレスは変更しないでください。
5. データベースとの間でセグメントを受け渡すために使用される入出力域は、構造として定義されます。
6. PCB は、UIB に入れて渡されたアドレスにもとづいて定義されます。
7. PCB 呼び出しは、ユーザーのプログラムで使用するために PSB をスケジュールします。
8. この非修飾 GHU 呼び出しはデータベースからセグメントをリトリーブします。セグメントは、呼び出しで参照されている入出力域に入れます。この呼び出しを出す前に、プログラムは、リトリーブする特定のセグメントを指定するために SSA のキーまたはデータ値を初期設定しなければなりません。
9. CICS オンライン・プログラムでは、DB PCB 内の状況コードを検査する前に、UIB に戻された戻りコードを検査する必要があります。
10. この REPL 呼び出しは、最後の Get Hold 呼び出しでリトリーブされたセグメントを置き換えます。呼び出しで参照される入出力域に、置き換え対象のセグメントが入ります。
11. TERM 呼び出しは、プログラムが以前にスケジュールした PSB を終了させます。
12. 処理が終了すると、プログラムは EXEC CICS RETURN ステートメントを出します。

関連資料**:** アプリケーション・プログラムのインストールについて詳しくは、「*CICS Transaction Server for z/OS CICS* アプリケーション・プログラミング・ガイド」を参照してください。

関連資料:

270 ページの『UIB の指定 (CICS オンライン・プログラムのみ)』

238 ページの『PL/I での CICS オンライン・プログラム・コーディング』

第 12 章 IMS DB のアプリケーション・プログラム・エレメントの定義

アセンブラー言語、C 言語、COBOL、Pascal、および PL/I で作成するアプリケーション・プログラムでは、言語インターフェースから DL/I 呼び出しを行うために、以下の固有のパラメーターおよび形式を使用します。

言語インターフェースのための DL/I 呼び出しのフォーマット設定

アセンブラー言語、C 言語、COBOL、Pascal、または PL/I おける DL/I 呼び出しを使用する場合には、DL/I 言語インターフェースを呼び出して、DL/I 呼び出しで指定された機能を開始する必要があります。

IMS では、DL/I 呼び出しへのインターフェースには以下のものがあります。

- Language Environment®準拠の任意のプログラムのための言語非依存インターフェース (CEETDLI)
- サポートされるすべての言語に対応する言語固有の各インターフェース (xxxTDLI)
- サポートされるすべての言語に対応する言語固有でないインターフェース (AIBTDLI)

Java では、3 つすべての DL/I 言語インターフェースを使用しますが、その使用は内部的なものであるため、DL/I 呼び出しで指定した機能を開始する際に、呼び出しは必要ありません。

関連概念:

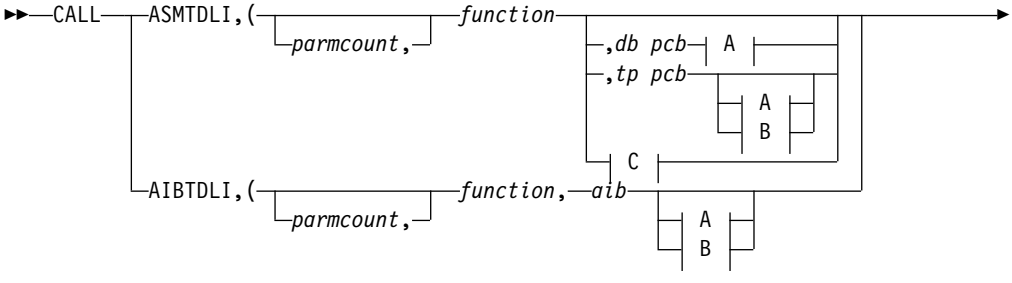
693 ページの『第 38 章 Java 開発用の IMS ソリューションの概要』

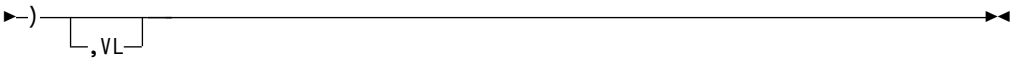
アセンブラー言語によるアプリケーション・プログラミング

アセンブリー言語によるアプリケーション・プログラムは、以下の形式、パラメーター、および DL/I 呼び出しを使用して、IMS データベースと通信します。

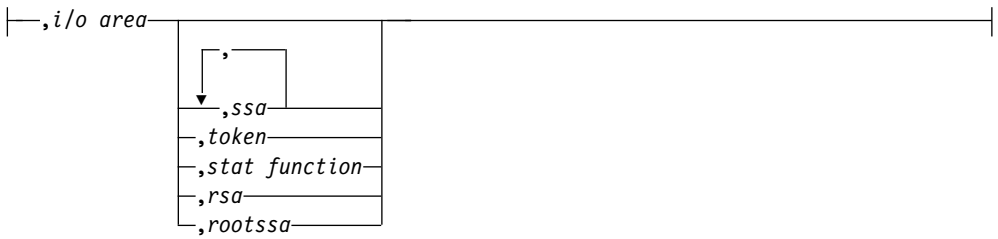
アセンブラー言語プログラムでは、アドレスとして渡されるすべての DL/I 呼び出しパラメーターはレジスターで渡すことができます。ただし、使用する際には括弧で囲まなければなりません。

フォーマット

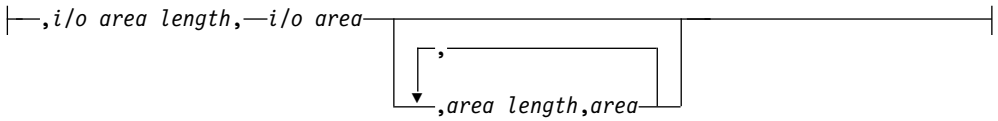




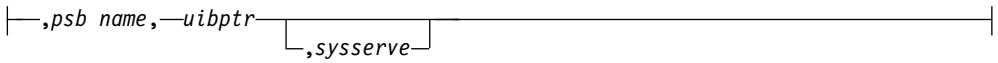
A:



B:



C:



パラメーター

parmcount
ユーザー定義ストレージの、4 バイト・フィールドのアドレスを指定します。ここには、*parmcount* の後に続くパラメーター・リスト内のパラメーターの数が入ります。アセンブラー言語のアプリケーションのプログラムでは、必ず *parmcount* か **VL** のいずれかを使用してください。

function
ユーザー定義ストレージ内の、呼び出し機能が入る 4 バイト・フィールドのアドレスを指定します。呼び出し機能は左寄せし、ブランクを埋め込む必要があります (例えば、GUbb)。

db pcb
呼び出しに使用するデータベース PCB のアドレスを指定します。PCB アドレスは、アプリケーション・プログラムの入り口で PCB リストに入れて渡される PCB アドレスの 1 つでなければなりません。

tp pcb
呼び出しに使用する I/O PCB または代替 PCB のアドレスを指定します。PCB アドレスは、アプリケーション・プログラムの入り口で PCB リストに入れて渡される PCB アドレスの 1 つでなければなりません。

aib
ユーザー定義ストレージ内のアプリケーション・インターフェース・ブロック (AIB) のアドレスを指定します。

i/o area
ユーザー定義ストレージ内の、呼び出しで使用される入出力域のアドレスを指定します。入出力域は、返されるデータが入るだけの大きさがなければなりません。

i/o area length
ユーザー定義ストレージの、4 バイトのフィールドのアドレスを指定します。ここには、入出力域の長さが入ります (2 進数で指定)。

area length
ユーザー定義ストレージの、4 バイトのフィールドのアドレスを指定します。ここには、パラメーター・リスト内のすぐあとの区域の長さが入ります (2 進数で指定)。7 つまでの area length または area ペアを指定できます。

area
チェックポイントをとるユーザー定義ストレージの区域のアドレスを指定します。 7 つまでの area length または area ペアを指定できます。

token
ユーザー定義ストレージの、4 バイトのフィールドのアドレスを指定します。ここには、ユーザー・トークンが入ります。

stat function
ユーザー定義ストレージ内の、実行すべき stat 機能が入る 9 バイト・フィールドのアドレスを指定します。

ssa
呼び出しのために使用する SSA が入る、ユーザー定義ストレージ内のアドレスを指定します。最大 15 個までの SSA を指定できます。そのうちの 1 つは *rootssa* です。

rootssa
ユーザー定義ストレージ内のルート・セグメント検索引数のアドレスを指定します。

rsa
ユーザー定義ストレージ内の、レコード検索引数が入る区域のアドレスを指定します。

psb name
ユーザー定義ストレージ内の、呼び出しで使用される 8 バイトの PSB 名のアドレスを指定します。

uibptr
ユーザー定義ストレージ内の、ユーザー・インターフェース・ブロック (UIB) のアドレスを指定します。

sysserve
ユーザー定義ストレージ内の、呼び出しで使用される 8 バイト・フィールドのアドレスを指定します。

VL
パラメーター・リストの終わりを示します。アセンブラー言語のプログラムでは、必ず *parmcount* または **VL** のいずれかを使用してください。

DL/I 呼び出し形式の例

DL/I AIBTDLI インターフェースの使用法:

```
CALL    AIBTDLI,(function,aib,i/o area,ssa1),VL
```

DL/I 言語固有インターフェースの使用法:

```
CALL    ASMTDLI,(function,db pcb,i/o area,ssa1),VL
```

関連概念:

279 ページの『AIBTDLI インターフェース』

関連資料:

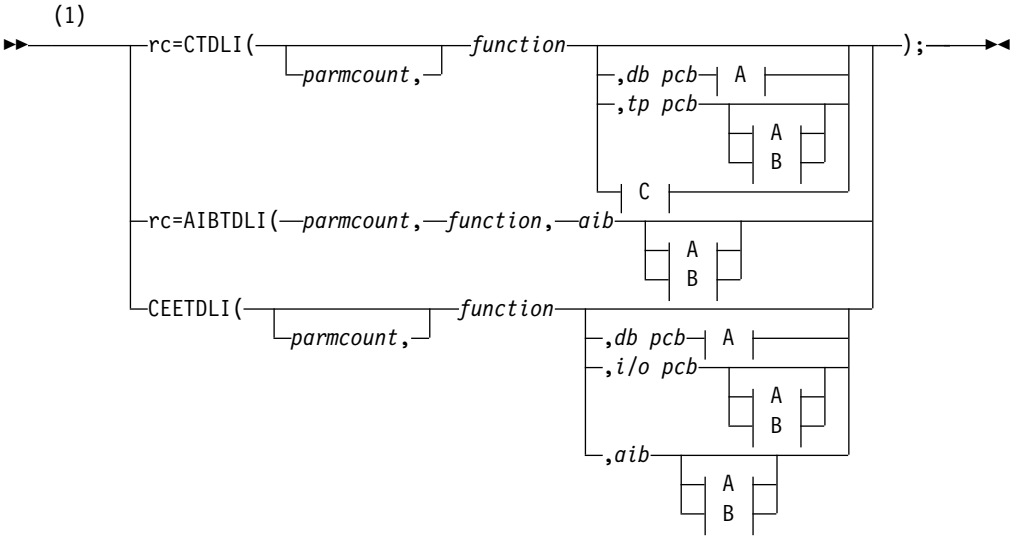
データベース管理のための DL/I 呼び出し (アプリケーション・プログラミング API)

IMS DB システム・サービスのための DL/I 呼び出し (アプリケーション・プログラミング API)

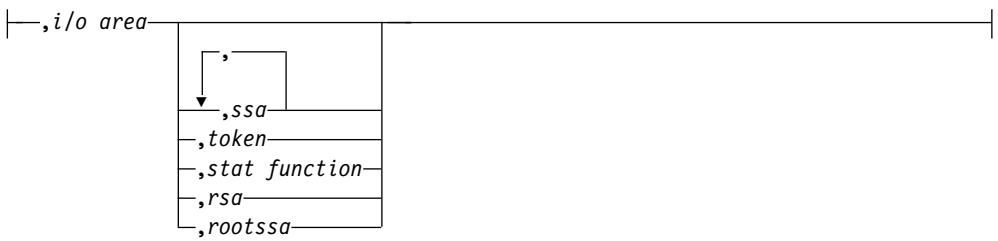
C 言語によるアプリケーション・プログラミング

C 言語によるアプリケーション・プログラムは、以下の形式、パラメーター、および DL/I 呼び出しを使用して、IMS データベースと通信します。

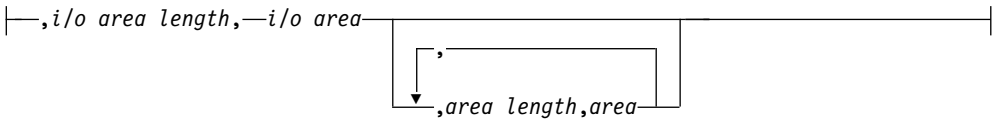
フォーマット



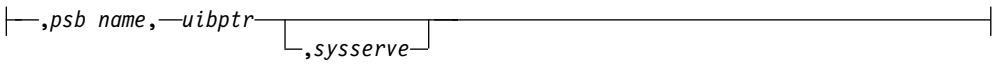
A:



B:



C:



注:

1 AIBTDLI の場合、C アプリケーションには *parmcount* が必要です。

パラメーター

rc このパラメーターは、DL/I の状況コードまたは戻りコードを受け取るためのものです。これは、整変数 (int) の下位 2 バイトにシフトする 2 文字フィールドです。状況コードが 2 つのブランクである場合、このフィールドにはゼロが入ります。**rc** パラメーターは、**if** ステートメントで検査できます。例えば、`if (rc == 'IX')` を使用します。また、**switch** ステートメントで **rc** を使用することもできます。 **rc** に入った値を無視して、その代わりに PCB に戻された状況コードを使用することもできます。

parmcount
パラメーター・リストにおける parmcount の後のパラメーターの数を含むユーザー定義ストレージ内の固定長 2 進 (31) 変数の名前を指定します。

function
ユーザー定義ストレージ内の、使用する呼び出し機能を含む、左揃えされた文字 (4) 変数の名前を指定します。呼び出し機能は左寄せし、ブランクを埋め込む必要があります (例えば、GUbb)。

db pcb
呼び出しで使用されるデータベースのアドレスを含む、ポインター変数の名前を指定します。PCB アドレスは、常に、アプリケーション・プログラムへの入り口で PCB リストで渡される PCB アドレスの 1 つでなければなりません。

tp pcb
ポインター変数の名前を指定します。ここには、呼び出しで使用される I/O PCB アドレスまたは代替 PCB アドレスが含まれます。 PCB アドレスは、アプリケーション・プログラムの入り口で PCB リストに入れて渡される PCB アドレスの 1 つでなければなりません。

aib
ポインター変数の名前を指定します。ここには、ユーザー定義ストレージのアプリケーション・インターフェース・ブロック (AIB) を定義する構造のアドレスが入ります。

i/o area
呼び出しで使用されるユーザー定義ストレージ内の入出力域を定義する、大構

造、配列、または文字ストリングを指すポインター変数の名前を指定します。入出力域は、戻されたデータをすべて収めるために十分な長さでなければなりません。

i/o area length
入出力域の長さを含む、ユーザー定義ストレージ内の固定長 2 進 (31) 変数の名前を指定します。

area length
パラメーター・リストにおける直後の区域の長さを含む、ユーザー定義ストレージ内の固定長 2 進 (31) 変数の名前を指定します。7 つまでの area length または area ペアを指定できます。

area
ポインター変数の名前を指定します。ここには、チェックポイントをとるユーザー定義ストレージを定義する構造のアドレスが入ります。 7 つまでの area length または area ペアを指定できます。

token
ユーザー定義ストレージの文字 (4) 変数の名前を指定します。ここには、ユーザー・トークンが入ります。

stat function
ユーザー定義ストレージ内の、実行すべき stat 機能が入る文字 (9) 変数の名前を指定します。

ssa
呼び出しのために使用する SSA が入る、ユーザー定義ストレージ内の文字変数の名前を指定します。最大 15 個までの SSA を指定できます。そのうちの 1 つは *rootssa* です。

rootssa
ユーザー定義ストレージ内のルート・セグメント検索引数を定義する、文字変数の名前を指定します。

rsa
GU 呼び出しのレコード検索引数を含む文字変数の名前、あるいは IMS が ISRT または GN 呼び出しの *rsa* を戻す文字変数の名前を指定します。

psb name
呼び出しで使用される PSB 名が入る文字 (8) 変数の名前を指定します。

uibptr
ユーザー定義ストレージ内で使用されるユーザー・インターフェース・ブロック (UIB) を定義する構造のアドレスを含む、ポインター変数の名前を指定します。

sysserve
ユーザー定義ストレージ内の、呼び出しで使用される文字 (8) 変数ストリングの名前を指定します。

入出力域

C では、入出力域は、構造または配列を含む任意の型にできます。 **ims.h** 内の **ctdli** 宣言には、プロトタイプ情報が含まれていないため、パラメーターの型検査は実行されません。この区域は、**auto** または **static** であっても、また (**malloc** または **calloc**) で割り振られたものであってもかまいません。C では、ヌルでストリン

グを終わらせる ('¥0') きまりですが、DL/I はそれに従いません。したがって、C ストリングには特別の注意が必要です。通常の **strcpy** 関数と **strcmp** 関数の代わりに、**memcpy** および **memcmp** を使用するとよいでしょう。

DL/I 呼び出し形式の例

DL/I CEETDLI インターフェースの使用法:

```
#include <leawi.h>
⋮
CEETDLI (function,db pcb,i/o area,ssa1);
```

DL/I AIBTDLI インターフェースの使用法:

```
int rc;
⋮
rc=AIBTDLI (parmcount,function,aib,i/o area,ssa1);
```

DL/I 言語固有インターフェースの使用法:

```
#include <ims.h>
int rc;
⋮
rc=CTDLI (function,db pcb,i/o area,ssa1);
```

関連概念:

279 ページの『AIBTDLI インターフェース』

関連資料:

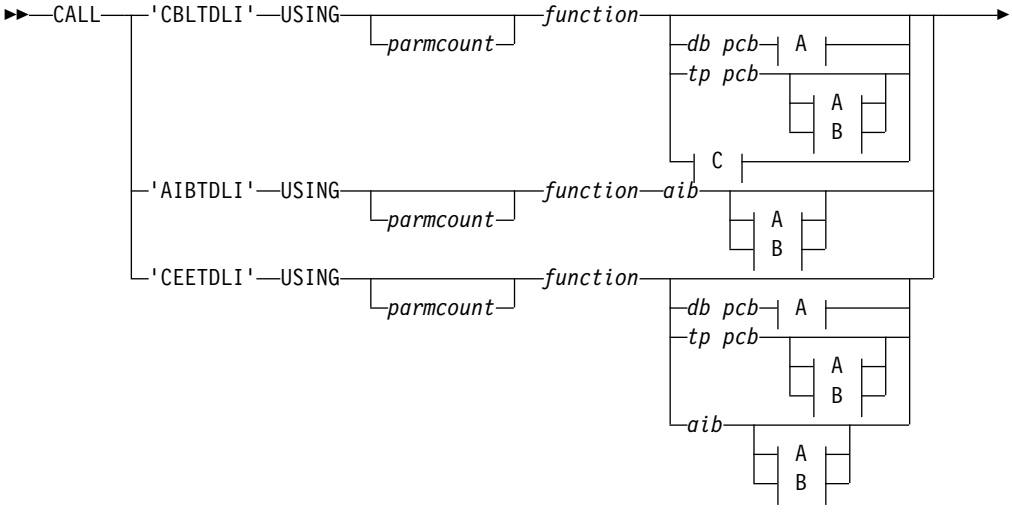
データベース管理のための DL/I 呼び出し (アプリケーション・プログラミング API)

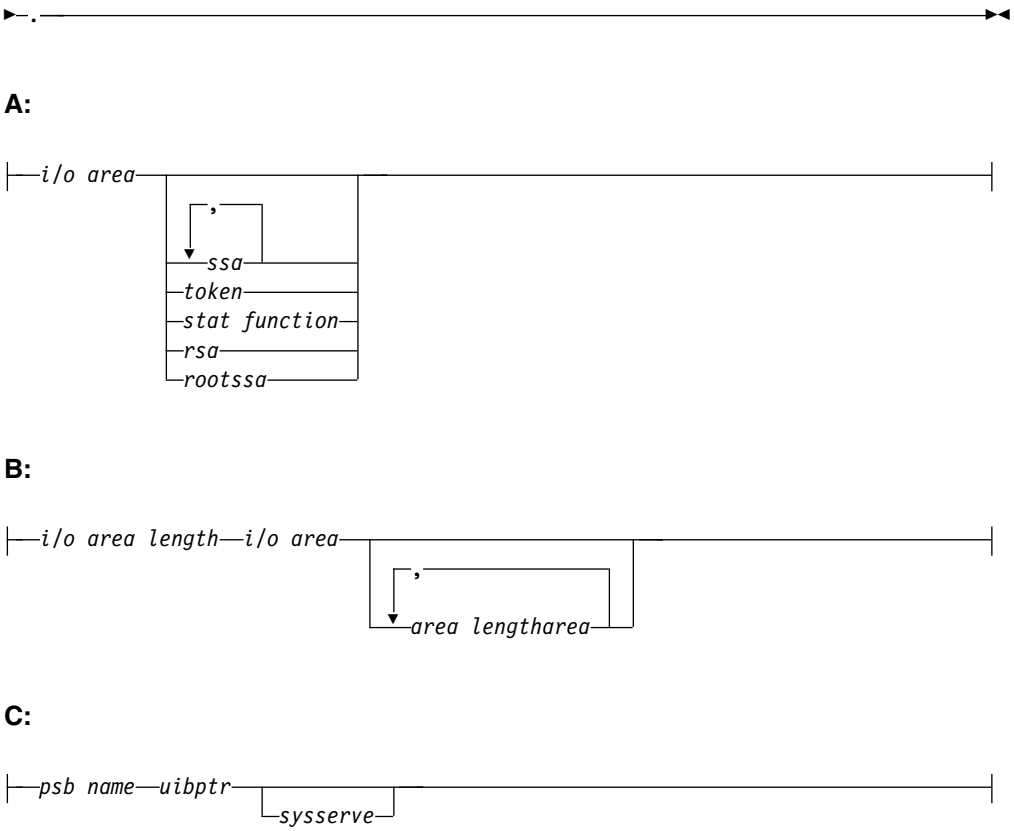
IMS DB システム・サービスのための DL/I 呼び出し (アプリケーション・プログラミング API)

COBOL によるアプリケーション・プログラミング

COBOL によるアプリケーション・プログラムは、以下の形式、パラメーター、および DL/I 呼び出しを使用して、IMS データベースと通信します。

フォーマット





注: すべてのアポストロフィ (') は引用符 (") で置き換えることができ、APOST/QUOTE コンパイラー (または CICS 変換プログラム) オプションに関係なく置き換えられます。

パラメーター

parmcount
パラメーター・リストにおける *parmcount* の後のパラメーターの数が入る、ユーザー定義ストレージ内の使用法 2 進 (4) バイト・データ項目の ID を指定します。このフィールドを COMP-5 (COMP、COMP-4、または BINARY ではなく) と定義した場合、COBOL TRUNC コンパイラー・オプションの設定にかかわらず、そのフィールドにはとりうる最大値が入ります。

function
ユーザー定義ストレージ内の、使用される呼び出し機能が入る、左揃えされた使用法表示 (4) バイト・データ項目の ID を指定します。呼び出し機能は左寄せし、ブランクを埋め込む必要があります (例えば、GUbb)。

db pcb
アプリケーション・プログラムの入り口で渡される、 PCB リスト内のデータベース PCB グループ項目の ID を指定します。この ID が呼び出しに使用されます。

tp pcb
エントリーにあるアプリケーション・プログラムへ渡された PCB リストから、I/O PCB または代替 PCB グループ項目の ID を指定します。この ID が呼び出しに使用されます。

aib
ユーザー定義ストレージ内のアプリケーション・インターフェース・ブロック (AIB) を定義するグループ項目の ID を、指定します。

i/o area
呼び出しで使用されるユーザー定義ストレージ内の入出力域の長さを定義する、大グループ項目、表、または使用表示データ項目の ID を指定します。入出力域は、戻されたデータをすべて収めるために十分な長さでなければなりません。

i/o area length
(2 進数で指定された) 入出力域の長さが入る、ユーザー定義ストレージ内の使用法 2 進 (4) バイト・データ項目の ID を指定します。このフィールドを COMP-5 (COMP、COMP-4、または BINARY ではなく) と定義した場合、COBOL TRUNC コンパイラー・オプションの設定にかかわらず、そのフィールドにはとりうる最大値が入ります。

area length
パラメーター・リストにおける直後の区域の (2 進数で指定された) 長さが入る、ユーザー定義ストレージ内の使用法 2 進 (4) バイト・データ項目の ID を指定します。7 つまでの area length または area ペアを指定できます。このフィールドを COMP-5 (COMP、COMP-4、または BINARY ではなく) と定義した場合、COBOL TRUNC コンパイラー・オプションの設定にかかわらず、そのフィールドにはとりうる最大値が入ります。

area
チェックポイントを取るべきユーザー定義ストレージを定義するグループ項目の ID を指定します。7 つまでの area length または area ペアを指定できます。

token
ユーザー定義ストレージ内の、ユーザー・トークンが入る使用法表示 (4) バイト・データ項目の ID を指定します。

stat function
ユーザー定義ストレージ内の、実行すべき stat 機能が入る使用法表示 (9) バイト・データ項目の ID を指定します。

ssa
呼び出しのために使用する SSA が入る、ユーザー定義ストレージ内の使用法表示データ項目の ID を指定します。最大 15 個までの SSA を指定できます。そのうちの 1 つは *rootssa* です。

rootssa
ユーザー定義ストレージ内の、ルート・セグメント検索引数を定義する使用法表示データ項目の ID を指定します。

rsa
レコード検索引数が入る使用法表示データ項目の ID を指定します。

psb name
呼び出しで使用される PSB 名が入る、使用法表示 (8) バイト・データ項目の ID を指定します。

uibptr
ユーザー定義ストレージ内で使用される、ユーザー・インターフェース・ブロック (UIB) を定義するグループ項目の ID を指定します。

sysserve
ユーザー定義ストレージ内の、呼び出しに使用される使用法表示 (8) バイト・データ項目の ID を指定します。

DL/I 呼び出し形式の例

DL/I CEETDLI インターフェースの使用法:

```
CALL    'CEETDLI' USING function,db pcb,i/o area,ssa1.
```

DL/I AIBTDLI インターフェースの使用法:

```
CALL    'AIBTDLI' USING function,aib,i/o area,ssa1.
```

DL/I 言語固有インターフェースの使用法:

```
CALL    'CBLTDLI' USING function,db pcb,i/o area,ssa1.
```

関連資料:

データベース管理のための DL/I 呼び出し (アプリケーション・プログラミング API)

IMS DB システム・サービスのための DL/I 呼び出し (アプリケーション・プログラミング API)

IMS 用の Java アプリケーション・プログラミング

IMS は Java プログラミング言語を使用したアプリケーション開発にサポートを提供します。

Java 開発用の IMS ソリューションのドライバーおよびリソース・アダプターを使用して、IMS データベースへのアクセスや IMS トランザクションの処理を実行する Java アプリケーションを作成できます。

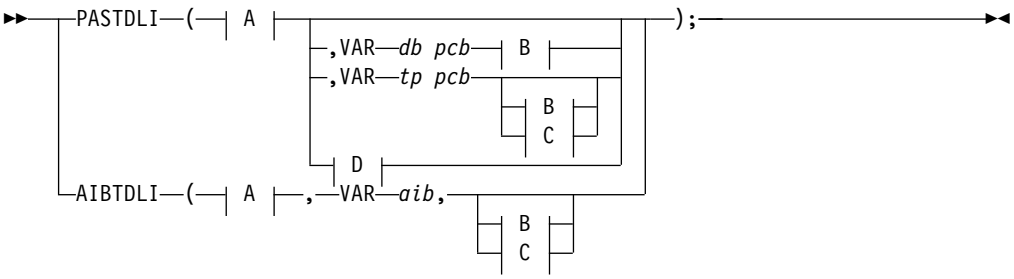
関連概念:

693 ページの『第 38 章 Java 開発用の IMS ソリューションの概要』

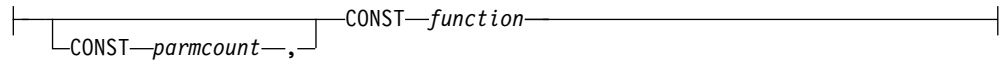
Pascal によるアプリケーション・プログラミング

Pascal によるアプリケーション・プログラムは、以下の形式、パラメーター、および DL/I 呼び出しを使用して、IMS データベースと通信します。

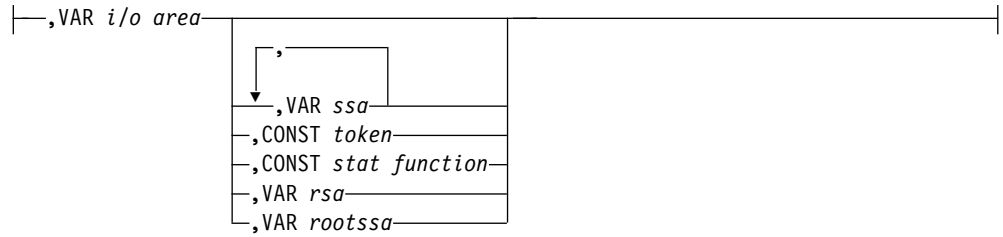
フォーマット



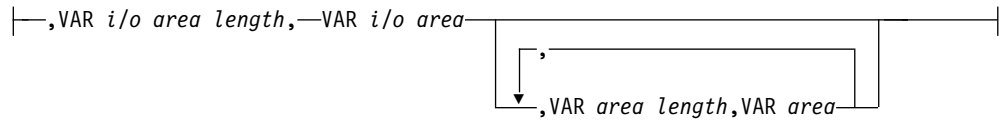
A:



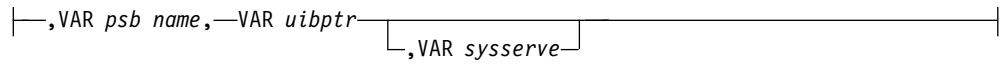
B:



C:



D:



パラメーター

parmcount
パラメーター・リストにおける parmcount の後のパラメーターの数を含むユーザー定義ストレージ内の固定長 2 進 (31) 変数の名前を指定します。

function
ユーザー定義ストレージ内の、使用する呼び出し機能を含む、左揃えされた文字 (4) 変数の名前を指定します。呼び出し機能は左寄せし、ブランクを埋め込む必要があります (例えば、GUbb)。

db pcb
呼び出しプロシージャー・ステートメントで定義されたデータベース PCB のアドレスを含む、ポインター変数の名前を指定します。

tp pcb
呼び出しプロシージャー・ステートメントで定義された I/O PCB または代替 PCB のアドレスを含む、ポインター変数の名前を指定します。

aib
ポインター変数の名前を指定します。ここには、ユーザー定義ストレージのアプリケーション・インターフェース・ブロック (AIB) を定義する構造のアドレスが入ります。

i/o area
呼び出しで使用されるユーザー定義ストレージ内の入出力域を定義する、大構造、配列、または文字ストリングを指すポインター変数の名前を指定します。入出力域は、戻されたデータをすべて収めるために十分な長さでなければなりません。

i/o area length
入出力域の長さを含む、ユーザー定義ストレージ内の固定長 2 進 (31) 変数の名前を指定します。

area length
パラメーター・リストにおける直後の区域の長さを含む、ユーザー定義ストレージ内の固定長 2 進 (31) 変数の名前を指定します。7 つまでの area length または area ペアを指定できます。

area
ポインター変数の名前を指定します。ここには、チェックポイントをとるユーザー定義ストレージを定義する構造のアドレスが入ります。 7 つまでの area length または area ペアを指定できます。

token
ユーザー定義ストレージの文字 (4) 変数の名前を指定します。ここには、ユーザー・トークンが入ります。

stat function
ユーザー定義ストレージ内の、実行すべき stat 機能が入る文字 (9) 変数の名前を指定します。

ssa
呼び出しのために使用する SSA が入る、ユーザー定義ストレージ内の文字変数の名前を指定します。最大 15 個までの SSA を指定できます。そのうちの 1 つは *rootssa* です。

rootssa
ユーザー定義ストレージ内のルート・セグメント検索引数を定義する、文字変数の名前を指定します。

rsa
レコード検索引数が入る文字変数の名前を指定します。

psb name
呼び出しで使用される PSB 名が入る文字 (8) 変数の名前を指定します。

uibptr
ユーザー定義ストレージ内で使用されるユーザー・インターフェース・ブロック (UIB) を定義する構造のアドレスを含む、ポインター変数の名前を指定します。

sysserve
ユーザー定義ストレージ内の、呼び出しで使用される文字 (8) 変数ストリングの名前を指定します。

DL/I 呼び出し形式の例

DL/I AIBTDLI インターフェースの使用法**:**

```
AIBTDLI(CONST function,
  VAR aib,
  VAR i/o area,
  VAR ssa1);
```

DL/I 言語固有インターフェースの使用法**:**

```
PASTDLI(CONST function,
  VAR db pcb,
  VAR i/o area,
  VAR ssa1);
```

関連資料:

データベース管理のための DL/I 呼び出し (アプリケーション・プログラミング API)

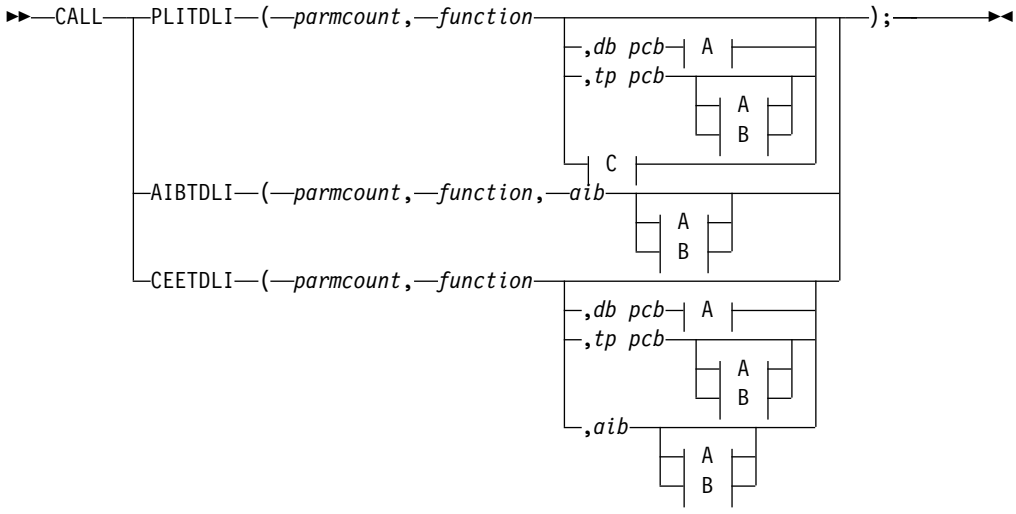
IMS DB システム・サービスのための DL/I 呼び出し (アプリケーション・プログラミング API)

PL/I のアプリケーション・プログラミング

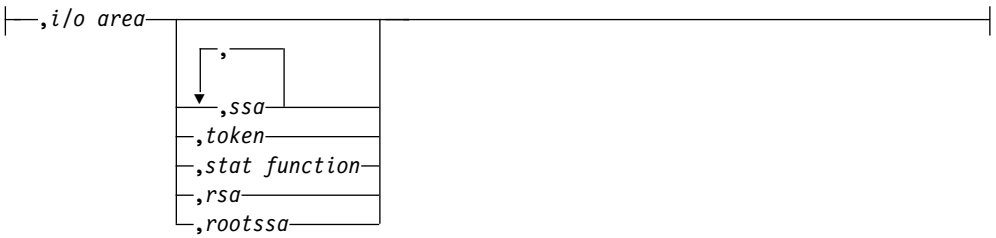
PL/I によるアプリケーション・プログラムは、以下の形式、パラメーター、および DL/I 呼び出しを使用して、IMS データベースと通信します。

制約事項**:** PLITDLI インターフェースでは、*parmcount* 以外のすべてのパラメーターは間接ポインターです。AIBTDLI インターフェースでは、すべてのパラメーターは直接ポインターです。

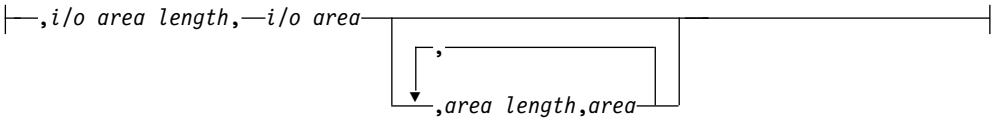
フォーマット



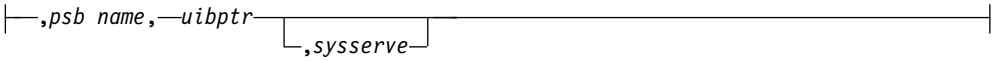
A:



B:



C:



パラメーター

parmcount
parmcount の後の引数の数が入る、固定長 2 進 (31 ビット) 変数の名前を指定します。

function
使用する呼び出し機能が含まれている固定長文字 (4 バイト) の変数 (左寄せしてブランクが埋め込まれた文字ストリング) の名前を指定します (例えば、GUbb)。

db pcb
呼び出しに使用するデータベース PCB に関連した構造を指定します。この構造は、アプリケーション・プログラムへの入り口で渡される PCB アドレスの 1 つでなければならない PCB アドレスに基づいています。

tp pcb
呼び出しに使用する I/O PCB または代替 PCB に関連した構造を指定します。

aib
アプリケーション・プログラム内の AIB を定義する構造の名前を指定します。

i/o area
呼び出しで使用する入出力域の名前を指定します。入出力域は、戻されたデータをすべて収めるために十分な長さでなければなりません。

i/o area length
入出力域長が入る固定長 2 進 (31) 変数の名前を指定します。

area length
パラメーター・リストにおける直後の区域の長さが入る、固定長 2 進 (31) 変数の名前を指定します。7 つまでの area length または area ペアを指定できます。

area
チェックポイントをとる区域の名前を指定します。7 つまでの area length または area ペアを指定できます。

token
文字 (4) 変数の名前を指定します。ここには、ユーザー・トークンが入ります。

stat function
実行すべき stat 機能が入る文字 (9) 変数ストリングの名前を指定します。

ssa
呼び出しで使用される SSA が入る文字変数の名前を指定します。最大 15 個までの SSA を指定できます。そのうちの 1 つは *rootssa* です。

rootssa
ルート・セグメント検索引数が入る文字変数の名前を指定します。

rsa
レコード検索引数が入る文字変数の名前を指定します。

psb name
呼び出しで使用される PSB 名が入る文字 (8) 変数の名前を指定します。

uibptr
ユーザー・インターフェース・ブロック (UIB) の名前を指定します。

sysserve
呼び出しで使用される文字 (8) 変数ストリングの名前を指定します。

DL/I 呼び出し形式の例

DL/I CEETDLI インターフェースの使用法:

```
CALL    CEETDLI (parmcount,function,db pcb,i/o area,ssa1);
```

DL/I AIBTDLI インターフェースの使用法:

```
CALL    AIBTDLI (parmcount,function,aib,i/o area,ssa1);
```

DL/I 言語固有インターフェースの使用法:

```
%INCLUDE CEEIBMAW;
CALL    PLITDLI (parmcount,function,db pcb,i/o area,ssa1);
```

関連資料:

データベース管理のための DL/I 呼び出し (アプリケーション・プログラミング API)

IMS DB システム・サービスのための DL/I 呼び出し (アプリケーション・プログラミング API)

I/O PCB マスクの指定

ユーザーのプログラムが入出力プログラム連絡ブロック (I/O PCB) を使用して呼び出しを出すと、IMS は、I/O PCB に呼び出しの結果に関する情報を戻します。呼び出しの結果を判別するには、IMS が返す情報をプログラムでチェックする必要があります。

システム・サービス呼び出しを発行するには、I/O PCB が必要です。 I/O PCB はユーザーのプログラムの外部にあるため、ユーザーは自分のプログラム内に PCB のマスクを定義し、IMS 呼び出しの結果をチェックする必要があります。マスクには、I/O PCB と同じフィールドが同じ順序で入らなければなりません。このようにすると、プログラムで、PCB マスクを介して PCB 内のフィールドを参照することができます。

次の表は、I/O PCB に含まれる各フィールド、そのフィールド長、および各フィールドに適用できる環境を示しています。

表 *34. I/O PCB* マスク

記述子	長さ **(**バイト数**)**	**DB/DC**	**DBCTL**	**DCCTL**	**DB** バッチ	**TM** バッチ
論理端末名 [1]	8	X		X		
IMS に予約済み [2]	2	X		X		
状況コード [3]	2	X	X	X	X	X
4 バイトの ローカル日付および 時間 [4]						
日付	4	X		X		
時間	4	X		X		
入力メッセージの順序番号 [5]	4	X		X		
メッセージ出力記述子名 [6]	8	X		X		
ユーザー ID [7]	8	X		X		
グループ名 [8]	8	X		X		
12 バイトのタイム・スタンプ [9]						
日付	4	X		X		
時間	6	X		X		
UTC オフセット	2	X		X		
ユーザー ID 標識[10]	1	X		X		
IMS に予約済み [2]	3					

注**:**

1. 論理端末名

   このフィールドには、メッセージを送信した端末の名前が入ります。ユーザーのプログラムが入力メッセージを受け取ると、IMS は、メッセージを送信した論理端末の名前をこのフィールドに入れます。この端末にメッセージを送り返すときは、ISRT 呼び出しを発行するときに I/O PCB を参照してください。IMS は、I/O PCB からの論理端末名を宛先とします。

2. **IMS** に予約済み

   これらのフィールドは予約済みです。

3. 状況コード

IMS は、DL/I 呼び出しの結果を記述する状況コードを、このフィールドに入れます。プログラムが出す各 DL/I 呼び出しの後で、IMS は状況コードを更新します。DL/I 呼び出しを発行した後は、必ずプログラムで状況コードをテストしてください。

状況コードは、次の 3 つのカテゴリーに分類されます。

- 成功した状況コード、または例外だが有効な条件を伴う状況コード。このカテゴリーには、エラーは入りません。呼び出しが完全に成功した場合は、このフィールドにはブランクが入ります。このカテゴリーに属するコードのうちの多くは、情報のみを表すものです。例えば、QC 状況コードは、プログラムのメッセージ・キューにこれ以上メッセージが存在しないことを意味します。この状況コードを受け取ったときは、プログラムを終了してください。
- プログラミング・エラー。このカテゴリーに属するエラーは、通常は訂正可能です。例えば、AD 状況コードは、無効な機能コードを示します。
- 入出力エラーまたはシステム・エラー

2 つ目と 3 つ目のカテゴリーでは、ユーザー・プログラムに、プログラムの終了前に出された最後の呼び出しに関する情報を印刷するエラー・ルーチンを指定する必要があります。ほとんどのインストール先では、そのインストール先のすべてのアプリケーション・プログラムで使用される標準のエラー・ルーチンが用意されています。

4. ローカル日時

ローカルの現在日時は、非メッセージ・ドリブン BMP から発信されたものを除き、すべての入力メッセージの接頭部に置かれます。ローカル日付は、右寄せしたパック 10 進数で、`yyddd` の形式です。ローカル時刻はパック 10 進数の時刻であり、形式は `hhmmsst` です。ローカルの現在日時は、いつ IMS がメッセージ全体を受け取り、メッセージをプログラムに対する入力データとしてキューに入れたかを示すもので、アプリケーション・プログラムがメッセージを受け取った時刻ではありません。アプリケーションの処理時刻を入手するには、使用するプログラミング言語の時刻機構を使用する必要があります。

会話の場合、プログラムから発信される入力メッセージの場合、または複数システム結合機能 (MSC) を使用して受け取られたメッセージの場合には、時刻および日付は、元のメッセージがいつ端末から受け取られたかを示します。

注: I/O PCB のローカル日時と、オペレーティング・システムから戻された現在時刻を比較する際は注意してください。I/O PCB 日時は現在時刻と整合していない場合があります。この時刻は、以下の理由により現在時刻より進んでいることもあります。

- I/O PCB のタイム・スタンプは IMS がメッセージを受信した地方時です。メッセージの受信後に地方時が変更された場合、現在時刻が I/O PCB 時刻より見かけ上早くなる可能性があります。これは、秋の時刻変更でクロックを 1 時間遅らせて設定した直後の 1 時間に起こる場合があります。
- I/O PCB のタイム・スタンプはメッセージと共に保管された内部の IMS タイム・スタンプから派生しています。この内部タイム・スタンプは協定世界時 (UTC) にあり、メッセージがエンキューされた際に有効であった時間

帯オフセットを含みます。この時間帯オフセットは、UTC 時間に追加され、I/O PCB に配置されている地方時が入手されます。ただし、保管されている時間帯オフセットは 15 分のみです。実際の時間帯オフセットが 15 分の整数倍でない場合、I/O PCB で戻された地方時は実際の時間と 7.5 分前後異なります。これにより I/O PCB 時刻が現在時刻より遅れることになり場合があります。詳細については、「*IMS V15* オペレーションおよびオートメーション」を参照してください。

I/O PCB にあるローカルのタイム・スタンプの値に関する不安事項は、 IMS V6 で導入された拡張タイム・スタンプの使用により削減されます。システム管理者は拡張タイム・スタンプの形式を地方時にも UTC にもなるように選択できます。状態によっては、アプリケーションがオペレーティング・システムから UTC での時刻を要求し、その時刻と拡張タイム・スタンプの UTC フォームを比較すると有効な場合があります。これは、地方時の変更に際して、IMS UTC オフセットと z/OS UTC オフセットとの同期を保つ ETR がないインストール・システムで選択可能なオプションです。

5. 入力メッセージの順序番号

   入力メッセージ・シーケンス番号は、非メッセージ・ドリブンの BMP から発信されたものを除き、すべての入力メッセージの接頭部に置かれます。このフィールドには、IMS が入力メッセージに割り当てた順序番号が入ります。数値は 2 進数です。 IMS は、物理端末ごとに順序番号を割り当てます。この番号は、IMS の最後の始動時から連続しています。

6. メッセージ出力記述子名

   このフィールドを使用するのは、MFS を使用するときだけです。メッセージ出力記述子 (MOD) を伴う `GU` 呼び出しが出されると、IMS はこの区域にその名前を入れます。プログラムがエラーを検出した場合には、画面の形式を変更し、このフィールドを使用して端末にエラー・メッセージを送ることができます。これを行うためには、`ISRT` 呼び出しまたは `PURG` 呼び出しで MOD 名のパラメーターを指定して、プログラムで MOD 名を変更する必要があります。

   MFS は APPC をサポートしませんが、LU 6.2 プログラムでインターフェースを使用すると、MFS をエミュレートすることができます。例えば、アプリケーション・プログラムでエラー・メッセージの形式設定の方法を指定する場合、MOD 名を使用して IMS と通信することができます。

   関連資料**:** MOD 名と LTERM インターフェースの詳細については、「*IMS V15* コミュニケーションおよびコネクション」を参照してください。

7. ユーザー **ID**

   このフィールドは、RACF サインオン・セキュリティーとともに使用されます。システム内でサインオンがアクティブでない場合には、このフィールドにはブランクが入ります。

   サインオンがシステムでアクティブになっている場合、このフィールドには次のいずれかが入ります。

   - 発信元端末からのユーザーの識別名。

- 発信元端末の LTERM 名 (その端末についてサインオンがアクティブでない場合)。
- 許可 ID。バッチ指向 BMP の場合の許可 ID は、PROCLIB メンバー DFSDCxxx の中の BMPUSID= キーワードに指定されている値によって、次のように異なります。
  - BMPUSID=USERID が指定されている場合、JOB ステートメントの USER= キーワードの値が使用されます。
  - JOB ステートメントで USER= が指定されていない場合は、プログラムの PSB 名が使用されます。
  - BMPUSID=PSBNAME が指定されている場合、または BMPUSID= がまったく指定されていない場合は、プログラムの PSB 名が使用されます。 PSBNAME が RACF に対して定義されていない場合は、現行アドレス・スペースのユーザー ID が使用されます。これは、LSO=Y または PARDLI=1 が BMP に対して指定されている場合、ホーム従属領域のものか、制御領域のものになります。DFSBSEX0 が RC08 を返した場合も、現行アドレス・スペースのユーザー ID が使用されます。

    関連資料: 従属領域のリソース使用の許可の詳細については、「*IMS V15* システム管理」を参照してください。

8. グループ名

   SQL 呼び出しのセキュリティーのため DB2 により使用されるグループ名は、IMS トランザクションによって作成されます。

   グループ名に適用される 3 つのインスタンスは次のとおりです。

   - IMS システムで RACF およびサインオンを使用している場合、RACROUTE SAF (抽出) 呼び出しは 8 文字のグループ名を返します。
   - IMS システムでセキュリティー・パッケージを使用する場合、RACROUTE SAF 呼び出しはそのパッケージから任意の 8 文字の名前を返し、それをグループ名として扱います。 RACROUTE SAF 呼び出しが 4 または 8 の戻りコードを返すと、グループ名は返されず、IMS はグループ名フィールドをすべてブランクにします。
   - LU 6.2 を使用する場合、トランザクション・ヘッダーにはグループ名を含めることができます。

     関連資料: LU 6.2 の詳細については、「*IMS V15* コミュニケーションおよびコネクション」を参照してください。

9. **12** バイト・タイム・スタンプ

   このフィールドに現在日付と時刻が収められますが、IMS 内部パック 10 進数形式になっています。タイム・スタンプは、以下の各部分から構成されています。

   日付 `yyyydddf`

   このパック 10 進数日付は、年 (yyyy)、年間通算日 (ddd)、および有効なパック 10 進数の + 記号、例えば (f) から成ります。

   時間 `hhmmssthmiju`

このパック 10 進数時刻は、時間、分、秒 (hhmmss)、およびマイクロ秒までの、秒の小数部分 (thmiju) から成ります。タイム・スタンプのこの部分には、パック 10 進数の符号は加えられていません。

UTC オフセット

aqq$

パック 10 進数 UTC オフセットの接頭部は、4 ビットから成る属性 (a) です。(a) の 4 番目のビットが 0 であれば、タイム・スタンプは UTC、そうでなければタイム・スタンプは、地方時です。DFSPBxxx PROCLIB メンバーで指定されている制御領域パラメーター TSR=(U/L) が、地方時または UTC 時刻のいずれかでのタイム・スタンプの表現を制御します。

オフセット値 (qq$) は、地方時または UTC 時刻に変換するためにそれぞれ UTC 時刻または地方時に加えるべき、4 分の 1 時間単位のオフセットです。

オフセットの符号 ($) は、パック 10 進数の正負の符号規則に従います。

I/O PCB マスクのフィールド 4 は、常にローカルの日時を持っています。フィールド 4 の説明については、前の表の注を参照してください。

関連資料: 内部パック 10 進数時刻形式の詳細については、「*IMS V15* システム・ユーティリティー」を参照してください。

10. ユーザー **ID** 標識

ユーザー ID 標識は、I/O PCB 内および INQY 呼び出しへの応答に入れられます。ユーザー ID 標識には、以下のいずれかが入ります。

- U - サインオン時にソース端末から得られたユーザー ID
- L - 送信元端末の LTERM 名 (サインオンがアクティブでない場合)
- P - 送信元 BMP またはトランザクションの PSBNAME
- O - その他の名前

ユーザー ID 標識フィールド内の値は、ユーザー ID フィールドの内容を示します。

DB PCB マスクの指定

IMS は、ユーザーのプログラムが出した呼び出しの結果を、その呼び出しで参照された DB PCB に記述します。DL/I 呼び出しが成功したか失敗したかを調べるために、アプリケーション・プログラムに DB PCB のマスクを含め、その DB PCB のマスクを介して DB PCB のフィールドを参照してください。

DB PCB マスクには、以下の表に示すフィールドが含まれている必要があります。(ユーザーのプログラムでは、DB PCB のフィールドを調べることはできますが、変更することはできません。)DB PCB マスクのフィールドは、ここで示すフィールドと同じ順序および同じ長さで定義しなければなりません。DB PCB マスクをコーディングする場合には、名前も指定しますが、その名前はマスクの一部ではありませ

ん。ユーザーのプログラムが処理する各 PCB を参照するには、この名前 (PL/I の場合にはポインター) を使用します。 GSAM の DB PCB マスクは、他の DB PCB マスクとは若干異なっています。

9 つのフィールドの中で、プログラムを作成するときに重要なものは 5 つのみです。これらのフィールドは、呼び出しの結果に関する情報を表すものです。すなわち、セグメント・レベル番号、状況コード、セグメント名、キー・フィードバック域の長さ、およびキー・フィードバック域です。状況コードは、呼び出しが成功したかどうかを調べるために、ユーザーのプログラムが最も頻繁に使用するフィールドです。キー・フィードバック域には、ユーザーが指定したセグメントから得られたデータが入ります。レベル番号とセグメント名は、非修飾 GN または GNP 呼び出しの後でリトリーブされたセグメント・タイプを判別したり、エラーの発生後または呼び出しの失敗後のデータベース内におけるユーザーの位置を判別したりするために役立ちます。

表 *35. DB PCB* マスク

記述子	長さ (バイト数)	DB/DC	DBCTL	DCCTL	DB バッチ	TM バッチ
データベース名 [1]	8	X	X		X	
セグメント・レベル番号 [2]	2	X	X		X	
状況コード [3]	2	X	X		X	
処理オプション [4]	4	X	X		X	
IMS に予約済み[5]	4	X	X		X	
セグメント名 [6]	8	X	X		X	
キーの長さフィードバック域 [7]	4	X	X		X	
センシティブ・セグメントの数 [8]	4	X	X		X	
キー・フィードバック域 [9]	可変長	X	X		X	

注:

1. ここには、データベースの名前が入ります。このフィールドは 8 バイト長で、文字データが入ります。
2. セグメント・レベル番号

   このフィールドには数字データが入ります。このフィールドは 2 バイト長で、右寄せされます。IMS は、要求されたセグメントをリトリーブする際に、そのセグメント・レベル番号をこのフィールドに入れます。階層パス内の複数のセグメントを 1 つの呼び出しでリトリーブする場合、IMS は、リトリーブした最も低いレベルのセグメントの番号をこのフィールドに入れます。IMS は、ユーザーが要求するセグメントが見つからない場合、ユーザーの呼び出しを満たしたセグメントのうちで最後に検出したもののレベル番号をこのフィールドに入れます。
3. 状況コード

各 DL/I 呼び出しが出されるたびに、このフィールドには DL/I 呼び出しの結果を記述する 2 文字の状況コードが入ります。各呼び出しが出されるたびに、IMS はこのフィールドを更新します。呼び出しと呼び出しの間、その値は消去されません。アプリケーション・プログラムは、各呼び出しの後でこのフィールドを検査して、その呼び出しが成功したかどうかを確認する必要があります。

このプログラムの最初のスケジュール時には、このフィールドにはデータ可用性状況コードが入ります。このコードは、セグメント・センシティビティーと処理オプションに基づく何らかのアクセス制約の可能性があることを示します。

関連資料: これらの状況コードの詳細については、「*IMS V15* アプリケーション・プログラミング *API*」のトピック『INIT 呼び出し』を参照してください。

通常の処理中における状況コードのカテゴリーには、次の 4 つのものがあります。

- 成功した状態、または例外ではあるが有効な状態。呼び出しが完全に成功した場合は、このフィールドにはブランクが入ります。このカテゴリーに属するコードのうちの多くは、情報のみを表すものです。例えば、GB は、要求が満たされないうちに IMS がデータベースの終わりに達したという意味です。この状況は、順次処理では予想されることであり、エラーの結果として生じるものではありません。
- プログラム内のエラー。例えば、AK はセグメント検索引数 (SSA) にユーザーが無効なフィールド名を指定したという意味です。ユーザーのプログラムでは、このような状況コードが戻された場合に使用できるエラー・ルーチンを用意しておく必要があります。IMS がユーザーのプログラムにエラー状況コードを戻した場合には、そのプログラムを終了させるようにしてください。そのうえで、問題を見つけだし、それを修正し、プログラムを再始動できます。
- 入出力エラーまたはシステム・エラー。例えば、AO 状況コードは、OSAM、BSAM、または VSAM に関する入出力エラーが発生したことを表します。ユーザーのプログラムは、このカテゴリーの状況コードを受け取ったときにはただちに終了する必要があります。このタイプのエラーは、通常、システム・プログラマー、データベース管理者、またはシステム管理者でなければ修正できません。
- データ使用可能性状況コード。これは、ユーザーのプログラムがこのような状況コードを扱う用意ができていることを示す `INIT` 呼び出しを出したときに限って戻されます。「*IMS V15* メッセージおよびコード 第 *4* 巻*: IMS* コンポーネント・コード」の中の『状況コードの説明』に、考えられる原因と修正方法が詳しく説明されています。

4. 処理オプション

   これは、このプログラムが出せる呼び出しのタイプを IMS に知らせるコードが入る 4 バイト・フィールドです。このフィールドは、特定のプログラムにデータベースの読み取りだけを行わせて、そのデータベースを更新できないようにする、セキュリティー機構として機能します。この値は、アプリケーション・プログラムの PSB を生成するときに、PCB ステートメントの PROCOPT パラメーターでコーディングされます。この値は、変更されることはありません。

5. **IMS** に予約済み

この 4 バイト・フィールドは、IMS により内部結合のために使用されます。アプリケーション・プログラムによって使用されることはありません。

6. セグメント名

   IMS は、呼び出しが正常に行われるたびに、その呼び出しを満たした最後のセグメントの名前をこのフィールドに入れます。リトリーブが成功すると、このフィールドにリトリーブされたセグメントの名前が入ります。リトリーブが成功しなかった場合には、このフィールドには、要求されたセグメントまでのパス内の、呼び出しの条件を満たした最後のセグメントが入ります。セグメント名フィールドの長さは 8 バイトです。

   プログラムが最初にスケジュールされるときには、データベース・タイプの名前がこの SEGNAME フィールドに入ります。例えば、データベース・タイプが DEDB であれば、このフィールドには DEDB が入ります。GSAM なら GSAM、HDAM なら HDAM、PHDAM なら PHDAM が入ります。

7. キー・フィードバック域の長さ

   これは、キー・フィードバック域の現行の長さを示す 4 バイトの 2 進数フィールドです。通常はキー・フィードバック域が呼び出しと呼び出しの間に消去されることはないため、プログラムは、キー・フィードバック域内の現行の関連する連結キーの長さを判別するために、この長さを使用する必要があります。

8. センシティブ・セグメントの数

   これは、アプリケーション・プログラムによってセンシティブと見なすデータベース内のセグメント・タイプの数が入る、4 バイトの 2 進数フィールドです。

9. キー・フィードバック域

   リトリーブまたは ISRT 呼び出しが完了すると、IMS は、リトリーブされたセグメントの連結キーをこのフィールドに入れます。この要求のキーの長さは、4 バイトのフィールドに示されます。 IMS が呼び出しの条件を満たすことができない場合、キー・フィードバック域には、条件が満たされた最後のレベルのセグメントのキーが入ります。セグメントの連結キーは、そのセグメントの親のそれぞれのキーと、そのセグメント自体のキーから成ります。各キーは、ルート・セグメントのキーから順に階層パスをたどって、左から右の順に配置されます。IMS は、通常、キー・フィードバック域を消去しません。IMS は、各呼び出しが完了すると、キー・フィードバック域のこの長さをセットして、区域の有効部分を示します。ユーザーのプログラムでは、キー・フィードバック域のうち、キー・フィードバック域の長さに含まれていない部分の内容を使用しないでください。

関連概念:

278 ページの『GSAM データベースのデータ域』

AIB マスクの指定

アプリケーションで PCB アドレスを受け取っていない場合、または呼び出し機能で PCB を使用しない場合に、プログラムは IMS との通信にアプリケーション・インターフェース・ブロック (AIB) を使用します。

使用可能であれば、アプリケーション・プログラムは、返された PCB アドレスを使用して PCB 内の状況コードを検査したり、そのアプリケーション・プログラムに必要なその他の情報を入手したりできます。 AIB マスクを使用すると、プログラムで制御ブロック定義を解釈することができます。 AIB の構造は、次の表に示すとおり、フィールドの順序およびバイト長に従って、作業用ストレージにフルワード境界で定義され、初期設定される必要があります。表の注は、各フィールドの内容を説明しています。

表 *36. AIB* フィールド

記述子	長さ **(**バイト数**)**	**DB/DC**	**DBCTL**	**DCCTL**	**DB** バッチ	**TM** バッチ
AIB ID [1]	8	X	X	X	X	X
DFSAIB 割り振り長さ [2]	4	X	X	X	X	X
副次機能コード [3]	8	X	X	X	X	X
リソース名 1[4]	8	X	X	X	X	X
リソース名 2 [5]	8	X	X	X	X	X
予約済み [6]	8					
出力域の最大長 [7]	4	X	X	X	X	X
使用される出力域の長さ [8]	4	X	X	X	X	X
リソース・フィールド [9]	4	X	X	X	X	X
オプション領域の長さ [10]	4	X	X	X	X	X
予約済み [11]	4	X	X	X	X	X
戻りコード [12]	4	X	X	X	X	X
理由コード [13]	4	X	X	X	X	X
エラー・コード拡張子 [14]	4	X	X	X		
リソース・アドレス 1 [15]	4	X	X	X	X	X
リソース・アドレス 2 [16]	4	X	X	X	X	X
リソース・アドレス 3 [17]	4	X	X	X	X	X
ユーザー定義トークン [18]	16	X	X	X	X	X
戻りトークン [19]	8	X	X		X	
予約済み [20]	16					

AIB ID (AIBID)

この 8 バイトのフィールドには、AIB ID が入ります。DL/I 呼び出しを出す前に、アプリケーション・プログラムで AIBID を DFSAIBbb に初期設定しておく必要があります。このフィールドは必須です。呼び出しが完了したとき、このフィールドに返された情報は変更されていません。

DFSAIB 割り振り長さ **(AIBLEN)**

このフィールドには、ユーザーのプログラムで定義された AIB の実際の長さが 4 バイトで入ります。DL/I 呼び出しを発行する前に、アプリケーション・プログラムで AIBLEN を初期設定しておく必要があります。最小でも 128 バイトの長さが必要です。呼び出しが完了しても、このフィールドに返される情報は変更されません。このフィールドは必須です。

副次機能コード **(AIBSFUNC)**

この 8 バイト・フィールドには、副次機能を使用する呼び出しのための副次機能コードが入ります。DL/I 呼び出しを発行する前に、アプリケーション・プログラムで AIBSFUNC を初期設定しておく必要があります。呼び出しが完了したとき、このフィールドに返された情報は変更されていません。

リソース名 **(AIBRSNM1)**

この 8 バイトのフィールドには、リソースの名前が入ります。リソースは呼び出しによって変わります。DL/I 呼び出しを発行する前に、アプリケーション・プログラムで AIBRSNM1 を初期設定しておく必要があります。呼び出しが完了しても、このフィールドに戻される情報は変更されません。このフィールドは必須です。

呼び出しリストで PCB アドレスを渡す代わりに、AIB を使用して PCB 名を渡す PCB 関連の呼び出しの場合、このフィールドには PCB 名が入ります。I/O PCB の PCB 名は IOPCBbb です。その他のタイプの PCB の PCB 名は、PSBGEN 内の PCBNAME= パラメーターで定義されています。

リソース名 **2 (AIBRSNM2)**

この 8 バイトのフィールドには、リソースの名前が入ります。リソースは呼び出しによって異なります。 DL/I 呼び出しを発行する前に、アプリケーション・プログラムで AIBRSNM2 を初期設定しておく必要があります。

予約済み

この 8 バイト・フィールドは予約されています。

出力域の最大長 **(AIBOALEN)**

この 4 バイトのフィールドには、呼び出しリストに指定された出力域の長さがバイト単位で入ります。出力域にデータを戻すすべての呼び出しについて、アプリケーション・プログラムで AIBOALEN を初期設定しておく必要があります。呼び出しが完了しても、この区域に戻される情報は変更されません。

使用された出力域の長さ **(AIBOAUSE)**

この 4 バイト・フィールドには、出力域にデータを返すすべての呼び出しで、IMS により返されるデータの長さが入ります。呼び出しが完了したとき、このフィールドには、この呼び出しに使用された入出力域の長さが入ります。

リソース・フィールド **(AIBRSFLD)**

この 4 バイト・フィールドにはリソースが含まれます。リソースは呼び出しによって変わります。DL/I 呼び出しを発行する前に、アプリケーション・プログラムで AIBRSFLD を初期設定しておく必要があります。

オプション領域の長さ **(AIBOPLEN)**

この 8 バイト・フィールドにはリソースが含まれます。リソースは呼び出しによって変わります。DL/I 呼び出しを発行する前に、アプリケーション・プログラムで AIBOPLEN を初期設定しておく必要があります。

予約済み

この 4 バイトのフィールドは予約済みです。

戻りコード **(AIBRETRN)**
: 呼び出しが完了すると、この 4 バイト・フィールドに戻りコードが入ります。

理由コード **(AIBREASN)**
: 呼び出しが完了すると、この 4 バイト・フィールドには、理由コードが入ります。

エラー・コード拡張子 **(AIBERRXT)**
: この 4 バイト・フィールドには、AIBRETRN の戻りコードおよび AIBREASN の理由コードに基づいて、追加のエラー情報が入ります。

リソース・アドレス **1 (AIBRSA1)**
: 呼び出しが完了すると、この 4 バイト・フィールドには呼び出し固有の情報が入ります。呼び出しリストで PCB アドレスを渡す代わりに、AIB を使用して PCB 名を渡す PCB 関連の呼び出しの場合、このフィールドは PCB アドレスを返します。

リソース・アドレス **2 (AIBRSA2)**
: 呼び出しが完了すると、この 4 バイト・フィールドには呼び出し固有の情報が入ります。

リソース・アドレス **3 (AIBRSA3)**
: 呼び出しが完了すると、この 4 バイト・フィールドには呼び出し固有の情報が入ります。

ユーザー定義トークン **(AIBUTKN)**
: この 16 バイト・フィールドにはユーザー定義トークンが含まれます。トークンは呼び出しによって異なります。

AIB 戻りトークン **(AIBRTKN)**
: AIB 戻りトークン。この 8 バイト・フィールドには、DL/I 呼び出しによって返されたトークンが入ります。これは、トークンを返す DL/I 呼び出しに固有の使用法です。

予約済み
: この 16 バイト・フィールドは予約済みです。

ODBA アプリケーションの AIB マスクの指定

次の表では、ODBA アプリケーションに対してアプリケーション・インターフェース・ブロック (AIB) マスクを指定するフィールドを説明しています。

後に続く注で、各フィールドの内容を説明しています。

表 *37. ODBA* アプリケーションで使用する *AIB* フィールド

AIB フィールド	長さ **(**バイト数**)**	**DB/DC**	**IMS DB**	**DCCTL**	**DB** バッチ	**TM** バッチ
AIB ID	8	X	X	X	X	X
DFSAIB 割り振り長さ	4	X	X	X	X	X
副次機能コード	8	X	X	X	X	X
リソース名 #1	8	X	X	X	X	X
リソース名 #2	8					

表 37. *ODBA* アプリケーションで使用する *AIB* フィールド (続き)

AIB フィールド	長さ **(**バイト数**)**	**DB/DC**	**IMS DB**	**DCCTL**	**DB** バッチ	**TM** バッチ
予約済み 1	8	X				
出力域の最大長	4	X	X	X	X	X
使用される出力域の長さ	4	X	X	X	X	X
予約済み 2	12					
戻りコード	4	X	X	X	X	X
理由コード	4	X	X	X	X	X
エラー・コード拡張	4	X				
リソース・アドレス #1	4	X	X	X	X	X
リソース・アドレス #2	4					
リソース・アドレス #3	4					
AIB 戻りトークン	8	X	X		X	
予約済み 3	32					
ODBA 用に予約済み	136					

AIB ID (AIBID)

この 8 バイトのフィールドには、AIB ID が入ります。DL/I 呼び出しを出す前に、アプリケーション・プログラムで AIBID を DFSAIBbb に初期設定しておく必要があります。このフィールドは必須です。呼び出しが完了しても、このフィールドに返される情報は変更されません。

DFSAIB 割り振り長さ **(AIBLEN)**

このフィールドには、ユーザーのプログラムで定義された AIB の実際の長さが 4 バイトで入ります。DL/I 呼び出しを発行する前に、アプリケーション・プログラムで AIBLEN を初期設定しておく必要があります。必要な最小の長さは 264 バイトです。呼び出しが完了しても、このフィールドに返される情報は変更されません。このフィールドは必須です。

副次機能コード **(AIBSFUNC)**

この 8 バイト・フィールドには、副次機能を使用する呼び出しのための副次機能コードが入ります。DL/I 呼び出しを発行する前に、アプリケーション・プログラムで AIBSFUNC を初期設定しておく必要があります。呼び出しが完了しても、このフィールドに返される情報は変更されません。

リソース名 **(AIBRSNM1) #1**

この 8 バイトのフィールドには、リソースの名前が入ります。リソースは呼び出しによって異なります。 DL/I 呼び出しを発行する前に、アプリケーション・プログラムで AIBRSNM1 を初期設定しておく必要があります。呼び出しが完了しても、このフィールドに戻される情報は変更されません。このフィールドは必須です。

呼び出しリストで PCB アドレスを渡す代わりに、AIB を使用して PCB 名を渡す PCB 関連の呼び出しの場合、このフィールドには PCB 名が入ります。I/O PCB の PCB 名は IOPCBbb です。その他のタイプの PCB の PCB 名は、PSBGEN 内の PCBNAME= パラメーターで定義されています。

リソース名 **(AIBRSNM2) #2**
: ODBA 始動テーブル DFSxxxx0 の 4 文字 ID を指定します。xxxx が 4 文字 ID です。

予約済み **1**
: この 8 バイト・フィールドは予約されています。

出力域の最大長 **(AIBOALEN)**
: この 4 バイトのフィールドには、呼び出しリストに指定された出力域の長さがバイト単位で入ります。出力域にデータを戻すすべての呼び出しについて、アプリケーション・プログラムで AIBOALEN を初期設定しておく必要があります。呼び出しが完了しても、この区域に戻される情報は変更されません。

使用された出力域の長さ **(AIBOAUSE)**
: この 4 バイト・フィールドには、出力域にデータを返すすべての呼び出しで、IMS により返されるデータの長さが入ります。呼び出しが完了したとき、このフィールドには、この呼び出しに使用された入出力域の長さが入ります。

予約済み **2**
: この 12 バイトのフィールドは予約済みです。

戻りコード **(AIBRETRN)**
: 呼び出しが完了すると、この 4 バイト・フィールドに戻りコードが入ります。

理由コード **(AIBREASN)**
: 呼び出しが完了すると、この 4 バイト・フィールドには、理由コードが入ります。

エラー・コード拡張子 **(AIBERRXT)**
: この 4 バイト・フィールドには、AIBRETRN の戻りコードおよび AIBREASN の理由コードに基づいて、追加のエラー情報が入ります。

リソース・アドレス **(AIBRSA1) #1**
: 呼び出しが完了すると、この 4 バイト・フィールドには呼び出し固有の情報が入ります。呼び出しリストで PCB アドレスを渡す代わりに、AIB を使用して PCB 名を渡す PCB 関連の呼び出しの場合、このフィールドは PCB アドレスを返します。

リソース・アドレス **(AIBRSA2) #2**
: この 4 バイト・フィールドは、ODBA 用に予約されています。

リソース・アドレス **(AIBRSA3) #3**
: この 4 バイト・トークンは、APSB 呼び出しで返されます。後の DLI 呼び出しと、このスレッドに関連する DPSB 呼び出しで必要とされます。

AIB 戻りトークン **(AIBRTKN)**
: AIB 戻りトークン。この 8 バイト・フィールドには、DL/I 呼び出しによって返されたトークンが入ります。これは、トークンを返す DL/I 呼び出しに固有の使用法です。

予約済み **3**
: この 32 バイト・フィールドは予約済みです。

ODBA 用に予約済み
この 136 バイト・フィールドは、ODBA 用に予約されています。

使用可能であれば、アプリケーション・プログラムは、返された PCB アドレスを使用して PCB 内の状況コードを検査したり、そのアプリケーション・プログラムに必要なその他の情報を入手したりできます。

COBOL AIB マスクの例

```
01  AIB.
    02 AIBRID             PIC x(8).
    02 AIBRLEN            PIC 9(9) USAGE BINARY.
    02 AIBRSFUNC          PIC x(8).
    02 AIBRSNM1           PIC x(8).
    02 AIBRSNM2           PIC x(8).
    02 AIBRESV1           PIC x(8).
    02 AIBOALEN           PIC 9(9) USAGE BINARY.
    02 AIBOAUSE           PIC 9(9) USAGE BINARY.
    02 AIBRESV2           PIC x(12).
    02 AIBRETRN           PIC 9(9) USAGE BINARY.
    02 AIBREASN           PIC 9(9) USAGE BINARY.
    02 AIBERRXT           PIC 9(9) USAGE BINARY.
    02 AIBRESA1           USAGE POINTER.
    02 AIBRESA2           USAGE POINTER.
    02 AIBRESA3           USAGE POINTER.
    02 AIBRESV4           PIC x(40).
    02 AIBRSAVE     OCCURS 18 TIMES USAGE POINTER.
    02 AIBRTOKN     OCCURS 6 TIMES  USAGE POINTER.
    02 AIBRTOKC           PIC x(16).
    02 AIBRTOKV           PIC x(16).
    02 AIBRTOKA     OCCURS 2 TIMES PIC 9(9) USAGE BINARY.
```

Assembler AIB マスクの例

```
DFSAIB   DSECT
AIBID    DS    CL8'DFSAIB'
AIBLEN   DS    F
AIBSFUNC DS    CL8
AIBRSNM1 DS    CL8
AIBRSVM2 DS    CL8
         DS    2F
AIBOALEN DS    F
AIBOAUSE DS    F
         DS    2F
         DS    H
         DS    H
AIBRETRN DS    F
AIBREASN DS    F
AIBRRXT  DS    F
AIBRSA1  DS    A
AIBRSA2  DS    A
AIBRSA3  DS    A
         DS    10F
AIBLL    EQU   *-DFSAIB
AIBSAVE  DS    18F
AIBTOKN  DS    6F
AIBTOKC  DS    CL16
AIBTOKV  DS    XL16
AIBTOKA  DS    2F
AIBAERL  EQU   *-DFSAIB
```

UIB の指定 (CICS オンライン・プログラムのみ)

ユーザーの CICS オンライン・プログラムと DL/I の間のインターフェースは、ユーザー・インターフェース・ブロック (UIB) に追加情報を入れてユーザーのプログラムに渡します。UIB には、PCB リストのアドレス、およびユーザーのプログラムが DB PCB 内の状況コードの検査に先立って検査する必要のある戻りコードが入ります。

プログラムで使用する PSB を獲得するために PCB 呼び出しを出すと、そのプログラムのための UIB が作成されます。ご使用のプログラムの範囲外の領域と同様に、プログラムに UIB の定義を含めて、UIB へのアドレス可能度を確立しておく必要があります。 CICS は、すべてのプログラミング言語に対して UIB の定義を提供します。

- COBOL プログラムでは、COPY DLIUIB ステートメントを使用してください。
- PL/I プログラムでは、%INCLUDE DLIUIB ステートメントを使用してください。
- アセンブラー言語プログラムでは、DLIUIB マクロを使用してください。

ユーザーのプログラムにとって、UIB 内の 3 つのフィールド UIBPCBAL、UIBFCTR、 UIBDLTR は重要です。 UIBPCBAL には、PCB アドレス・リストのアドレスが含まれています。ユーザーはこのフィールドを利用して、使用したい PCB のアドレスを得ることができます。ユーザーのプログラムは、DB PCB 内の状況コードのチェックに先立って、 UIBFCTR (および、場合によっては UIBDLTR) 内の戻りコードをチェックする必要があります。 UIBFCTR および UIBDLTR の内容がヌルでない場合に、DB PCB 内の状況コード・フィールドの内容は意味のないものになります。戻りコードは、「*IMS V15* メッセージおよびコード 第 *4* 巻*: IMS* コンポーネント・コード」のトピック『CICS-DL/I ユーザー・インターフェース・ブロックの戻りコード』で説明されています。

ユーザーのプログラムでは、UIB を定義するステートメントの直後に、 PCB アドレス・リストおよび PCB マスクを定義する必要があります。

次のサンプル・コードは、VS COBOL II プログラムでの COPY DLIUIB ステートメントの使用方法を示しています。

VS COBOL II の場合の UIB、PCB アドレス・リスト、および PCB マスクの定義

```
LINKAGE SECTION.

    COPY DLIUIB.
01  OVERLAY-DLIUIB REDEFINES DLIUIB.
    02  PCBADDR USAGE IS POINTER.
    02  FILLER PIC XX.

01  PCB-ADDRESSES.
    02  PCB-ADDRESS-LIST
            USAGE IS POINTER OCCURS 10 TIMES.
01  PCB1.
    02  PCB1-DBD-NAME PIC X(8).
    02  PCB1-SEG-LEVEL PIC XX.
        .
        .
        .
```

COBOL COPY DLIUIB サンプル集

```
01  DLIUIB.
*                                   Address of the PCB addr list
    02  UIBPCBAL PIC S9(8) COMP.
*                                   DL/I return codes
    02  UIBRCODE.
*                                   Return codes
        03  UIBFCTR  PIC X.
            88  FCNORESP     VALUE ' '.
            88  FCNOTOPEN    VALUE ' '.
            88  FCINVREQ     VALUE ' '.
            88  FCINVPCB     VALUE ' '.
*                                   Additional information
        03  UIBDLTR  PIC X.
            88  DLPSBNF      VALUE ' '.
            88  DLTASKNA     VALUE ' '.
            88  DLPSBSCH     VALUE ' '.
            88  DLLANGCON    VALUE ' '.
            88  DLPSBFAIL    VALUE ' '.
            88  DLPSBNA      VALUE ' '.
            88  DLTERMNS     VALUE ' '.
            88  DLFUNCNS     VALUE ' '.
            88  DLINA        VALUE ' '.
```

レベル 88 の項目内の値は印刷不能です。これらの値は、「*IMS V15* メッセージおよびコード 第 *4* 巻*: IMS* コンポーネント・コード」のトピック『CICS-DL/I ユーザー・インターフェース・ブロックの戻りコード』で説明されています。フィールド名の意味とその 16 進値を以下に示します。

FCNORESP
通常応答値 X'00'

FCNOTOPEN
未オープン値 X'0C'

FCINVREQ
無効な要求値 X'08'

FCINVPCB
無効 PCB 値 X'10'

DLPSBNF
PSB が見つからない値 X'01'

DLTASKNA
タスクが許可されていない値 X'02'

DLPSBSCH
PSB は既にスケジュール済み値 X'03'

DLLANGCON
言語の矛盾値 X'04'

DLPSBFAIL
PSB 初期設定の失敗値 X'05'

DLPSBNA
PSB が許可されていない値 X'06'

DLTERMNS
終了の不成功値 X'07'

DLFUNCNS
スケジュールされていない機能値 X'08'

DLINA
DL/I が活動状態でない値 X'FF'

次のサンプル・コードは、PL/I の場合の UIB、PCB アドレス・リスト、および PCB マスクの定義方法を示しています。

PL/I の場合の UIB、PCB アドレス・リスト、および PCB マスクの定義

```
DCL UIBPTR PTR;                     /* POINTER TO UIB             */
DCL 1 DLIUIB UNALIGNED BASED(UIBPTR),
                                    /* EXTENDED CALL USER INTFC BLK*/
  2 UIBPCBAL PTR,                   /* PCB ADDRESS LIST           */
  2 UIBRCODE,                       /* DL/I RETURN CODES          */
    3 UIBFCTR BIT(8) ALIGNED,       /* RETURN CODES               */
    3 UIBDLTR BIT(8) ALIGNED;       /* ADDITIONAL INFORMATION     */
```

次のサンプル・コードは、アセンブラー言語の場合の UIB、PCB アドレス・リスト、および PCB マスクの定義方法を示しています。

アセンブラー言語の場合の UIB、PCB アドレス・リスト、および PCB マスクの定義

```
DLIUIB   DSECT
UIB      DS    0F                 EXTENDED CALL USER INTFC BLK
UIBPCBAL DS    A                  PCB ADDRESS LIST
UIBRCODE DS    0XL2               DL/I RETURN CODES
UIBFCTR  DS    X                  RETURN CODE
UIBDLTR  DS    X                  ADDITIONAL INFORMATION
         DS    2X                 RESERVED
         DS    0F                 LENGTH IS FULLWORD MULTIPLE
UIBLEN   EQU   *-UIB              LENGTH OF UIB
```

関連資料:

228 ページの『COBOL での CICS オンライン・プログラム・コーディング』

238 ページの『PL/I での CICS オンライン・プログラム・コーディング』

220 ページの『アセンブラー言語での CICS オンライン・プログラム・コーディング』

280 ページの『言語固有のエントリー・ポイント』

入出力域の指定

入出力域は、アプリケーション・プログラムと IMS との間でセグメントの受け渡しを行うために使用します。

入出力域には、呼び出しのタイプによって以下のものが入ります。

- セグメントをリトリーブすると、IMS は要求されたセグメントを入出力域に入れます。
- 新しいセグメントを追加するときには、ユーザーは最初に、新しいセグメントを入出力域内で作成します。

- セグメントを修正するときには、ユーザーのプログラムは、まず最初にそのセグメントをリトリーブする必要があります。セグメントをリトリーブすると、IMS はセグメントを入出力域に入れます。

プログラムと IMS との間で渡されるレコード・セグメントの形式は、固定長でも可変長でもかまいません。その際にアプリケーション・プログラムにとって重要な相違点は 1 つだけです。すなわち、セグメントのデータ域の先頭にあるメッセージ・セグメントには 2 バイト (PLITDLI インターフェースの場合は 4 バイト) の長さのフィールドが含まれています。

IMS 呼び出しのための入出力域は、ユーザーのプログラムが IMS からリトリーブしたり、IMS に送信したりする最大のセグメントを収められる大きさでなければなりません。

ユーザーのプログラムが、D コマンド・コードを使用する Get または ISRT 呼び出しを出す場合、入出力域は、プログラムがリトリーブまたは挿入するセグメントの最大パスを収められる大きさでなければなりません。

セグメント検索引数 (SSA) のフォーマット設定

アセンブラー言語、C 言語、COBOL、Java、Pascal、および PL/I の各アプリケーション・プログラム内のセグメント検索引数は、ここで説明する規則とフォーマットに従ってコーディングする必要があります。

SSA コーディング規則

セグメント検索引数をコーディングするには、以下の規則を使用します。

- SSA はユーザー・プログラムのデータ域で定義してください。
- セグメント名フィールドは、
  - 8 バイト長でなければなりません。指定するセグメント名の長さが 8 バイト未満の場合には、左揃えにして右側にブランクを埋め込む必要があります。
  - ユーザーのアプリケーション・プログラムが使用する DBD 内で定義された、セグメント名を含んでいなければなりません。すなわち、正確なセグメント名を使用しなければ、SSA が無効になります。
  - あるいは、DL/I 呼び出しがコマンド・コード O を使用する場合、セグメント・フィールド名は、開始オフセットおよびリトリーブしたいデータの長さになります。開始オフセットは、物理セグメント定義に対する相対値で、1 から始まります。リトリーブ可能な最大長は、データベース・タイプの最大セグメント・サイズで、最小長は 1 です。標準フィールド名の代わりに、'*oooollll*' の形式で 2 つのフィールドが指定されます。*oooo* がオフセット位置、*llll* がリトリーブしたいデータの長さです。
- SSA にセグメント名しか含まれていない場合、9 番目のバイトはブランクになっていなければなりません。
- SSA に 1 つ以上のコマンド・コードが含まれている場合、
  - 9 番目のバイトは、アスタリスク (*) でなければなりません。
  - SSA に修飾ステートメントが含まれていない場合には、最後のコマンド・コードのあとにブランクを 1 つ続けなければなりません。SSA に修飾ステート

メントが含まれている場合には、コマンド・コードのあとに修飾ステートメントの左括弧が続いていなければなりません。

- SSA に修飾ステートメントが含まれている場合、
  - 修飾ステートメントは、左括弧で始まり、右括弧で終わらなければなりません。
  - セグメント名およびコマンド・コード (使用される場合) と左括弧の間には、ブランクがあってはなりません。
  - このフィールド名は 8 バイト長でなければなりません。フィールド名が 8 バイト未満の場合、左揃えして右側をブランクで埋めなければなりません。フィールド名は、アプリケーション・プログラムが使用している DBD 内で、指定されたセグメント・タイプに関して定義されていなければなりません。
  - フィールド名のあとに関係演算子が続きます。これは、2 バイト長の英字または記号で表す必要があります。以下の表に関係演算子をリストします。

表 *38.* 関係演算子

シンボル	英字	意味
=b=	EQ	等しい
>= または =>	GE	以上
<= または =<	LE	以下
>b>	GT	より大きい
<b<	LT	より小さい
¬= または =¬	NE	等しくない

  - 関係演算子の後に比較値が続きます。この値の長さは、フィールド名で指定したフィールドの長さに等しくなければなりません。この長さは、DBD で定義されています。比較値には、必要に応じて、数値の場合には先行ゼロを、また英字値の場合には後書きブランクを含める必要があります。比較値に括弧を含むことはできません。

- 1 つの SSA 内で複数の修飾ステートメント (ブール修飾ステートメント) を使用する場合、それらの修飾ステートメントを次のいずれかの記号で区切る必要があります。

  ***** または **&**
  従属 AND

  + または **|**
  論理 OR

  # 独立 AND

  これらの記号の 1 つは、その記号が接続する 2 つの修飾ステートメントの間に指定しなければなりません。
- 最後の修飾ステートメントのあとには、右括弧を続けなければなりません。

アプリケーション・プログラムで作成される SSA は、その SSA に関して PSB で割り振られたスペースを超過してはなりません。

関連資料: PSB SSA サイズの定義の詳細については、「*IMS V15* データベース・ユーティリティー」の PSBGEN ステートメントに関する説明を参照してください。

SSA コーディング形式

アセンブラー言語、C 言語、COBOL、Pascal、および PL/I でセグメント検索引数をコーディングするには、以下の形式を使用します。

アセンブラー言語による SSA 定義の例

以下のサンプル・コードは、コマンド・コードを使用せずに、修飾された SSA を定義する方法を示しています。この SSA でコマンド・コードを使用する場合には、8 バイトのセグメント名フィールドと、修飾ステートメントを開始する左括弧の間に、アスタリスク (*) とコマンド・コードをコーディングします。

```
*        CONSTANT AREA
⋮
SSANAME  DS    0CL26
ROOT     DC    CL8'ROOT    '
         DC    CL1'('
         DC    CL8'KEY     '
         DC    CL2' ='
NAME     DC    CLn'vv...v'
         DC    CL1')'
```

この SSA は次のようになります。

```
ROOTbbbb(KEYbbbbbb=vv...v)
```

C 言語による SSA 定義の例

コマンド・コードを使用しない非修飾 SSA は、C 言語では次のようになります。

```
    const struct {
    char seg_name_u[8];
    char blank[1];
} unqual_ssa = {"NAME    ", " "};
```

このようにコーディングした SSA を、非修飾 SSA を必要とする各 DL/I 呼び出しで使用する場合には、プログラム実行時に使用したいセグメント・タイプの名前を指定してください。ストリング・サイズ宣言により、構造の中に C のヌル終止符が現れないようになっていることにご注意ください。

もちろん、これを単一のストリングとして次のように宣言することもできます。

```
const char unqual_ssa[] = "NAME      "; /* 8 chars + 1 blank  */
```

DL/I は末尾のヌル文字を無視します。

SSA は、入出力域の項で説明した任意の方法で定義できます。

修飾された SSA を作成する最も簡単な方法は、**sprintf** 関数を使用する方法です。ただし、COBOL または PL/I で使用される方法と類似の方法で定義することもできます。

以下に示すのは、コマンド・コードのない修飾された SSA の例です。この SSA でコマンド・コードを使用するためには、8 バイトのセグメント名フィールドと、修飾ステートメントを開始する左括弧の間に、アスタリスク (*) およびコマンド・コードを入れてください。

```
struct {
    seg_name       char[8];
    seg_qual       char[1];
    seg_key_name   char[8];
    seg_opr        char[2];
    seg_key_value  char[n];
    seg_end_char   char[1];
} qual_ssa = {"ROOT    ", "(", "KEY     ", " =", "vv...vv", ")"};
```

もう 1 つの方法として、**sprintf** を使用して SSA をストリングとして定義する方法があります。プリプロセッサー指示 **#include <stdio.h>** を使用することを忘れないでください。

```
char qual_ssa[8+1+8+2+6+1+1]; /* the final 1 is for the  */
                                    /* trailing '¥0' of string */
sprintf(qual_ssa,
        ",
        "ROOT", "KEY", "=", "vvvvv");
```

あるいは、値だけを変更する場合には、次のような **sprintf** 呼び出しを使用できます。

```
sprintf(qual_ssa,
        "ROOT    (KEY      =, "vvvvv");
      /* 12345678 12345678 */
```

いずれの場合にも、SSA は次のようになります。

```
ROOTbbbb(KEYbbbbbb=vv...v)
```

COBOL SSA 定義の例

コマンド・コードを使用しない非修飾 SSA は、COBOL では次のようになります。

```
DATA DIVISION.
WORKING-STORAGE SECTION.
.
.
.
01  UNQUAL-SSA.
    02 SEG-NAME    PICTURE X(08)   VALUE '........'.
    02 FILLER      PICTURE X       VALUE ' '.
```

このようにコーディングした SSA を、非修飾 SSA を必要とする各 DL/I 呼び出しで使用する場合には、プログラム実行時に使用したいセグメント・タイプの名前を指定してください。

プログラムで使用する各 SSA を定義するには、01 レベルの作業用ストレージ項目を使用します。その後で、SSA に指定した名前を DL/I 呼び出しのパラメーターとして使用してください。この場合には、

```
UNQUAL-SSA,
```

次の SSA は、コマンド・コードを使用しない修飾された SSA の例です。この SSA でコマンド・コードを使用するためには、8 バイトのセグメント名フィールドと、修飾ステートメントを開始する左括弧の間に、アスタリスク (*) およびコマンド・コードを入れてください。

```
DATA DIVISION.
WORKING-STORAGE SECTION.
:
01  QUAL-SSA-MAST.
    02 SEG-NAME-M       PICTURE X(08)   VALUE 'ROOT    '.
    02 BEGIN-PAREN-M    PICTURE X       VALUE '('.
    02 KEY-NAME-M       PICTURE X(08)   VALUE 'KEY     '.
    02 REL-OPER-M       PICTURE X(02)   VALUE ' ='.
    02 KEY-VALUE-M      PICTURE X(n)    VALUE 'vv...v'.
    02 END-PAREN-M      PICTURE X       VALUE ')'.
```

この SSA は次のようになります。

```
ROOTbbbb(KEYbbbbbb=vv...v)
```

Pascal SSA 定義の例

コマンド・コードを使用しない非修飾 SSA は、Pascal では次のようになります。

```
type
  STRUCT = record
             SEG_NAME  : ALFA;
             BLANK     : CHAR;
           end;
const
  UNQUAL_SSA = STRUCT('NAME',' ');
```

この SSA を次のように単一ストリングで宣言することもできます。

```
const
  UNQUAL_SSA = 'NAME     ';
```

次の例に示す SSA は、コマンド・コードを使用しない修飾された SSA です。この SSA でコマンド・コードを使用するためには、8 バイトのセグメント名フィールドと、修飾ステートメントを開始する左括弧の間に、アスタリスク (*) およびコマンド・コードを入れてください。

```
type
  STRUCT = record
             SEG_NAME       : ALFA;
             SEG_QUAL       : CHAR;
             SEG_KEY_NAME   : ALFA;
             SEG_OPR        : CHAR;
             SEG_KEY_VALUE  : packed array[1..n] of CHAR;
             SEG_END_CHAR   : CHAR;
           end;
const
  QUAL_SSA = STRUCT('ROOT','(','KEY',' =','vv...v',')');
```

この SSA は次のようになります。

```
ROOTbbbb(KEYbbbbbb=vv...v)
```

PL/I SSA 定義の例

コマンド・コードを使用しない非修飾 SSA は、PL/I では次のようになります。

```
DCL 1  UNQUAL_SSA     STATIC UNALIGNED,
    2      SEG_NAME_U CHAR(8) INIT('NAME    '),
    2      BLANK      CHAR(1) INIT(' ');
```

このようにコーディングした SSA を、非修飾 SSA を必要とする各 DL/I 呼び出しで使用する場合には、プログラム実行時に使用したいセグメント・タイプの名前を指定してください。

PL/I では、構造宣言の中で SSA を定義します。IMS との SSA データ交換のためには、非位置合わせ属性を指定する必要があります。SSA 文字ストリングは、ストレージ内で連続していなければなりません。例えば、可変キー値を割り当てた結果、そのキー値によって位置合せ属性が変更されると、IMS が無効な SSA を作成する可能性があります。

キー・フィールドの値はセグメント・タイプによって異なるため、プログラムがアクセスする各セグメント・タイプごとに別個の SSA 構造が必要です。(キー値以外の) フィールドを初期設定した後では、SSA を再度変更する必要はないはずです。SSA は、入出力域の項で説明した任意の方法で定義できます。

以下に示すのは、コマンド・コードのない修飾された SSA の例です。この SSA でコマンド・コードを使用する場合は、8 バイトのセグメント名フィールドと、修飾ステートメントを開始する左括弧の間に、アスタリスク (*) およびコマンド・コードを入れてください。

```
DCL 1    QUAL_SSA              STATIC UNALIGNED,
         2       SEG_NAME                  CHAR(8) INIT('ROOT    '),
         2       SEG_QUAL                  CHAR(1) INIT('('),
         2       SEG_KEY_NAME              CHAR(8) INIT('KEY     '),
         2       SEG_OPR                   CHAR(2) INIT(' ='),
         2       SEG_KEY_VALUE             CHAR(n) INIT('vv...v'),
         2       SEG_END_CHAR              CHAR(1) INIT(')');
```

この SSA は次のようになります。

```
ROOTbbbb(KEYbbbbbb=vv...v)
```

関連概念:

799 ページの『SSAList インターフェースを使用したセグメント検索引数の指定』

GSAM データベースのデータ域

汎用順次アクセス方式 (GSAM) データベースは、バッチ・プログラム、バッチ指向 BMP、トランザクション指向 BMP、または JBP のいずれかとして実行できるアプリケーション・プログラムでのみ使用できます。 GSAM データベース呼び出しで使用されるプログラム連絡ブロック (PCB) マスクおよびレコード検索引数 (RSA) は、特殊な形式になっています。

GSAM の DB PCB マスクは、他の DB PCB マスクとは若干異なっています。相違するフィールドは、キー・フィードバック域の長さとキー・フィードバック域です。また、不定長レコードを使用する場合、リトリーブまたは挿入されるレコードの長さを指定するために追加フィールドが 1 つあります。

RSA は、GN および ISRT 呼び出しで戻すことができる、基本フォーマット・データ・セットの 8 バイト・トークンまたはラージ・フォーマット・データ・セットの

12 バイト・トークンです。アプリケーション・プログラムは RSA を保管し、後続の GU 呼び出しで使用することができます。

関連概念:

351 ページの『第 20 章 GSAM データベースの処理』

関連資料:

260 ページの『DB PCB マスクの指定』

AIBTDLI インターフェース

AIBTDLI は、アプリケーション・プログラムと IMS との間のインターフェースとして使用します。

制約事項**:** AIB 内のフィールドは、IMS で定義されている場合を除き、アプリケーション・プログラムで使用することはできません。

AIBTDLI インターフェースを使用するときには、AIB のリソース名フィールドに (PSBGEN で定義した) PCB 名を入れることにより、呼び出しに必要なプログラム連絡ブロック (PCB) を指定します。PCB アドレスは指定しません。 AIB には PCB 名が入るので、アプリケーション・プログラムは、PCB アドレスではなく PCB 名を参照します。アプリケーション・プログラムは、PCB リスト内の PCB の相対的な位置を知る必要はありません。呼び出しが完了すると、AIB は、アプリケーション・プログラムが渡した PCB 名に対応する PCB アドレスを返します。

DB PCB および代替 PCB の名前は、PSBGEN の間にユーザーが定義します。入出力 PCB はすべて、PCB 名 bbb で生成されます。生成済みプログラム仕様ブロック (GPSB) の場合は、IOPCBbbb という PCB 名の入出力 PCB が生成され、また TPPCB1bb という PCB 名の修正可能な代替 PCB が生成されます。

PCB 名を渡すことができるということは、PCB リスト内の相対的な PCB 番号を知る必要がないことを意味します。さらに、AIBTDLI インターフェースにより、アプリケーション・プログラムは、PCB リストにない PCB で呼び出しを行うことができます。LIST= キーワードは、PSBGEN の間に PCB マクロで定義され、その PCB が PCB リストに含まれるかどうかを制御します。

AIB は、AIBTDLI インターフェースを使用する DL/I 呼び出しのために IMS に渡されるユーザー定義のストレージ内にあります。呼び出しが完了したとき、AIB は IMS により更新されます。AIB に割り振るストレージは、128 バイト以上にする必要があります。

関連概念:

352 ページの『GSAM データベース用の PCB マスク』

関連資料:

449 ページの『PL/I のアプリケーション・プログラミング』
446 ページの『Pascal のアプリケーション・プログラミング』
440 ページの『C 言語によるアプリケーション・プログラミング』
437 ページの『アセンブラー言語によるアプリケーション・プログラミング』
241 ページの『アセンブラー言語によるアプリケーション・プログラミング』

言語固有のエントリー・ポイント

アセンブラー言語、C、COBOL、Pascal、または PL/I で作成されたアプリケーション・プログラムでは、制御は IMS からエントリー・ポイント を介して渡されます。

エントリー・ポイントは、PSB で定義されているとおりの順序で PCB を参照しなければなりません。各 DL/I 呼び出しをコーディングするときには、その呼び出しに使用する PCB を指定する必要があります。 CICS オンライン以外のすべての場合には、プログラムがアクセスできる PCB のリストが、エントリー・ポイントでプログラムに渡されます。CICS オンラインの場合には、「*IMS V15* アプリケーション・プログラミング *API*」のトピック『システム・サービス呼び出し: PCB』で説明されているように、最初に PSB をスケジュールしなければなりません。

Java アプリケーション・インターフェースなどの、AIB 構造 (AIBTDLI または CEETDLI) を使用するアプリケーション・インターフェースは、PCB 構造ではなく PCB 名を使用するので、入り口でアプリケーションに PCB リストを渡す必要はありません。

CICS オンライン・プログラムでは、PCB のアドレスは入り口ステートメントからではなく、ユーザー・インターフェース・ブロック (UIB) から得られます。

アプリケーションが、z/OS および VM 用の IBM Language Environment® (言語環境プログラム) で使用可能になっている場合は、この値はブランクにしておいてください。

アセンブラー言語のエントリー・ポイント

アセンブラー言語の DL/I プログラムへの入り口ステートメントには、任意の名前を使用できます。 IMS がアプリケーション・プログラムに制御を渡すときには、レジスター 1 には、可変長フルワード・パラメーター・リストのアドレスが入っています。リスト内の各語には、PCB のアドレスが入っています。このアドレスを上書きする前に、レジスター 1 の内容を保管します。 IMS は、リストの終わりを示すために、リストの最後のフルワードの上位バイトを X'80' に設定します。順方向および逆方向のチェーニングには、標準の z/OS リンケージ規則を使用してください。

C 言語のエントリー・ポイント

IMS がユーザーのプログラムに制御を渡すと、プログラムが使用する各 PCB のアドレスがポインターの形で渡されます。IMS によって起動されるプログラムでは、通常の **argc** および **argv** 引数は使用できません。 IMS パラメーター・リストは、**__pcblist** マクロを使用してアクセスできます。PCB は、**__pcblist[0]**、**__pcblist[1]** により直接参照することができます。または、より分かりやすい名前を付与するマクロを定義することもできます。正しいタイプを得るためには、I/O PCB を次のようにキャストする必要があることに注意してください。

```
(IO_PCB_TYPE *)(__pcblist[0])
```

C 言語プログラムの入り口ステートメントは、**main** ステートメントです。

```
#pragma runopts(env(IMS),plist(IMS))
#include <ims.h>

main()
{
.
.
.
}
```

env オプションでは、C 言語プログラムが実行される操作環境を指定します。例えば、C 言語プログラムが IMS のもとで呼び出され、IMS 機能を使用する場合には、**env(IMS)** を指定します。 **plist** オプションは、ユーザーの C 言語プログラムが起動される時に受け取る呼び出しパラメーターの形式を指定します。ユーザーのプログラムがシステム・サポート・サービス・プログラムによって呼び出される場合、ユーザーの主プログラムに渡されるパラメーターの形式を C 言語形式の **argv**、 **argc**、および **envp** に変換する必要があります。この変換を行うには、ユーザーの C 言語プログラムが受け取るパラメーター・リストの形式を指定する必要があります。 **ims.h** インクルード・ファイルには、PCB マスクの宣言が入ります。

終了する方法は以下の 3 とおりがあります。

- 明示的な **return** ステートメントなしでメイン・プロシージャーを終了する。
- **main** から **return** ステートメントを実行する。
- 任意の個所から **exit** または **abort** 呼び出しを実行するか、あるいは、**main** に戻る **longjmp** を出してから通常の戻りを行う。

C 言語プログラムでは、**system** 関数を使用して別のプログラムに制御権を渡すことができます。パラメーター引き渡しのための通常の規則が適用されます。この場合、情報を提供するには **argc** および **argv** 引数を使用できます。呼び出されたプログラムは、初期 **__pcblist** を使用できるようになります。

COBOL のエントリー・ポイント

プロシージャー・ステートメントは、必ず入出力 PCB を最初に、使用する代替 PCB を次に、使用する DB PCB を最後に参照しなければなりません。代替 PCB および DB PCB は、PSB に定義されている順番でリストされていなければなりません。

```
PROCEDURE DIVISION USING PCB-NAME-1 [,...,PCB-NAME-N]
```

IMS の過去バージョンでは、入り口ステートメントに USING をコーディングして PCB を参照できました。しかし、IMS では入り口ステートメント上のそのようなコーディングを引き続き受け入れます。

推奨事項: PCB を参照するには、入り口ステートメントではなくプロシージャー・ステートメントを使用してください。

Pascal のエントリー・ポイント

エントリー・ポイントは、REENTRANT プロシージャーとして宣言しなければなりません。IMS が Pascal プロシージャーに制御を渡すと、パラメーター・リスト内の最初のアドレスは Pascal で使用するために予約され、その他のアドレスはプログラムで使用する PCB となります。PCB タイプは、この入り口ステートメントの前

に定義しなければなりません。IMS インターフェース・ルーチン PASTDLI は、GENERIC 指令で宣言しなければなりません。

```
procedure ANYNAME(var SAVE: INTEGER;
                  var pcb1-name: pcb1-name-type[;
                  ...
                  var pcbn-name: pcbn-name-type]); REENTRANT;
procedure ANYNAME;
(* Any local declarations *)
  procedure PASTDLI; GENERIC;
begin
  (* Code for ANYNAME *)
end;
```

PL/I のエントリー・ポイント

入り口ステートメントは、プログラムの最初の実行可能ステートメントとして指定しなければなりません。 IMS がユーザーのプログラムに制御を渡すと、ユーザー・プログラムが使用する各 PCB のアドレスがポインターの形で渡されます。入り口ステートメントをコーディングするときには、必ず PCB 名ではなく、PCB を指すポインターとしてこのステートメントのパラメーターをコーディングします。

```
anyname: PROCEDURE (pcb1_ptr [,..., pcbn_ptr]) OPTIONS (MAIN);
:
RETURN;
```

入り口ステートメントは PL/I で有効な任意の名前にできます。

CEETDLI、AIBTDLI、および AERTDLI インターフェースの考慮事項

以下の考慮事項は、CEETDLI、AIBTDLI、および AERTDLI に適用されます。

CEETDLI の考慮事項は以下のとおりです。

- PL/I プログラムでは、CEETDLI のエントリー・ポイントは、CEEIBMAW 組み込みファイルに定義されます。また、そのエントリー・ポイントをユーザー自身で宣言することもできます。ただし、その場合は、アセンブラー言語入り口 (DCL CEETDLI OPTIONS(ASM);) として宣言しなければなりません。
- C 言語アプリケーション・プログラムの場合、**env(IMS)** および **plist(IMS)** を指定する必要があります。これらを指定すると、アプリケーション・プログラムは引数の PCB リストを受け入れることができます。CEETDLI 関数は、<leawi.h> の中で定義されます。また、CTDLI 関数は、<ims.h> の中で定義されます。

AIBTDLI の考慮事項は以下のとおりです。

- AIBTDLI インターフェースを C/MVS™、Enterprise COBOL、あるいは PL/I 言語のアプリケーション・プログラム用に使用する場合、異常終了インターセプトを抑制するための言語実行時オプション (すなわち、NOSPIE および NOSTAE) を指定する必要があります。ただし、言語環境プログラム準拠のアプリケーション・プログラムの場合、NOSPIE および NOSTAE 制限は除去されます。
- PL/I プログラムの場合には、AIBTDLI エントリー・ポイントは、アセンブラー言語の入り口 (DCL AIBTDLI OPTIONS (ASM) ;) として宣言しなければなりません。

- C 言語アプリケーションの場合、**env(IMS)** および **plist(IMS)** を指定する必要があります。これらを指定すると、アプリケーション・プログラムは引数の PCB リストを受け入れることができます。

AERTDLI の考慮事項は以下のとおりです。

- AERTDLI インターフェースを C/MVS、COBOL、あるいは PL/I 言語アプリケーション・プログラム用に使用する場合、異常終了インターセプトを抑制するための言語実行時オプション (すなわち、NOSPIE および NOSTAE) を指定しなければなりません。ただし、言語環境プログラム準拠のアプリケーション・プログラムの場合、NOSPIE および NOSTAE 制限は除去されます。
- PL/I プログラムの AERTDLI エントリー・ポイントは、アセンブラー言語入り口 (DCL AERTDLI OPTIONS(ASM);) として宣言しなければなりません。
- C 言語アプリケーションでは、**env(IMS)** と **plis(IMS)** を指定しなければなりません。この指定により、アプリケーション・プログラムは引数の PCB リストを受け入れることができます。
- AERTDLI は、31 ビットのアドレス可能度で制御を受け取らなければなりません。

関連資料:

270 ページの『UIB の指定 (CICS オンライン・プログラムのみ)』

プログラム連絡ブロック (PCB) リスト

アプリケーション・プログラムでは、PCB または GPSB リストを以下の形式でコーディングします。

PCB リストの形式

次の例は、PCB リストの一般形式を示しています。

```
[IOPCB]
[Alternate PCB ... Alternate PCB]
[DB PCB ... DB PCB]
[GSAM PCB ... GSAM PCB]
```

各 PSB には、少なくとも 1 つの PCB がなければなりません。ほとんどのシステム・サービス呼び出しには I/O PCB が必要です。トランザクション管理呼び出しには、I/O PCB または代替 PCB が必要です。 (代替 PCB は IMS TM に存在します。) DL/I データベース用の DB PCB は、DBCTL のもとで IMS Database Manager でのみ使用されます。GSAM PCB は、DCCTL で使用できます。

GPSB PCB リストの形式

生成済みプログラム仕様ブロック (GPSB) は、次の形式になります。

```
[IOPCB]
[Alternate PCB]
```

GPSB には、入出力 PCB と変更可能代替 PCB が各 1 つだけ入ります。 (変更可能代替 PCB を使用すると、プログラムの実行中に代替 PCB の宛先を変更できます。)GPSB はすべてのトランザクション管理用アプリケーション・プログラムで使用することができ、これを使用すると、そのアプリケーション・プログラムのための特別な PSB がなくても、指定された PCB にアクセスできるようになります。

GPSB 内の PCB には、PCB 名が事前定義されています。入出力 PCB の 名前は IOPCB です。代替 PCB の 名前は TPPCB1bb です。IMS が DBCTL 環境で GPSB のために生成する入出力作業域の最小サイズは 600 バイトです。

PCB の要約

システム・サービス要求を発行することにしている場合は、さまざまなタイプのアプリケーション・プログラムでの入出力 PCB と代替 PCB の相違点に留意する必要があります。

DB バッチ・プログラム
: PSBGEN で CMPAT=Y を指定すると、I/O PCB が PCB リストに含まれます。指定しないと、I/O PCB は含まれず、プログラムはシステム・サービス呼び出しを出すことができません。代替 PCB は、IMS によってプログラムに提供される PCB リストに常に含まれています。

BMP、**MPP**、および **IFP**
: I/O PCB と代替 PCB は、常に BMP、MPP、および IFP に渡されます。

  PCB リストには常に I/O PCB のアドレスが含まれ、それに続いて代替 PCB がある場合にはそのアドレス、さらに続いて DB PCB のアドレスが含まれています。

CICS を使用する **CICS** オンライン・プログラム
: PCB 呼び出しで IOPCB オプションを指定すると、PCB リスト内の最初の PCB アドレスは I/O PCB のアドレスになり、その後に代替 PCB のアドレスが続き、さらに DB PCB のアドレスが続きます。

  I/O PCB オプションを指定しない場合、PCB リストの最初の PCB アドレスは最初の DB PCB を指します。

次の表では、入出力 PCB と代替 PCB の情報を要約しています。

表 *39. I/O PCB* と代替 *PCB* の情報の要約：

環境	**DL/I** の呼び出し	
	PCB リスト内の **I/O PCB** アドレス	**PCB** リスト内の代替 **PCB** アドレス
MPP	あり	あり
IFP	あり	あり
BMP	あり	あり
DB バッチ[1]	なし	あり
DB バッチ[2]	あり	あり
TM バッチ[3]	あり	あり
CICS DBCTL[4]	なし	なし
CICS DBCTL[5]	あり	あり

注**:**

1. CMPAT = N を指定した場合。
2. CMPAT = Y を指定した場合。

3. CMPAT = オプション。CMPAT = N を指定した場合でも、デフォルトは常に Y です。
4. IOPCB または SYSSERVE オプションを指定しないで SCHD 要求を出した場合。
5. CICS DBCTL 要求、あるいは DBCTL を使用する CICS システムによって満たされる機能伝送要求のために、IOPCB または SYSSERVE を指定した SCHD 要求を出した場合。

AERTDLI インターフェース

AERTDLI インターフェースを使用して、ODBA アプリケーションで AIB があるデータベース呼び出しを行うことができます。

要件: AIB には 264 バイトのストレージを割り当ててください。

AERTDLI インターフェースを使用する場合、データベース呼び出しに使う AIB は、APSB 呼び出しに使う AIB と同じでなければなりません。その AIB のリソース名フィールドに、呼び出しに使いたい PCB の名前 (PSBGEN で定義された PCB 名) を指定してください。 PCB アドレスは指定しません。AIB に PCB 名が含まれているため、ユーザーのアプリケーションは PCB アドレスではなく PCB 名を参照できます。 AERTDLI 呼び出しでは、PCB へのポインターではなく、名前によって PCB を直接選択できます。呼び出しが完了すると、AIB からは、アプリケーション・プログラムから渡された PCB 名に対応する PCB アドレスが返されます。

AERTDLI 呼び出しで PCB を使用するためには、PSBGEN で名前を割り当てなければなりません。PCBNAME= か、PCB ステートメントのラベルを使用してください。 LIST=NO を指定しない限り、名前を割り当てられた PCB は、定位置ポインター・リストにも載せられます。PSBGEN の間に、DB PCB および代替 PCB の名前を定義します。すべての I/O PCB は、IOPCBbbb という PCB 名を用いて生成されます。

PCB 名を渡しますから、PCB リスト内の相対 PCB 番号は知っておく必要がありません。また、AERTDLI インターフェースでは、PCB リストに入っていない PCB についてもアプリケーション・プログラムから呼び出しを行うことができます。LIST= キーワードは、PSBGEN の間に PCB マクロで定義され、その PCB が PCB リストに含まれるかどうかを制御します。

AIB はユーザー定義ストレージに収められます。このストレージは、AERTDLI インターフェースを使用する DL/I 呼び出しのために IMS に渡されます。呼び出しが完了すると、AIB は IMS によって更新されます。 AIB のフィールドには、IMS が内部的に使用するものがいくつかあります。したがって、当該 PSB についてこれ以後出されるすべての呼び出しでは、同じ APSB AIB を使用しなければなりません。

言語環境プログラム

IBM 言語環境プログラムは、1 つ以上の高水準言語で書かれたユーザーのアプリケーション・プログラムを実行するための戦略的な実行環境を提供します。

この言語環境プログラムは、言語固有の実行時サポートだけでなく、初期設定、終了、メッセージ処理、条件処理、ストレージ管理のサポートおよび各国語サポートなどのアプリケーションに対する実行時サービスを言語間にわたって提供します。言語環境プログラムのサービスの多くは、各種プログラミング言語間で共通の一連の言語環境プログラム・インターフェースにより、明示的にアクセスできます。つまり、これらのサービスは、言語環境プログラムに準拠したすべてのプログラムからアクセスできます。

言語環境プログラムに準拠したプログラムは、次のコンパイラーを使ってコンパイルすることができます。

- IBM C++/MVS™
- IBM COBOL
- IBM PL/I

デフォルトでは、言語環境プログラム・インフラストラクチャーは 31 ビット・アドレッシング・モードを使用します。JVM=64 を指定すると、言語環境プログラムは 64 ビット・アドレッシング・モードを使用するよう変更されます。

言語環境プログラムは、C、C++、およびアセンブリー言語のインターオペラビリティーを 64 ビット・アドレッシング・モードでサポートしますが、COBOL および PL/I のインターオペラビリティーを 64 ビット・アドレッシング・モードではサポートしません。Java アプリケーションが COBOL または PL/I のいずれかを呼び出す場合は、JVM=64 に切り替えないでください。JVM=64 を使用するようリージョンを不注意に切り替えてしまった場合、互換性のない相互運用可能なアプリケーションが実行されていると、アプリケーションはシステム異常終了やユーザー異常終了を受け取る可能性があります。

IMS への CEETDLI インターフェース

IMS への言語非依存型の CEETDLI インターフェースは、言語環境プログラムによって提供されます。これは、言語環境プログラムが提供する拡張エラー処理機能をサポートする唯一の IMS インターフェースです。CEETDLI インターフェースは、他の IMS アプリケーション・インターフェースと同じ機能をサポートします。特性は次のとおりです。

- parmcount 変数は任意指定。
- 長さフィールドは 2 バイト長。
- 直接ポインターを使用。

関連資料: 言語環境プログラムの詳細については、「*z/OS* 言語環境プログラム プログラミング・ガイド」を参照してください。

PL/I の互換性のための PSBGEN の LANG= オプション

PLICALLA エントリー・ポイントを使用する互換モードで実行する IMS PL/I アプリケーションの場合には、PSBGEN に LANG=PLI を指定しなければなりません。その他のオプションでは、エントリー・ポイントを変更し、SYSTEM (IMS) をコンパイル段階の EXEC PARM に追加することができるので、PSBGEN で LANG=ブランクまたは LANG=PLI を指定することができます。以下の表では、LANG=ブランクおよび LANG=PLI が使用可能である状況を要約しています。

表 40. 言語環境プログラムにおける *PL/I* との互換性のための *LANG=* オプションの使用

コンパイル **EXEC** ステートメントが **PARM=(...,SYSTEM(IMS)...**	エントリー・ポイント名が **PLICALLA**	有効な **LANG=** 値
あり	あり	LANG=PLI
あり	なし	LANG=ブランクまたは LANG=PLI
なし	なし	注: IMS PL/I アプリケーションには無効
なし	あり	LANG=PLI

PLICALLA は、言語環境プログラムと PL/I との互換性を確保するためだけに有効なエントリー・ポイントです。バインド時に PLICALLA 入り口を使用する PL/I アプリケーションを、PLICALLA 入り口を持つ言語環境プログラムを用いてバインドすると、そのバインドは成功します。ただし、PSB で LANG=PLI を指定する必要があります。アプリケーションを、PL/I for z/OS & VM バージョン 1 リリース 1 またはそれ以降を使用して再コンパイルした後、言語環境プログラムのバージョン 1 リリース 2 またはそれ以降でバインドすると、バインドは失敗します。PLICALLA 入り口ステートメントをバインドから外さなければなりません。

IMS DB プログラミングにおける特殊な DL/I の状況

IMS DB のアプリケーション・プログラミングにおける特殊なケースとして、GUR 呼び出しの使用、HALDB に対するプログラム・スケジューリング、混合言語プログラミング、z/OS の拡張アドレッシング機能の使用、およびプリロード・プログラム用の COBOL コンパイラー・オプションの設定などがあります。

GUR 呼び出し

Get Unique Record (GUR) DL/I 呼び出しは、常に IMS カタログ・データベースにアクセスするため、特殊なケースになります。カタログが使用可能な場合、IMS はアプリケーション・プログラムに代わって、カタログ PCB に動的に接続します。アプリケーション・プログラムで GUR 呼び出しを使用することで、特定のカタログ・レコードのカタログ・データを単一の XML インスタンス文書の形式で取得することができます。その他の DL/I 読み取り呼び出しを発行して、他のデータベースの場合と同じ方法でカタログ・データベースを処理することもできます。GUR 呼び出しによって、DBD または PSB の場合にカタログ・レコード全体をリトリーブするために必要な処理ステップ数を減らすことができます。

制約事項: SSA コマンド・コードの使用は許可されません。

HALDB に対するアプリケーション・プログラムのスケジューリング

アプリケーション・プログラムは、非 HALDB に対してスケジュールされるのと同じ要領で、HALDB に対してもスケジュールされます。このスケジューリングは、HALDB マスターの使用可能状況に基づいて行われ、個々の区画のアクセスや状況には影響されません。

アプリケーション・プログラマーは、HALDB の使用不能データの取り扱いが変わったことを認識しておかなければなりません。データが使用可能かどうかを伝える PSB スケジュール時のフィードバックは、HALDB マスターが使用可能かどうかを

表すものであり、区画が使用可能かどうかを示すものではありません。しかし、DL/I 呼び出しの処理中、ある区画が初めて参照されたとき、その区画のデータが使用不能であると、それを表すエラー設定値は非 HALDB のそれと同じになります。つまり、状況コード BA か、疑似 ABENDU3303 です。

例えば、区画の半数に対して IMS /DBR コマンドを発行してオフラインにしても、残りの区画はプログラムで使用できます。

アプリケーション・プログラムが区画にアクセスすると、そのアプリケーションのインスタンスの継続時間中は、その区画はアプリケーションが使用中であると見なされます。DBDUMP、DBRECOVERY、および START コマンドが作用できる区画は、現在使用中でないものに限られます。コマンドは、BMP がアクセス中の区画は処理しません。 BMP が使用中の区画については、DFS0565I メッセージが発行されます。この規則の例外は、アクセス中の BMP が CHKP 呼び出しを発行済みで、しかも後続の DL/I 呼び出しが発行していない区画の場合です。停止中の区画に対してアプリケーションがデータ・アクセスを行おうとすると、疑似異常終了 ABENDU3303 が起こります。あるいは、BA 状況コードがアプリケーションに返されます。アプリケーションがその区画のデータに再度アクセスしようとし、それ以前に区画が STA DB コマンドで始動していると、DL/I 呼び出しは正常に処理されます。

混合言語プログラミング

アプリケーション・プログラムが言語環境プログラムの言語非依存インターフェース CEETDLI を使用する場合、IMS は呼び出し側プログラムの言語を意識する必要はありません。

アプリケーション・プログラムが言語依存インターフェースで IMS を呼び出すと、IMS は、CALL ステートメントで指定された入り口名から呼び出し側プログラムの言語を判別します。すなわち、IMS は、プログラムが次の言語で書かれているものと想定します。

- アプリケーション・プログラムが CALL ASMTDLI を使用している場合には、アセンブラー言語
- アプリケーション・プログラムが rc=CTDLI を使用している場合には、C 言語
- アプリケーション・プログラムが CALL CBLTDLI を使用している場合には、COBOL
- アプリケーション・プログラムが CALL PASTDLI を使用している場合には、Pascal
- アプリケーション・プログラムが CALL PLITDLI を使用している場合には、PL/I

例えば、PL/I プログラムがアセンブラー言語サブルーチンを呼び出し、そのアセンブラー言語サブルーチンが CALL ASMTDLI を使用して DL/I 呼び出しを行う場合、そのアセンブラー言語サブルーチンでは、PL/I 規則ではなくアセンブラー言語の呼び出し規則を使用する必要があります。

この場合、入出力域で LLZZ 形式が使用されますが、LL は PL/I の場合のようなフルワードではなく、ハーフワードになります。

z/OS の拡張アドレッシング機能

z/OS の拡張アドレッシング機能の 2 つのモードは、アドレッシング・モード (AMODE) と常駐モード (RMODE) です。 IMS では、アプリケーション・プログラムの RMODE および AMODE には制約は課されません。プログラムは、拡張仮想記憶域内に常駐させることができます。呼び出しで参照されるパラメーターも拡張仮想記憶域に入れることができます。

プリロードされたプログラムのための COBOL コンパイラー・オプション

ユーザーの COBOL プログラムを、VS COBOL II コンパイラーを使用してコンパイルし、それをプリロードする場合には、COBOL コンパイラー・オプション RES および RENT を使用する必要があります。

IMS カタログを使用したアプリケーション・プログラミング

IMS カタログ・データベースが IMS システム用に使用可能にされていれば、標準 IMS DB アプリケーション・プログラムから IMS カタログ・データベースにアクセスできます。

IMS カタログの情報

IMS カタログ・データベースでは、アプリケーションおよびデータベースのメタデータを標準 IMS DB アプリケーション・プログラムがアクセス可能な形式で保管します。この情報には、データベース定義、プログラム仕様、およびユーザー・コメントが含まれます。この情報は、どのアプリケーション・プログラムでも読み取ることができます。ただし、カタログ・データベースは書き込み保護されており、IMS Catalog Populate ユーティリティー (DFS3PU00) などの許可されたシステム・ユーティリティーでのみ更新可能です。

デフォルトでは、IMS カタログの名前は DFSCD000 です。IMS に別名接頭部が定義されていれば、*DFSC* という接頭部はその別名接頭部と置き換えられます。

IMS カタログの副次索引内の情報

IMS カタログの副次索引には、単一のセグメント・タイプ DBDPSB が含まれています。これは、IMS カタログ・データベース内の DBDXREF セグメント・タイプに論理的にリンクされます。DBDXREF セグメント・タイプは、IMS PSB のすべてのカタログ・レコードに含まれています。カタログ副次索引を使用することで、IMS カタログ全体を処理しなくても、特定のユーザー・データベースを参照するのがどの IMS プログラムであるかを判別することができます。

デフォルトでは、IMS カタログの名前は DFSCX000 です。IMS に別名接頭部が定義されていれば、*DFSC* という接頭部はその別名接頭部と置き換えられます。

アプリケーション・プログラム用の IMS カタログ PSB および PCB

IMS では、ユーザー PSB に IMS カタログ・データベースまたは副次索引用の PCB を含める必要がありません。カタログ PSB の DFSCP000、DFSCP002、および DFSCP003 は、カタログ・データベースに対する DL/I 呼び出し、または INIT

DB QUERY 呼び出しを発行するすべてのユーザー PSB に動的に付加されます。各 PSB は、以下のように異なるタイプのアプリケーション・プログラムで使用します。

DFSCP000
高水準アセンブラーおよび COBOL アプリケーション

DFSCP002
PL/I アプリケーション

DFSCP003
PASCAL アプリケーション

制約事項**:** IMS カタログ PSB は、生成済みの PSB または GSAM 専用 PSB には動的に付加されません。

以下の PCB は、別のカタログ処理モデルをサポートするために組み込まれています。

DFSCAT00
DFSCD000 (IMS カタログ) データベースの全データにアクセスする 1 次 PCB。標準カタログ処理を実行するには、この PCB を使用します。

DFSCATSX
この PCB は、カタログ PSB レコードの DBDXREF セグメント・タイプ用の SENSEG を提供するもので、PROCSEQ=DFSCX000 を使用します。カタログ副次索引によるカタログ・データベースの処理を高速化するには、この PCB を使用します。

DFSCATX0
この PCB は、カタログ副次索引レコードの DBDPSB セグメント・タイプ用の SENSEG を提供します。カタログ副次索引を直接処理するには、この PCB を使用します。

カタログ PCB はすべて常駐しています。すべてのカタログ処理が PROCOPT=GP を使用して実行されます。

IMS は、カタログ PSB を付加する際、ユーザー PSB に割り振られたスペースを自動的に増やします。96 バイトの追加スペースが、PSB CSA ストレージ・プールの各ユーザー PSB に割り振られます。カタログ PSB 自体が DLIPSB プール内で 12kb を占め、カタログ PSB を使用するユーザー PSB ごとに 500 バイトの CSAPSB プールを占有します。ご使用のストレージ・プールのサイズを最大サイズ (「各プールのカタログ PSB」×「同時にカタログにアクセスするユーザー PSB の数」) まで増やすことが必要な場合があります。

GUR 呼び出し

アプリケーション・プログラムで Get Unique Record (GUR) DL/I 呼び出しを使用することで、特定のカタログ・レコードのカタログ・データを単一の XML インスタンス文書の形式で取得することができます。その他の DL/I 読み取り呼び出しを発行して、他の IMS データベースの場合と同じ方法でカタログ・データベースを処理することもできます。GUR 呼び出しによって、DBD または PSB の場合にカタログ・レコード全体をリトリーブするために必要な処理ステップ数を減らすことができます。

制約事項**:** SSA コマンド・コードの使用は許可されません。

関連概念:

IMS カタログ・データベース内のレコードのフォーマット (データベース管理)

IMS カタログの副次索引 (データベース管理)

関連資料:

287 ページの『IMS DB プログラミングにおける特殊な DL/I の状況』

GUR 呼び出し (アプリケーション・プログラミング API)

第 13 章 データベースのバージョン管理およびアプリケーション・プログラミング

IMS システムでデータベースのバージョン管理が有効になっている場合、IMS は 1 つのデータベースについて複数のバージョンの構造定義を維持できます。これにより、新しいアプリケーション・プログラムをサポートするためにデータベースが変更された後でも、既存のアプリケーション・プログラムが引き続きデータベースにアクセスできます。

新規バージョンのデータベースを定義する場合、データベース管理者は、新規データベース定義のバージョン番号を指定します。その後、バージョン番号は、そのバージョンのデータベースへのアクセスを要求するために使用されます。

複数のバージョンのデータベースが使用可能な場合、アプリケーション・プログラムで特定のデータベース・バージョンが指定されなければ、IMS は、デフォルトで現行バージョンのデータベースへのアクセスを提供します。現行バージョンのデータベースは、最も大きいバージョン番号を持ち、データベースに対する最新の変更が含まれるデータベースです。この IMS システムのデフォルトは、IMS が現行バージョンでではなくバージョン 0 のデータベースへのアクセスを提供するように変更することができます。

IMS システムのデフォルトは、PSB 生成時に PSBGEN ステートメントで DBLEVEL パラメーターを指定することで、プログラム仕様ブロック (PSB) レベルでオーバーライドすることもできます。

アプリケーション・プログラムで特定のデータベース・バージョンが必要な場合は、PCB を定義する際の PCB ステートメントの DBVER パラメーターで、または DL/I INIT VERSION 呼び出しの発行による実行時に、そのバージョン番号を明示的に指定することができます。

要求されたバージョンのデータベース定義が見つからない場合、あるいはバージョンの要求時にデータベースのバージョン管理が有効にされていない場合は、IMS は異常終了 3303 でプログラムを強制終了し、メッセージ DFS3303I を発行します。このメッセージには、異常終了の原因に関する詳細が含まれます。オプションで、アプリケーション・プログラムは、INIT STATUS GROUPA 呼び出しを発行して、異常終了 3303 の代わりに BA 状況コードを受け取ることができます。

重要: 新規バージョンのデータベースが作成されたら、アプリケーション・プログラムが新規バージョンのデータベースを更新する前に、以前のバージョンのデータベースにまだアクセスできることを確認してください。

データベースのバージョン管理では、データベース定義に対する特定の変更のみがサポートされます。データベースに対してサポートされない変更が行われると、アプリケーション・プログラムは、旧バージョンのデータベースにアクセスできなくなります。現行バージョンのデータベースのみがアクセス可能になります。

ほとんどのデータベース・タイプでは、旧バージョンのデータベースを使用するアプリケーション・プログラムがスケジュールされない限り、サポートされない変更は検出されません。ただし、HALDB データベースに対する構造変更を適用するのに HALDB 変更関数を使用している場合は、IMS は、変更処理中にサポートされないデータベースの変更を検出します。

新規バージョンのデータベースにサポートされない変更が含まれている場合、すべてのアプリケーション・プログラムを新規バージョンのデータベース構造を使用するように更新するか、サポートされない構造変更を除去するようにデータベース定義を変更する必要があります。

バッチ・アプリケーション・プログラムおよびデータベースのバージョン管理

IMS.PROCLIB データ・セットの DFSDF*xxx* メンバーで DBVERSION=Y を指定することで、DLIBATCH または DBBBATCH 領域で実行されているオフライン DL/I バッチ・アプリケーション・プログラムのデータベース・バージョン管理を有効にすることができます。

DLIBATCH または DBBBATCH アプリケーション・プログラムは、それぞれの JCL の EXEC ステートメントで DFSDF=*xxx* パラメーターを指定することで、DFSDF*xxx* メンバーを参照します。以下に例を示します。

```
//STEP1    EXEC PGM=DFSRRC00,REGION=0M,
// PARM=(DLI,DFSDDLT0,PSBCJK03,,01,,,,,,BCH1,,Y,Y,,,,,,,,,,,,,,,,,,,,
//       ,,,,,'DFSDF=C35')
```

重要: DLIBATCH アプリケーション・プログラムは、ACB ライブラリーではなく、PSB および DBD ライブラリーを使用します。データベースのバージョン管理を使用する場合、DLIBATCH アプリケーション・プログラムは、現行の物理データベース構造に一致する DBD メンバーを含む DBD ライブラリーを使用する必要があります。

関連概念:

- データベースのバージョン管理 (データベース管理)

関連タスク:

- オンライン HALDB データベースの定義の変更 (データベース管理)

関連資料:

- INIT 呼び出し (アプリケーション・プログラミング API)
- PSBGEN ステートメント (システム・ユーティリティー)

第 14 章 COBOL または PL/I からの DL/I インターフェースの確立

COBOL または PL/I から DL/I インターフェースを確立するには、CBLTDLI プロシージャーまたは PLITDLI プロシージャーを使用します。

CBLTDLI

次に示す制御ステートメントは、COBOL から DL/I へのインターフェースを確立するのに必要です。 次のメンバーのブロック・サイズは、3200 以下でなければなりません。

```
LIBRARY SDFSRESL(CBLTDLI)    DL/I LANGUAGE INTERFACE
LIBRARY SDFSRESL(DFHEI01)    HLPI LANGUAGE INTERFACE
LIBRARY SDFSRESL(DFHEI1)     HLPI LANGUAGE INTERFACE
```

PLITDLI

次に示す制御ステートメントは、PL/I から DL/I へのインターフェースを確立するのに必要です。 次のメンバーのブロック・サイズは、3200 以下でなければなりません。

```
LIBRARY SDFSRESL(PLITDLI)    DL/I LANGUAGE INTERFACE
LIBRARY SDFSRESL(DFHEI01)    HLPI LANGUAGE INTERFACE
LIBRARY SDFSRESL(DFHEI1)     HLPI LANGUAGE INTERFACE
ENTRY PLICALLA
```

PLITDLI は、PL/I 最適化コンパイラーの使用時に有効です。

第 15 章 各呼び出し後のデータベース内の現在位置

位置付けにより、ユーザーが出した各呼び出しの後で、データベース内におけるユーザーの位置が DL/I によって常に記録されます。DL/I がデータベース内のユーザーの位置を記録することにより、ユーザーはデータベースを順次に処理できます。

呼び出しが成功した後の現在位置

GN、GNP、GHN、および GHNP 呼び出しを出してデータベースを順次に処理する際には、位置が重要になります。

現在位置は、ユーザーが呼び出しで指定したセグメントの検索を IMS が開始する位置です。

このセクションでは、成功した呼び出しの場合の現在位置について説明します。現在位置は、失敗したリトリーブ呼び出しまたは ISRT 呼び出しによっても影響を受けます。

データベースに対する最初の呼び出しを出す以前の現在位置は、データベースの最初のルート・セグメント・オカレンスの直前の場所になります。すなわち、非修飾 GN 呼び出しを出すと、IMS は最初のルート・セグメント・オカレンスをリトリーブします。現在位置は、DB PCB で定義されている階層における、ユーザーが参照したセグメント・オカレンスの 次のセグメント・オカレンスになります。

呼び出しの中には、データベース内のユーザー位置を取り消すものがあります。この位置は、GU 呼び出しで再設定できます。CHKP および SYNC (コミット・ポイント) 呼び出しは位置を取り消すため、これらのいずれかの呼び出しを出した後には、GU 呼び出しを出してください。ROLS 呼び出しと ROLB 呼び出しもデータベース内のユーザー位置を取り消します。

GU 呼び出しを出す場合、GU 呼び出しまたは使用する SSA のコーディング方法がデータベース内のユーザーの現在位置による影響を受けることはありません。プログラム実行中の異なる時点で (異なる位置が設定されているときに) 同じ GU 呼び出しを出した場合、この呼び出しを出すたびに同じ結果を受け取ります。この呼び出しが正しくコーディングされていると、ユーザーが要求したセグメントが現在位置の前にあっても後にあっても、IMS によってそのセグメント・オカレンスが戻されます。

例外: GU 呼び出しでその呼び出し内の各レベルに SSA の指定がない場合、IMS は、プログラム中の異なるポイントで異なるセグメントを戻す可能性があります。この場合のセグメントは、各レベルにおける位置に基づいて戻されます。

例えば、以下の図に示すデータ構造に対して、次の呼び出しを発行するとします。

```
GU   Abbbbbbb(AKEYbbbbbA1)
     Bbbbbbbb(BKEYbbbb=bB11)
     Dbbbbbbb(DKEYbbbbbD111)
```

この図の構造には、A、B、C、D、E、F の 6 つのセグメント・タイプが含まれています。図 49 は 1 つのデータベース・レコードを示しており、このレコードのルートは A1 です。

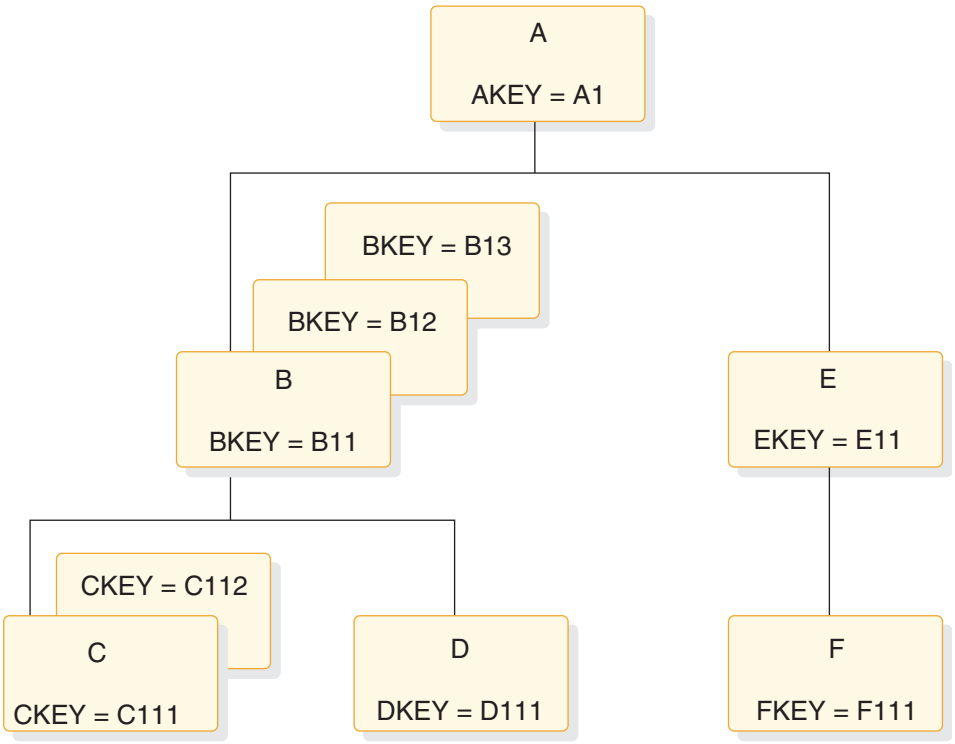


図 *49.* 現在位置の階層

この呼び出しを出すと、呼び出しを出した時点のユーザー位置がどこであっても、IMS はキー D111 をもつ D セグメントを戻します。この呼び出しがユーザー・プログラムで出される最初の呼び出しである場合 (そして、これがデータベース内の最初のデータベース・レコードである場合) には、この呼び出しを出す前における現在位置は、データベース内の最初のセグメント・オカレンスの直前、すなわち、キー A1 をもつセグメント A の直前になります。ユーザーがこの呼び出しを出すときの現在位置がセグメント D111 を過ぎている場合であっても (例えば、セグメント F111 の直前である場合)、IMS は、ユーザーのプログラムにセグメント D111 を戻します。このことは、現在位置が異なるデータベース・レコード内にある場合にもあてはまります。

GN 呼び出しおよび GNP 呼び出しを出す場合には、データベース内の現在位置によって、呼び出しおよび SSA のコーディング方法が影響を受けます。これは、GN または GNP 呼び出しで記述されたセグメントを検索する場合に、IMS が現在位置から検索を開始し、しかもデータベース内で前方方向しか行うことができないためです。IMS は、GN または GNP の条件を満たすためにセグメント・オカレンスを逆方向に検索することはできません。 F コマンド・コードが使用されないと、これらの呼び出しを満たそうとする場合に、データベース内を前方に移動することしかできません。 F コマンド・コードの使用は、 *IMS V15* アプリケーション・プログラミング *API*で説明されています。

既に通過しているセグメント・オカレンスを対象として GN 呼び出しを出すと、IMS は、現在位置から検索を開始し、データベースの終わりに達した時に検索を終了するか (この場合 GB 状況コードが戻される)、または要求したセグメントが検出できなかったことを SSA から判断された時に検索を終了します (この場合 GE 状況コードが戻される)。

挿入するセグメント・オカレンスの親に対して修飾 SSA を指定しなかった場合には、現在位置によって ISRT 呼び出しが影響を受けます。セグメント・オカレンスに対して非修飾 SSA だけを指定する場合には、データベースにおけるユーザー位置が、そのセグメント・オカレンスを挿入したい場所になるようにしなければなりません。

関連概念:

A コマンド・コード (アプリケーション・プログラミング API)

G コマンド・コード (アプリケーション・プログラミング API)

303 ページの『呼び出しが失敗した後の現在位置』

リトリーブ呼び出し後の位置

何らかの種類のリトリーブ呼び出しを発行した後における位置は、リトリーブしたセグメント・オカレンスの直後 — あるいは、D コマンド・コードを使用して複数のセグメント・オカレンスをリトリーブした場合には、パス内の最低レベルのセグメント・オカレンスの直後です。リトリーブ呼び出しで D コマンド・コードを使用した場合に成功した呼び出しは、IMS で完全に満たされた呼び出しです。

例えば、上図に示すデータベースに対して、次の呼び出しを発行すると、IMS はキー C111 を持つ C セグメント・オカレンスを戻します。現在位置は、C111 のすぐ後になります。ここで非修飾の GN 呼び出しを出すと、IMS はセグメント C112 をユーザーのプログラムに戻します。

```
GU   Abbbbbbb(AKEYbbbbEQA1)
     Bbbbbbbb(BKEYbbbbEQB11)
     Cbbbbbbb(CKEYbbbbEQC111)
```

ユーザーの現在位置は、GU、GN、GNP、または何らかの Get Hold のうちのどの呼び出しを使用してリトリーブを行った場合でも、セグメント C111 のリトリーブ後と変わりません。

D コマンド・コードを使用した Get 呼び出しを出して複数のセグメント・オカレンスをリトリーブした場合の現在位置は、リトリーブされた最低位セグメント・オカレンスの直後になります。上記の例の GU 呼び出しを出す際に、セグメント A および B に関する SSA に D コマンド・コードを組み込んだ場合にも、現在位置はセグメント C111 の直後になります。C111 は、この呼び出しに応じて IMS がリトリーブする最後のセグメントです。D コマンド・コードを使用した場合、この呼び出しは次のようになります。

```
GU   Abbbbbbb(AKEYbbbbEQA1)
     Bbbbbbbb(BKEYbbbbEQB11)
     Cbbbbbbb*D(CKEYbbbbEQC111)
```

C セグメントの SSA には D コマンド・コードを指定する必要がありません。これは、IMS は常に、最後の SSA で記述されたセグメント・オカレンスを入出力域に戻すためです。

DLET の後の位置

DLET 呼び出しが行えた後の現在位置は、削除されたセグメント・オカレンスの直後になります。これは、従属セグメントをもつセグメント・オカレンスを削除した場合にも、従属セグメントのないセグメント・オカレンスを削除した場合にもあてはまります。

例えば、以下のサンプル・コードで示されている呼び出しを発行してセグメント C111 を削除する場合、現在位置はセグメント C111 の直後になります。その後で非修飾の GN 呼び出しを出すと、IMS はセグメント C112 を戻します。

```
GHU   Abbbbbbb(AKEYbbbb=bA1)
      Bbbbbbbb(BKEYbbbb=bB11)
      Cbbbbbbb(CKEYbbbb=bC111)
DLET
```

次の図は、この呼び出しの後における階層の様子を示しています。 DLET 呼び出しによりセグメント C111 が正しく削除されています。

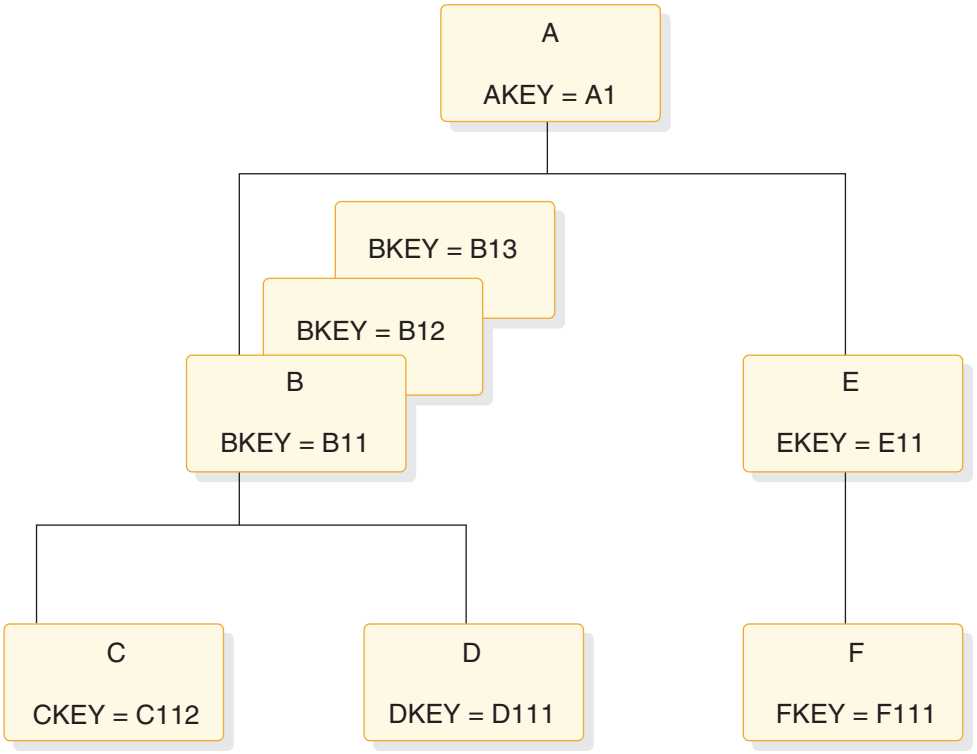


図 *50.* セグメント削除後の階層

従属セグメントをもつセグメント・オカレンスに出した DLET 呼び出しが成功すると、IMS は、そのセグメント・オカレンスとともに従属セグメントも削除します。この場合にも、現在位置は、削除されたセグメント・オカレンスの直後になります。非修飾 GN 呼び出しを出すと、削除されたセグメントの後のセグメント・オカレンスが戻されます。

例えば、上図に示した階層でセグメント B11 を削除すると、IMS はその従属セグメントである C112 および D111 も削除します。この後の現在位置は、セグメント B11 の直後、すなわちセグメント B12 の直前になります。ここで非修飾の GN 呼び出しを発行すると、IMS はセグメント B12 を戻します。次の図は、この呼び出しを発行した後の階層の様子を示しています。

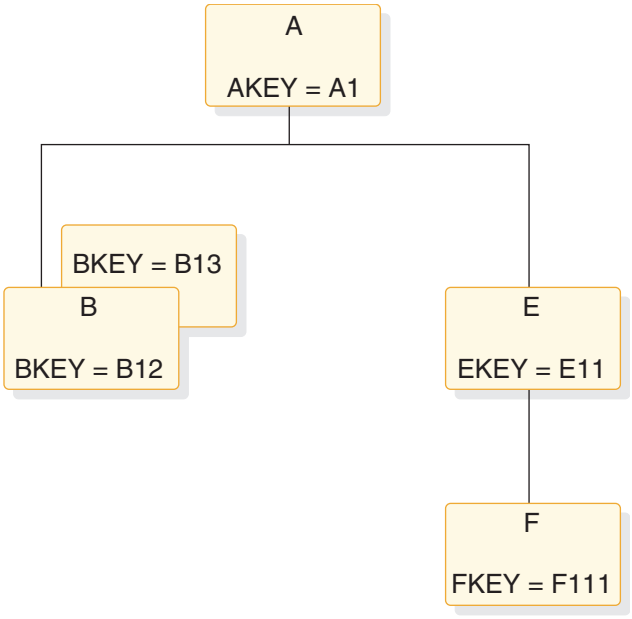


図 *51.* セグメントおよび従属セグメントの削除後の階層

IMS はセグメントの従属セグメントを削除するため、現在位置は、最後の (最低位の右端) 従属セグメントの直後にあると考えることができます。最初の図の例では、これはセグメント D111 の直後になります。ただし、この後で非修飾 GN 呼び出しを出すと、この場合にも IMS はセグメント B12 を戻します。現在位置はどちらの位置にあると考えることもでき、いずれの場合でも結果は同じになります。ただし、GU パス CALL の後で DLET を出した場合は、例外で、GE 状況コードが戻されます。

関連概念:

303 ページの『呼び出しが失敗した後の現在位置』

REPL の後の位置

REPL 呼び出しが成功した場合には、データベースにおけるユーザーの位置は変わりません。現在位置は、REPL 呼び出しを出す前と同じです。

これは、REPL 呼び出しを出す前に Get Hold 呼び出しでリトリーブされた最低位セグメントの直後になります。

例えば、GU 呼び出しの代わりに GHU 呼び出しを使用して、上図のセグメント B13 をリトリーブし、入出力域内のそのセグメントを変更し、その後 REPL 呼び出しを発行すると、現在位置はセグメント B13 の直後になります。

ISRT の後の位置

データベースに新しいセグメント・オカレンスを追加すると、現在位置は新しいセグメント・オカレンスの直後になります。

例えば、以下の図で、次の呼び出しを発行してセグメント C113 をデータベースに追加すると、現在位置はセグメント C113 の直後になります。非修飾 呼び出しを出すと、セグメント D111 がリトリーブされます。

```
ISRT   Abbbbbbb(AKEYbbbb=bA1)
       Bbbbbbbb(BKEYbbbb=bB11)
       Cbbbbbbb
```

ユニーク・キーをもつセグメントを挿入すると、IMS は新しいセグメントをキー・シーケンスで配置します。非ユニーク・キーを持つセグメントまたはキーをまったく持たないセグメントを挿入すると、IMS は、そのデータベースの DBD の SEGM ステートメントの規則パラメーターに従ってセグメントを配置します。「*IMS V15* アプリケーション・プログラミング *API*」の『ISRT 呼び出し』で、これらの規則が説明されています。

D コマンド・コードを使用して複数のセグメント・オカレンスを挿入すると、現在位置は挿入された最低位セグメント・オカレンスの直後になります。

例えば、新しいセグメント B (これは B14 になります)、および B14 の従属セグメントである新しい C セグメント・オカレンス (C141) を挿入するものとします。以下の図は、これらのセグメント・オカレンスの挿入後の階層の様子を示しています。これらを行うための呼び出しは、次のようになります。

```
ISRT   Abbbbbbb(AKEYbbbb=bA1)
       Bbbbbbbb
*D
       Cbbbbbbbb
```

C セグメントに関しては、SSA に D コマンド・コードを指定する必要はありません。ISRT 呼び出しの場合、D コマンド・コードを含む必要があるのは、挿入する最初のセグメントの SSA だけです。この呼び出しを出した後の位置は、キー C141 をもつ C セグメント・オカレンスの直後になります。その後で非修飾の GN 呼び出しを出すと、IMS はセグメント E11 を戻します。

ISRT 呼び出しの結果としてユーザーのプログラムが II 状況コードを受け取った場合 (これは、挿入しようとしたセグメントが既にデータベース内に存在していることを意味します)、現在位置は、挿入しようとしたセグメントと重複していたセグメントのすぐ 前 になります。

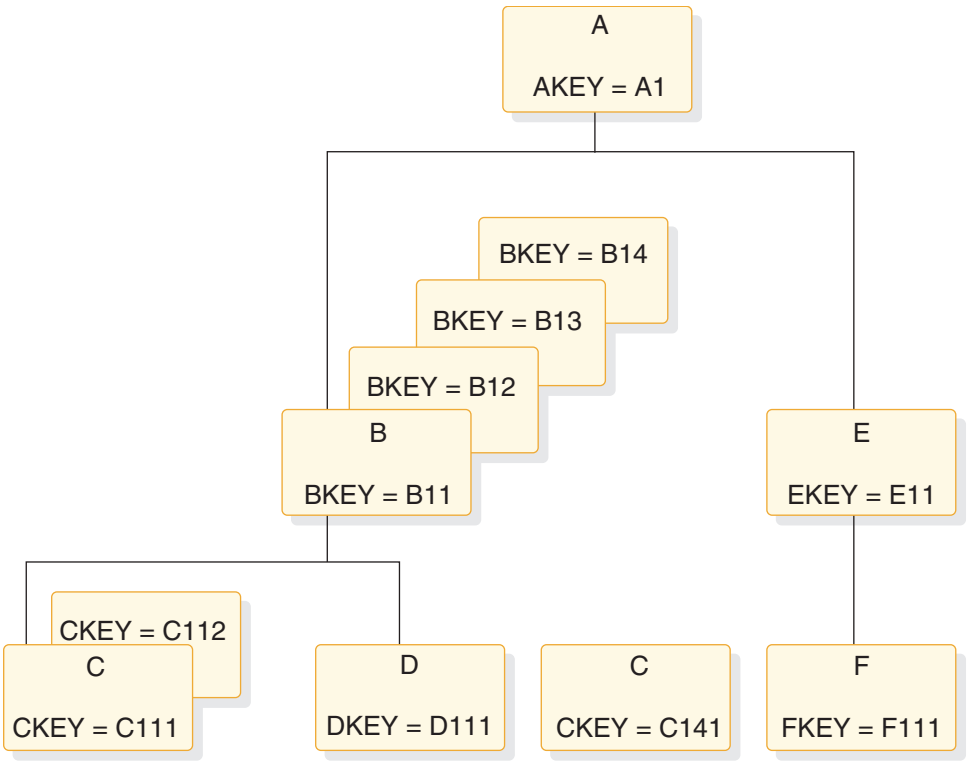


図 52. 新しいセグメントおよび従属セグメントを追加した後の階層

呼び出しが失敗した後の現在位置

これとは別に、リトリーブ呼び出しおよび ISRT 呼び出しを出したときに、IMS の設定する位置があります。この位置は、リトリーブまたは挿入されるセグメントへのパス内にある各階層レベルごとに、1 つのセグメント・オカレンスに設定されます。プログラムが発行するすべての DL/I 呼び出しが完全に成功するとは限りません。したがって、呼び出しが失敗した後における、データベース内でのユーザー位置を判別する方法を理解しておく必要があります。

「*IMS V15 アプリケーション・プログラミング API*」の U および V コマンド・コードの説明を理解するには、IMS がこの位置をどのように設定するかが分かっている必要があります。また、IMS がリトリーブ呼び出しまたは ISRT 呼び出しに対して未検出状況コードを戻したときの、データベース内でのユーザー位置についても、理解しておく必要があります。

失敗した DLET または REPL 呼び出しの後の位置

DLET または REPL 呼び出しが失敗した場合には、現在位置は影響を受けません。データベース内でのユーザーの位置は、その呼び出しを出す前の位置と同じです。しかし、Get 呼び出しまたは ISRT 呼び出しが失敗した場合には、ユーザーの現在位置が影響を受けます。

ユーザーが要求したセグメントを IMS が検出できなかった場合の、データベース内でのユーザーの位置を理解するためには、セグメントを検出できないことをどのようにして DL/I が判別するかを理解する必要があります。

リトリーブまたは挿入される最低位セグメントの後に現在位置を設定する他に、IMS は、リトリーブまたは挿入するセグメントまでのパス内で、階層レベルごとに 1 つのセグメント・オカレンスに別のタイプの位置を保持します。

例えば、次の図では、以下の SSA を指定して発行した GU 呼び出しが成功した場合、IMS は、各階層レベルで 1 つずつ位置を設定しています。

```
GU   Abbbbbbb(AKEYbbbb=bA1)
     Bbbbbbbb(BKEYbbbbbbB11)
     Cbbbbbbb(CKEYbbbb=bC111)
```

これにより、呼び出しの各レベルごとに 1 つずつ、合計 3 つの DL/I 位置が存在することになります。

- キー A1 の A セグメントに 1 つ
- キー B11 の B セグメントに 1 つ
- キー C111 の C セグメントに 1 つ

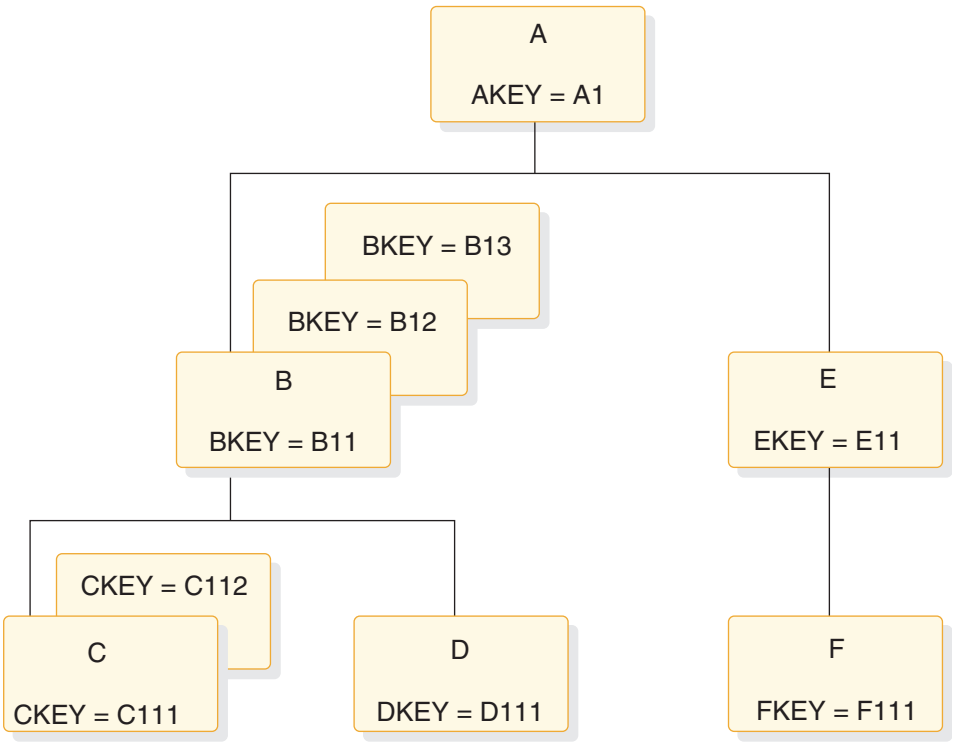


図 *53. DL/I* の位置

セグメント・オカレンスを検索する場合、IMS は、呼び出しの条件を満たすセグメント・オカレンスのうちで最初に検出されたものを受け入れます。そして IMS は、そのセグメント・オカレンスのキーをキー・フィードバック域に入れます。そして IMS は、そのセグメント・オカレンスのキーをキー・フィードバック域に入れます。

失敗したリトリーブまたは ISRT 呼び出しの後の位置

リトリーブ呼び出しまたは ISRT 呼び出しが GE 状況コードを受け取った後の現在位置は、呼び出しで指定された SSA の条件を満たすために IMS がデータベース内のどこまでリトリーブしたのかによって異なります。IMS は、ISRT 呼び出しを処理するために、ユーザーが挿入するセグメント・オカレンスの親をすべてチェックします。ISRT 呼び出しは、この点でリトリーブ呼び出しと類似しています。すなわち、IMS は呼び出しの各レベルの条件を満たすために、レベルごとにその呼び出しを処理し、セグメント・オカレンスの検出を試みます。リトリーブ呼び出しで IMS から GE 状況コードが戻された場合、この状況コードは、IMS が呼び出し内のレベルの 1 つを満たすセグメント・オカレンスを検出できなかったことを意味します。ISRT 呼び出しで IMS から GE 状況コードが戻された場合、この状況コードは、挿入されるセグメント・オカレンスの親のうちの 1 つを IMS が検出できなかったことを、意味します。これらは、未検出呼び出しと呼ばれます。

IMS は、リトリーブ呼び出しおよび ISRT 呼び出しを処理する際には、条件を満たすことができないことを判別するまで、呼び出しの条件を満たそうとします。ユーザーが SSA で指定した記述に一致するセグメントを IMS が最初に検出しようとしたときに、最初の親のもとに該当セグメントがない場合、IMS は、別の親のもとで該当セグメントの検索を試みます。IMS が前方に移動して別の親のもとで再び検索することができるかどうかは、呼び出しで SSA がどのようにコーディングされているのかによって異なります。

例えば、前掲の図に示す階層において、次の GN 呼び出しを発行して、キー C113 を持つ C セグメントをリトリーブするとします。

```
Abbbbbbb(AKEYbbbb=bA1)
Bbbbbbbb(BKEYbbbb=bB11)
Cbbbbbbb(CKEYbbbb=bC113)
```

この呼び出しを処理するときに、IMS は、キーが C113 の C セグメントを検索します。IMS は、A セグメントと B セグメントの修飾に一致する親をもつ C セグメントだけを探すことができます。このパスの一部である B セグメントは、キー B11 をもつものでなければならず、パスの一部である A セグメントは、キー A1 を持つものでなければなりません。そのうえで、IMS は最初の C セグメントを探します。その場合のキーは C111 です。その次の C セグメントのキーは C112 です。IMS は、B11 セグメント・オカレンスのもとで 3 番目の C セグメント・オカレンスを探します。B11 のもとには、それ以上は C セグメント・オカレンスがありません。

SSA で、C のパス内の A および B セグメント・オカレンスは特定の値に等しくなければならないことが指定されているため、IMS は、別の A または B セグメント・オカレンスのもとでキー C113 の C セグメント・オカレンスを探すことができません。親 B11 のもとにはそれ以上は C セグメント・オカレンスがありません。すなわち、C の親は B11 でなければならず、B11 の親は A1 でなければなりません。IMS は、ユーザーが指定したセグメントが存在しないことを判別し、未検出 (GE) 状況コードを戻します。

この呼び出しで GE 状況コードが戻された場合、キー・フィードバック域から位置を判別できます。キー・フィードバック域は、IMS が呼び出しの条件を満たすことができたレベルでの位置 (この場合には A1 および B11) を表しています。

この呼び出し後の現在位置は、ユーザーの呼び出しの条件を満たすために IMS が検査した最後のセグメント・オカレンス — この場合には C112 の直後になります。その後で非修飾の GN 呼び出しを出すと、IMS は D111 を戻します。

A および B が非ユニーク・キーを持つ場合には、この呼び出しの後の現在位置はこれとは異なります。 A のキーがユニークであって、B のキーが非ユニークであるとします。 IMS が B11 のもとで C113 セグメントを検索し、そのセグメントを検出できなかった場合、IMS は B11 から前方に移動し、キー B11 をもつ別の B セグメントを探します。IMS がこのセグメントを検出できなかった場合、DL/I は GE 状況コードを戻します。現在位置は、両方のキーがユニークであった場合よりも、データベース内をさらに前方に移動した位置になります。現在位置は、セグメント B11 の直後です。非修飾 GN 呼び出しを出すと、B12 が戻されます。

A と B の両方が非ユニーク・キーを持つ場合、上記の呼び出し後の現在位置は、セグメント A1 の直後になります。これ以上セグメント A1 がない場合、非修飾 GN 呼び出しを出すとセグメント A2 が戻されます。別の A1 がある場合、IMS は別の A1 のもとでセグメント C113 を検出しようとします。

しかし、セグメント B に関する SSA で「以上」関係演算子を指定して同じ呼び出しを出した場合には、これと異なります。

```
GU   Abbbbbbb(AKEYbbbb=>bA1)
     Bbbbbbbb(BKEYbbbb=>B11)
     Cbbbbbbb(CKEYbbbb=>bC113)
```

IMS は、セグメント A1 とセグメント B11 に位置を設定します。A1 と B11 は呼び出し内の最初の 2 つの SSA の条件を満たすため、IMS はそれらのキーをキー・フィードバック域に入れます。IMS はセグメント B11 のもとでセグメント C113 を検索します。ここには、セグメント C113 はありません。しかし、この場合、親 B のキーは B11 以上である可能性があるため、IMS が検索を続けることがあります。次のセグメントは B12 です。 B12 はセグメント B の修飾の条件を満たすため、IMS はキー B12 をキー・フィードバック域に入れます。その後で、IMS は B12 のもとで C113 を探しますが、これは検出されません。B13 についても同じことが起こります。IMS は、キー B13 をキー・フィードバック域に入れ、B13 のもとで C113 を探します。

IMS は、A1 のもとにそれ以上 B セグメントがないことが分かると、さらにデータベース内を移動して、別の親 A のもとで呼び出しの条件を満たす B および C セグメントを探そうとします。しかし、今回はそれが行えません。A セグメントに関する SSA で、A セグメントが A1 でなければならないことが指定されているためです。(これらのキーが非ユニークであれば、IMS は別の A1 セグメントを探すことができました。) これにより、IMS は、ユーザーが指定した親のもとで C113 を検出できないことを判別し、ユーザーのプログラムに GE 状況コードを戻します。

この例では、「以上」演算子が使用されているため、セグメント C113 を探すための検索対象は 1 つの B セグメントだけには限定されていません。位置は、予想よりも先に進んでいる可能性がありますが、キー・フィードバック域によって位置を知ることができます。キー・フィードバック域内の最後のキーは、セグメント B13 のキーです。 IMS の現在位置は、セグメント B13 の直後です。ここで非修飾の GN 呼び出しを出すと、IMS はセグメント E11 を戻します。

この呼び出しに関して IMS が検査するそれぞれの B セグメントは、B セグメント用の SSA の条件を満たすため、IMS は各セグメントのキーをキー・フィードバック域に入れます。しかし、IMS が調べたうちの 1 つ以上のセグメントが呼び出しの条件を満たせなかった場合には、IMS は、そのセグメントのキーをキー・フィードバック域に入れません。このことは、データベース内の位置が、キー・フィードバック域によって反映されるものよりも、前方に進んでいる可能性があることを意味します。例えば、キー・フィールドの他に、データ・フィールドでセグメント B を修飾して、同じ呼び出しを発行するとします。そのためには、セグメント B に関する複数の修飾ステートメントを使用します。

呼び出しを修飾するためのデータ・フィールドが BDATA という名前であるものとします。また、ユーザーが希望する値が 14 であって、BDATA の値が 14 になっているセグメントが 1 つだけ (B11) しかないものとします。

```
GN   Abbbbbbb(AKEYbbbb=bA1)
     Bbbbbbbb(BKEYbbbb>=B11*BDATAbbb=b14)
     Cbbbbbbb(CKEYbbbb=bC113)
```

この呼び出しを出した後のキー・フィードバック域には、セグメント B11 のキーが入ります。GE 状況コードを受け取るまでこの呼び出しの発行を続けると、現在位置はセグメント B13 の直後になりますが、キー・フィードバック域にあるのは引き続きセグメント B11 のキーだけです。 IMS が調べる B セグメントのうちで、呼び出しで指定された SSA の条件を満たすのは、1 つだけ (B11) です。

「より大きい」または「以上」関係演算子を使用すると、検索範囲は制限されません。この種の呼び出しで GE 状況コードを受け取り、IMS が検査したセグメントの中で SSA に適合しないものが 1 つでもある場合、データベース内の位置は、キー・フィードバック域で示されている位置よりもさらに先である可能性があります。次の GN または GNP の呼び出しを発行する際に、IMS の検索開始位置を「実際の」位置ではなくキー・フィードバック域に示された位置にする場合、以下のいずれかを行うことができます。

- 希望する場所に位置を再設定するために、完全修飾された GU 呼び出しを出す。
- U コマンド・コードを使用した GN または GNP 呼び出しを出す。 SSA に U コマンド・コードを指定すると、IMS に対して、そのレベルで設定した最初の位置を呼び出しの修飾として使用するように指示が出されます。これは、IMS がそのレベルで位置付けたセグメント・オカレンスに関して「等しい」関係演算子を指定した場合に、似た効果をもちます。

例えば、まず、セグメント B の SSA で「以上」関係演算子を指定して GU 呼び出しを発行し、次いで以下の GN 呼び出しを発行するものとします。

```
GN   Abbbbbbb*U
     Bbbbbbbb*U
     Cbbbbbbbb
```

この U コマンド・コードは、IMS に対して、セグメント A1 を親 A として使用し、セグメント B11 を親 B として使用するように指示します。IMS はセグメント C111 を戻します。しかし、U コマンド・コードを指定しないで同じ呼び出しを出した場合には、IMS はセグメント B13 から検索を開始し、B セグメントが検出されるまで次のデータベース・レコードへの前方に移動します。IMS は、検出した最初の B セグメントを戻します。

関連概念:

300 ページの『DLET の後の位置』

複数処理

アプリケーション・プログラムが階層の中でセグメントにアクセスする順序は、そのアプリケーション・プログラムの目的によって異なります。セグメントに対して直接アクセスを行うプログラムもあれば、順次アクセスを行うプログラムもあります。また、アプリケーション・プログラムによっては、プログラムが別々の階層パスまたは別々のデータベース・レコードにあるセグメントを並行して処理しなければならないこともあります。

プログラムが別々の階層パスまたは別々のデータベース・レコードのセグメントを並行して処理しなければならない場合、複数位置付けまたは複数 PCB を使用すると、プログラムの処理を単純にできます。以下に例を示します。

- プログラムが別々の階層パスから交互にセグメントをリトリーブしなければならないものとします。例えば、次の図で、B11、C11、B12、C12 という順序でリトリーブします。プログラムで複数位置付けを使用すると、IMS は両方の階層パスで位置を保持します。これにより、プログラムは異なるパスからセグメントをリトリーブするたびに GU 呼び出しを出して位置を設定し直す必要がなくなります。
- プログラムが別々のデータベース・レコードから交互にセグメントをリトリーブしなければならないものとします。例えば、A1 の下で B セグメントをリトリーブし、次に別の A ルート・セグメントの下で B セグメントをリトリーブします。プログラムで複数 **PCB** を使用すると、IMS は両方のデータベース・レコードで位置を保持します。これにより、プログラムは異なるデータベース・レコードにアクセスするたびに GU 呼び出しを出して位置を設定し直す必要がなくなります。

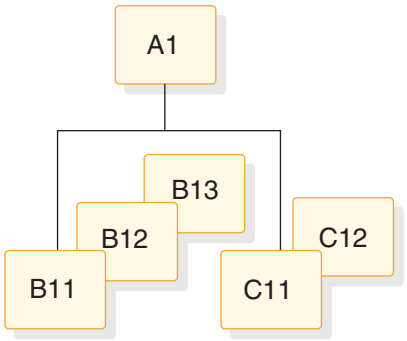


図 *54.* 複数処理

複数位置付け

アプリケーション・プログラムの PSB を定義するときには、使用したい位置付けの種類として、単一位置付けまたは複数位置付けを選択できます。これまでに使用してきたすべての例、および現在位置に関する説明では、単一位置付けが使用されていました。

PSB を定義するときには、各 PCB で使用したい位置の種類を PCB ステートメントで指定してください。 DEDB の場合には、POS オペランドは無視されます。DEDB では複数位置付けのみサポートされます。

単一位置付け
: IMS が、その PCB によって定義された階層の 1 つの階層パスで位置を保持することを意味します。セグメントをリトリーブするときには、IMS はすべての従属セグメントに関する位置、および同じレベルのすべてのセグメントに関する位置を消去します。

複数位置付け
: IMS が、アクセスされるデータベース・レコード内でそれぞれの階層パスに位置を保持することを意味します。セグメントをリトリーブするときには、IMS はすべての従属セグメントに関する位置を消去しますが、同じレベルのセグメントに関しては位置を保持します。同じ親のもとにある別々のセグメント・タイプを並行して処理できます。

例えば、以下の図に示される階層を使用して次の 2 つの呼び出しを発行するものとします。

```
GU    Abbbbbbb(AKEYbbbb=bA1)
      Bbbbbbbb(BKEYYbbbb=bB11)
      Cbbbbbbb(CKEYYbbbb=bC111)
GN    Ebbbbbbb(EKEYYbbbb=bE11)
```

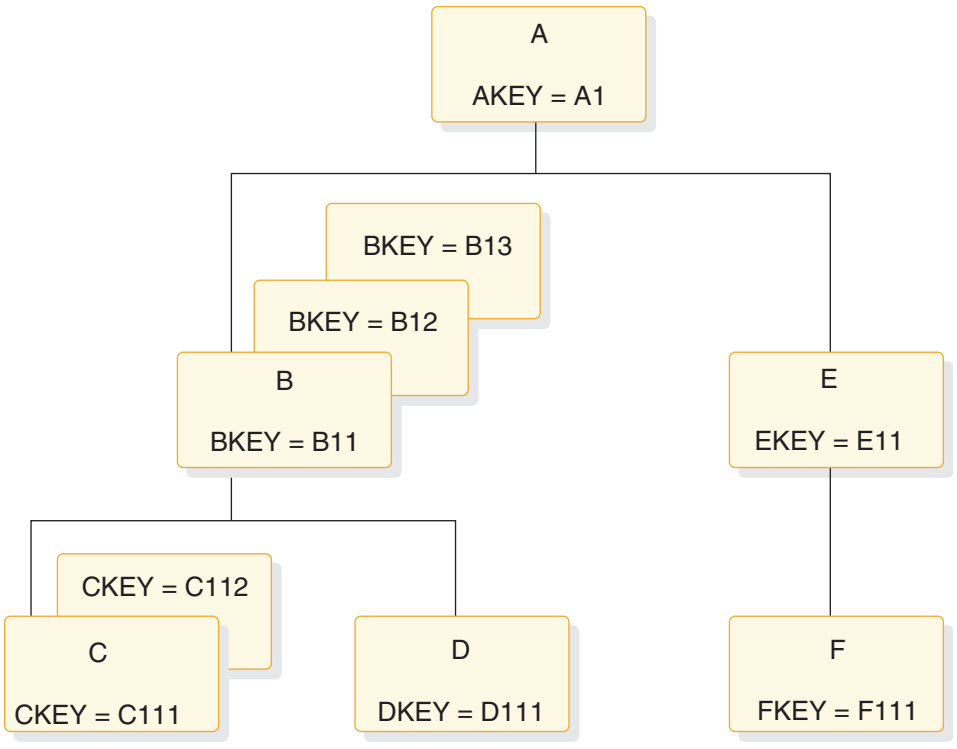


図 *55.* 複数位置付けの階層

単一位置付けを指定した最初の呼び出しを出すと、IMS は A1、B11、および C111 に 3 つの位置を設定します。2 番目の呼び出しを出すと、B11 と C111 にある位置は取り消されます。IMS はその後で A1 と E11 に位置を設定します。

単一位置付けおよび複数位置付けを指定した最初の呼び出しを出すと、IMS は A1、B11、および C111 に 3 つの位置を設定します。しかし、2 番目の呼び出しを出した後、単一位置付けで B11 と C111 上の位置は取り消されますが、複数位置付けでは B11 および C111 上の位置を保持します。さらに IMS は、セグメント A1 および E11 に単一位置付けおよび複数位置付けの位置を設定します。

複数位置付けを指定した最初の呼び出しを出すと、IMS は (単一位置付けの場合と同じように) A1、B11、および C111 に 3 つの位置を設定します。しかし、2 番目の呼び出しを出した後でも B11 および C111 にある位置は、保持されます。IMS はこれらの位置に加えて、セグメント A1 と E11 に位置を保持します。

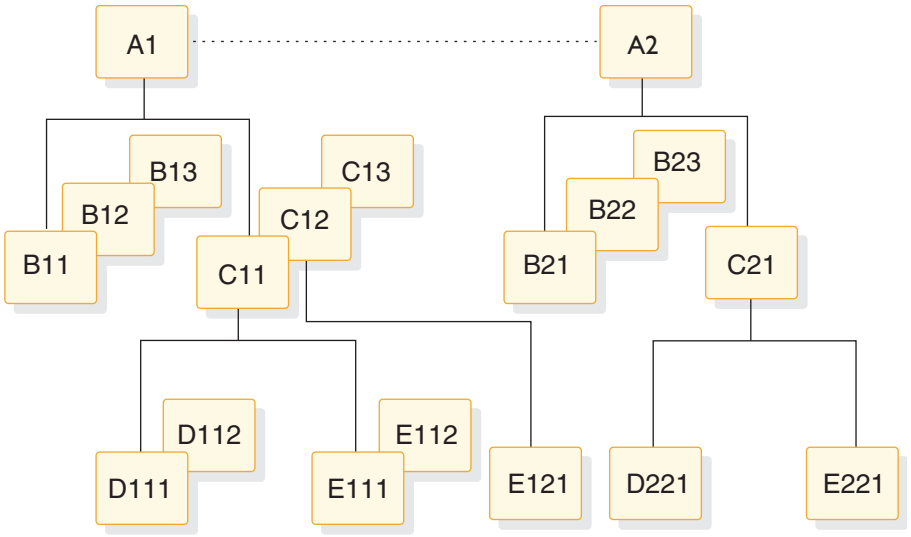


図 *56.* 単一および複数位置付けの階層

以下のいくつかの例で、次の図の階層を使用した場合の単一位置付けの結果と複数位置付けの結果を比較します。

表 *41. DL/I* 呼び出しでの単一位置付けと複数位置付けの結果

順序	単一位置付けの結果	複数位置付けの結果
例 **1**		
GU (AKEY が A1)	A1	A1
GNP B	B11	B11
GNP C	C11	C11
GNP B	検出されませんでした。	B12
GNP C	C12	C12
GNP B	検出されませんでした。	B13
GNP C	C13	C13
GNP B	検出されませんでした。	検出されませんでした。
GNP C	検出されませんでした。	検出されませんでした。
例 **2**		
GU A (AKEY が A1)	A1	A1
GN B	B11	B11

表 41. *DL/I* 呼び出しでの単一位置付けと複数位置付けの結果 (続き)

順序	単一位置付けの結果	複数位置付けの結果
GN C	C11	C11
GN B	B21	B12
GN C	C21	C12
例 **3**		
GU A (AKEY が A1)	A1	A1
GN C	C11	C11
GN B	B21	B11
GN B	B22	B12
GN C	C21	C12
例 **4**		
GU A (AKEY が A1)	A1	A1
GN B	B11	B11
GN C	C11	C11
GN D	D111	D111
GN E	E111	E111
GN B	B21	B12
GN D	D221	D112
GN C	次の A の下の C	C12
GN E	次の A の下の E	E121

複数位置付けは、2 つの階層パスにあるセグメントを検査または比較するときに役立ちます。これを使用すると、同じ親のもとにある別々のセグメント・タイプを並行して処理できます。複数位置付けを使用しない場合には、各パスの中で位置を再設定するための GU 呼び出しを出す必要があります。

複数位置付けを使用する利点

複数位置付けを使用することには、次のような利点があります。

- 単一位置付けを使用する場合よりも、データから独立性が高いプログラムを設計できます。処理されるセグメント・タイプの相対順序に関係なく、GN および GNP 呼び出しを使用するアプリケーション・プログラムや、SSA での欠落レベルで GU および ISRT 呼び出しを使用するアプリケーション・プログラムを、作成できます。セグメント・タイプの相対順序を変更してプログラムのパフォーマンスを改善する際に、これらのセグメント・タイプにアクセスするすべてのアプリケーション・プログラムが複数位置付けを使用する場合は、既存のアプリケーション・プログラムに影響を与えることなく、相対順序を変更できます。複数位置付けを使用しないでこれを行うためには、ユーザーのプログラムで、GN および GNP 呼び出しと、不完全指定の SSA を使用する GU および ISRT 呼び出しを、出す必要があります。
- ユーザーのプログラムは、単一位置付けを使用した場合よりも効率良く、従属セグメント・タイプを並行して処理することができます (GU 呼び出しを出して位置を設定し直さなくても、階層パス間を往復することができます)。必要なセグメントが入っている階層パスを、呼び出しの SSA で IMS に指示します。IMS は、その階層パスで設定された位置を使用して、呼び出しの条件を満たします。それぞれの位置付けのために IMS が作成する制御ブロックは、同じものです。複数位置付けを使用しても、そのために必要なストレージが増えたり、パフォーマンスに大きな影響が及んだりすることはありません。

複数位置付けのほうが単一位置付けよりも、多くのプロセッサー時間を必要とすることがあること、および、複数位置付けは HSAM データベースでは使用できないことに注意してください。

複数位置付けがユーザー・プログラムに与える影響

複数位置付けは、ユーザーの DL/I 呼び出しの順序と構造に影響を与えます。

GU および ***ISRT***

複数位置付けが GU および ISRT 呼び出しに影響を与えるのは、これらの呼び出しで階層パスに SSA を指定していない場合に限られます。階層パスの各レベルごとに SSA が指定されていない GU または ISRT 呼び出しを出すと、IMS は現在位置に応じて以下のように欠落レベルの SSA を作成します。

- IMS の位置が欠落レベルで設定されている場合には、IMS が使用する修飾は、DB PCB に反映されるように、その位置から取り込まれます。
- 欠落レベルに位置が設定されていない場合には、IMS はそのレベルのセグメント・タイプを想定します
- IMS はより高いレベルで設定された位置から前方に移動した場合、そのレベルのセグメント・タイプを想定します。

IMS は、欠落している修飾を現在位置に基づいて作成するため、複数位置付けを使用すると、IMS は同じ親オカレンスの下の他のセグメント・タイプのために設定されている現在位置に関係なく、修飾を完成させることができます。

複数位置付けを使用する ***DLET*** および ***REPL***

複数位置付けは、DLET または REPL 呼び出しには影響を与えません。これらの呼び出しに先行する Get Hold 呼び出しにだけ影響を与えます。

修飾された ***GN*** および ***GNP*** 呼び出し

ユーザーのプログラムが GN または GNP 呼び出しを出すと、IMS は、現在位置から前方に移動して呼び出しの条件を満たそうとします。複数位置付けを使用した場合、現在位置が複数存在することになります。IMS は **1** つの階層パスの各レベルではなく、すべての階層パスの各レベルで位置を保持します。複数位置付けを使用した GN および GNP 呼び出しの条件を満たすために、IMS は、SSA で参照されたパス内の現在位置から前方に移動します。

修飾と非修飾が混在する ***GN*** および ***GNP*** 呼び出し

複数位置付けは、並行処理とデータ独立性のために修飾呼び出しで使用するためのものですが、複数位置付けを指定して非修飾呼び出しを使用したい場合もあります。例えば、ある階層内のすべてのセグメント・オカレンスを、セグメント・タイプとは無関係に、順次にリトリーブしたいものとします。

推奨事項**:** 不整合な結果を避けるために、非修飾呼び出しは GNP 呼び出しに限定してください。修飾された SSA と非修飾 SSA を混用することは、並行処理に有効な場合がありますが、プログラムのデータ独立性を低下させる可能性もあります。

修飾と非修飾が混在する GN および GNP 呼び出しに対し、以下の 3 つの規則が適用されます。

1. 非修飾 GN または GNP を出すと、IMS は、その GN または GNP 呼び出しの条件を満たすために、先行の呼び出しで設定された位置を使用します。以下に例を示します。

プログラムが以下の呼び出しを出します	**DL/I** が以下のセグメントを戻します
GU A (AKEY = A1)	A1
GN B	B11
GN E	E11
GN	F111

   ユーザーのプログラムが非修飾 GN 呼び出し IMS は、最後の呼び出し (E セグメントに関する呼び出し) で設定された位置を使用して非修飾呼び出しの条件を満たします。

2. 非修飾 GN または GNP を使用したセグメントのリトリーブが成功すると、IMS は 1 つの階層パス (すなわち、その呼び出しでリトリーブされたセグメントを含むパス) だけで位置を設定します。 IMS は、他の階層パスの位置を取り消します。IMS は、リトリーブされたセグメントで現在位置を設定し、リトリーブされたセグメントの親で親子関係を設定します。非修飾呼び出しを出した後で、異なる階層パスにあるセグメントに関して修飾呼び出しを出すと結果は予想できなくなります。以下に例を示します。

プログラムが以下の呼び出しを出します	**DL/I** が以下のセグメントを戻します
GU A (AKEY = A1)	A1
GN B	B11
GN E	E11
GN	F111
GN B	予測不能

   ユーザーが非修飾 GN 呼び出しを出すと、IMS は他の階層パスにある位置を保持しなくなるため、B セグメントに関して GN 呼び出しを出した場合の結果は予測できません。

3. 非修飾 GN または GNP 呼び出しを出した場合に、IMS が、その非修飾呼び出しで検出される可能性のあるセグメントに位置を設定していると、呼び出しの結果は予測できません。また、非修飾呼び出しを出したときに、その呼び出しで「リトリーブされるべき」セグメントにユーザーが位置を設定している場合にも、結果は予測できません。

   以下に例を示します。

プログラムが以下の呼び出しを出します	**DL/I** が以下のセグメントを戻します
GU A (AKEY = A1)	A1
GN E	E11
GN D	D111
GN B	B12

プログラムが以下の呼び出しを出します	**DL/I** が以下のセグメントを戻します
GN B	B13
GN	E11 (IMS がもつ位置は GN 呼び出しで設定された位置のみです。)

この例では、IMS の位置は E11 に設定されます。非修飾 GN 呼び出しは、先行の呼び出しで設定された位置から前方に移動します。複数位置は失われます。したがって、IMS の位置は GN 呼び出しで設定された位置のみになります。

上記の規則を要約すると、次のようになります。

1. 非修飾の GN または GNP 呼び出しの条件を満たすために、IMS はその PCB に関して出された最後の呼び出しで設定された位置を使用します。
2. 非修飾 GN または GNP 呼び出しが成功すると、IMS は他のすべての階層パス内の位置を取り消します。リトリーブされたセグメントの経路内にある位置だけが保持されます。

複数位置付けでの位置のリセット

位置をリセットするためには、ユーザーのプログラムでルート・セグメントに関する GU 呼び出しを出してください。現在処理中のデータベース・レコード内の位置のリセットが必要な場合には、そのルート・セグメントに対して GU 呼び出しを発行できますが、この GU 呼び出しをパス CALL にすることはできません。

例: セグメント B11 および E11 に位置が設定されているものとします。ユーザー・プログラムで以下のいずれかの呼び出しを出して、次のデータベース・レコードに位置をリセットできます。

この呼び出しを出すと、IMS はデータベース・レコード A1 にあるすべての位置を取り消します。

```
GU   AbbbbbbbAKEYbbbb=bA2)
```

また、レコード A1 にあるセグメントの処理を続けたい場合には、次の呼び出しを出してレコード A1 にあるすべての位置を取り消します。

```
GU    AbbbbbbbAKEYbbbb=bA1)
```

この呼び出しをパス CALL として出すと、位置は取り消されません。

複数 DB PCB

プログラムに複数の PCB がある場合、通常は、複数のデータベースの視点が定義されていることになりますが、1 つのデータベース・レコードに複数の位置が必要であることを意味している場合もあります。データベースの同じ階層視点について複数の PCB を定義することによっても、1 つのデータベース・レコードに複数の位置を保持できます。

複数の PCB を使用すると、複数位置付けの機能を拡張することにもなります。これは、複数の PCB を使用すると、同じレコード内の 2 つ以上の階層パスの他に、2 つ以上のデータベース・レコードでも位置を保持することができるためです。

例えば、患者 A のデータベース・レコードを処理していて、その後患者 B のレコードを調べ、患者 A の位置に戻る必要があるとします。ユーザー・プログラムが医療階層について複数の PCB を使用している場合、PCB1 を使用して患者 A について最初の呼び出しを発行し、それから PCB2 を使用して患者 B について次の呼び出しを発行します。患者 A のレコードに戻りたい場合には、PCB1 を使用して次の呼び出しを出します。これにより、そのデータベース・レコード内の元の位置に戻ることができます。

複数の PCB を使用すると、位置を保持するために必要な Get 呼び出しの数を減らすことができ、また、パフォーマンスが向上することもあります。特に複数のデータベース・レコードのセグメントから得られた情報を比較したいときには、複数の PCB を使用すると特に便利です。ただし、PCB を定義するごとに、内部制御ブロックの要件は増加します。

PSB 生成時に、各 PCB に対して別々の PCBNAME を割り当てることにより、複数の PCB で AIBTDLI インターフェースを使用できます。複数の PCB に PSB PCBLIST で異なるアドレスを割り当てる必要があるように、AIBTDLI インターフェースを使用する場合には、複数の PCB に異なる PCBNAME を割り当てる必要があります。例えば、ユーザーのアプリケーション・プログラムが、同じデータベースを識別するリスト内の 2 つの異なる PCB に対して DL/I 呼び出しを出すと、PSB 生成時に 2 つの PCB で異なる PCBNAME を使用したときの AIBTDLI インターフェースと同じ効果を得ることができます。

第 16 章 IMS アプリケーション・プログラム同期点の使用

IMS アプリケーション・プログラムは、チェックポイントを取ることができます (およびその必要があります)。これらのチェックポイントとシステム同期点は、IMS の操作に影響を与える可能性があります。

コミット処理

アプリケーションの同期点処理の間に、IMS はログ・レコードを作成し、データベース変更のコミットメントを行い、出力メッセージを使用可能に設定します。このコミット処理は、IMS がこのログ・レコードを OLDS に物理的に書き込むまでは完了しません。なぜなら、一連の不完全なデータベース変更レコードとメッセージ・レコードがシステム再始動用のログに存在するからです。

コミット処理で行われる作業は、全機能アプリケーションの場合と高速機能アプリケーションの場合とでは異なります。全機能の場合、IMS は、DL/I 呼び出し時にバッファー・プール内でデータベース変更を行います。そして、コミット・ポイントより前に、変更をディスクに書き込むことができます。システムが再始動されると、IMS はログを使用してこれらのアンコミット変更をバックアウトします。IMS は挿入されたメッセージ・セグメントをメッセージ・キューに保管しますが、IMS はそれらも同様に廃棄する必要があります。

高速機能の場合、IMS は、コミット・レコードを物理的にログに記録するまでは、すべての変更をメモリー内で保持します。ログに記録してはじめて、IMS はデータベース変更を DASD に書き込み、出力メッセージを送信します。コミット・レコードが書き込まれるまでは、外部ストレージにはいかなる変更も加えられることがないため (ただしログは除く)、IMS はデータベースに対してバックアウト処理を実行しません。IMS は、メモリー内の更新を廃棄します。高速機能の場合は、システムを再始動するときに、IMS は必ずコミット済みの更新を DASD に書き込み、出力メッセージを送信したことを確認します。

チェックポイントと同期点との関係

IMS は、すべてのチェックポイントと同期点をトラッキングします。IMS は通常、リカバリー中に同期点を使用しますが、次のような状況ではチェックポイントに戻ります。以下の図に例を示すように、DB/DC 環境でシステム全体にわたる障害の発生時点が、MTO のシステム・チェックポイント設定の直後であり、かつプログラム B のコミット処理の直前であった場合 (プログラム A はその最後にコミットした時点から一切更新を行っていないものとします)、IMS は、Beta の開始以前のシステム・チェックポイントに戻る必要があります。

- DB/DC 環境での完全リカバリーの場合。IMS は、現行チェックポイントより前のチェックポイントまたは最初のアンコミット・アプリケーション・プログラム更新より前のチェックポイントのうち、最も古いチェックポイントに戻ります。
- DBCTL 環境での完全リカバリーの場合。IMS は、常に最初のアンコミット・アプリケーション・プログラム更新以前のチェックポイントに戻ります。

- DCCTL 環境での完全リカバリーの場合。IMS は、常に最新のシステム・チェックポイント以前のチェックポイントに戻ります。
- DB/DC 環境または DCCTL 環境で再始動時に BUILDQ が要求された場合。IMS は、最後の SNAPQ または DUMPQ チェックポイントに戻ります。 IMS は、このチェックポイントが再始動時に通常必要とされるチェックポイントより古かったとしても、このチェックポイントに戻ります。

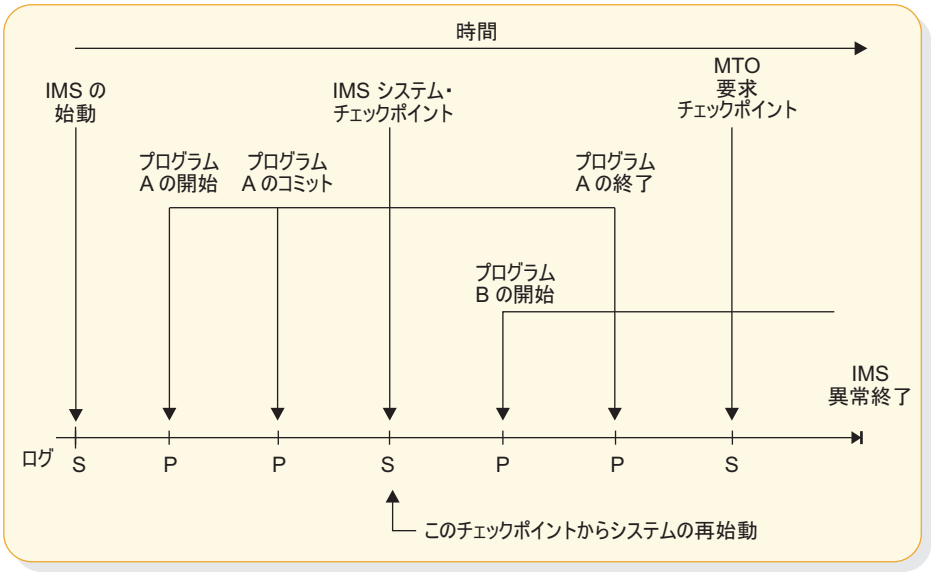


図 57. システム・チェックポイントおよびアプリケーション同期点の独立性

CPI 通信ドリブン・プログラムにおける同期点処理

拡張プログラム間通信機能 IMS (APPC/IMS) の元で稼働する CPI 通信ドリブン・プログラムの場合は、アプリケーション・プログラムはそのプログラム独自の同期点処理を制御します。アプリケーション・プログラムは、特定の CPI リソース・リカバリー呼び出し (データをコミットするための SRRCMIT 呼び出しおよびデータをバックアウトするための SRRBACK 呼び出し) を出すことができます。IMS (ローカル) で管理される保護リソースには、以下のものが含まれます。

- IMS TM メッセージ・キュー・メッセージ
- IMS DB データベース
- Db2 for z/OS データベース

会話用にサポートされる同期のうち最高水準のものは SYNCPT であるため、CPI 通信ドリブン・アプリケーションは保護会話を行えます。

同期点マネージャーとリソース・マネージャー

設定されている同期点レベルに応じて、IMS は同期点マネージャーにもリソース・マネージャーにもなり得ます。 SYNCLVL=NONE または CONFIRM、および AOS=B、S、または X の場合は、IMS は同期点マネージャーおよびリソース・マネージャーとなります。ただし RRS=Y および SYNCLVL=SYNCPT の場合は、z/OS リソース・リカバリー・サービス (RRS) が同期点マネージャーとなり、IMS

がリソース・マネージャーとなります。RRS=N の場合は、IMS は同期点マネージャーとなります。

同期化処理における 2 フェーズ・コミット

DBCTL、DCCTL、DB/DC、APPC/IMS、または OTMA 環境では、アプリケーション・プログラムは 2 フェーズ・コミット処理に参加して同期点を記録できます。2 フェーズ・コミットの完了時に、リソース・マネージャーはデータベース変更とメッセージ変更をコミットします。

この 2 つのフェーズは次のとおりです。

1. フェーズ 1。ここでは、同期点コーディネーターが同期点の準備を指図し、接続されているデータベースへの更新をコミットできるかどうかを接続されているリソース・マネージャーに尋ねます。

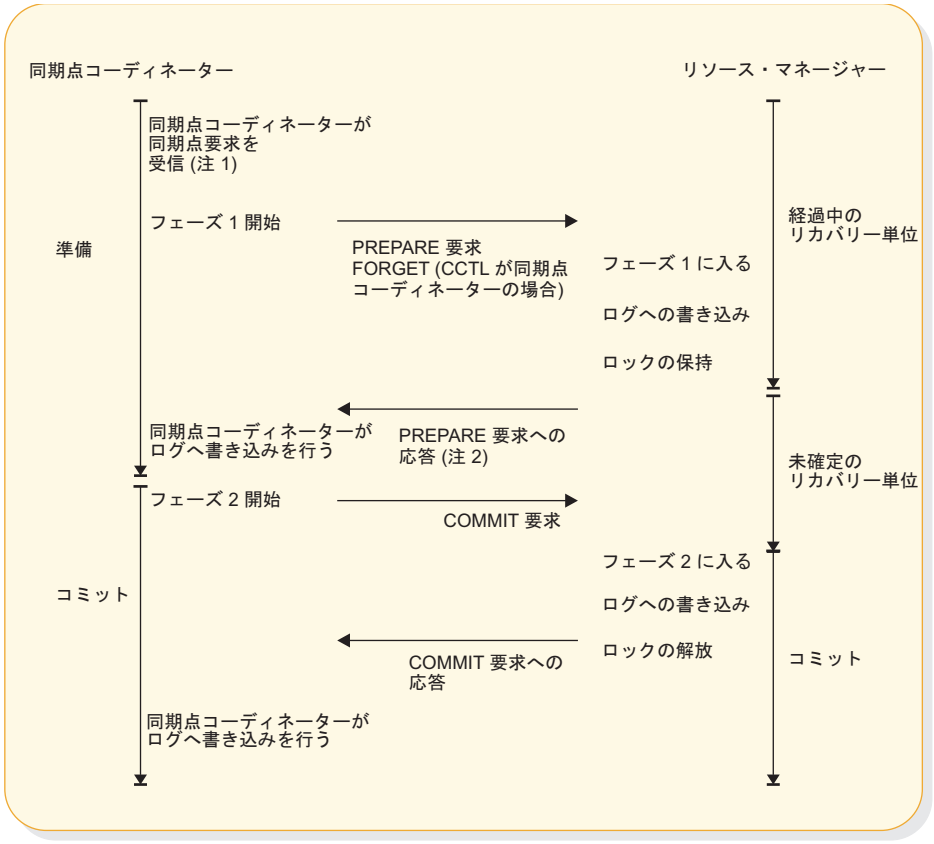
   以下に、この同期点コーディネーターになり得るものを示します。

   - IMS DB/DC サブシステム (このサブシステムのリソース・マネージャーおよびこのサブシステムにタスクを生成されているデータベースの場合)
   - IMS DCCTL サブシステム (タスクを生成されているデータベースの場合)
   - コーディネーター・コントローラー (CCTL) サブシステム (CCTL 領域に関連付けられている作業単位の場合)。IMS DB が、CCTL に接続されているとき、およびオープン・データベース・アクセス (ODBA) インターフェースを介して ODBA アプリケーション・プログラムからアクセスされる場合にも、リソース・マネージャーとして働きます。
   - z/OS リソース・リカバリー・サービス (RRS) (このサービスが APPC/IMS アプリケーション・プログラムまたは OTMA クライアントと保護会話を行う場合)。IMS が、RRS に接続されているときにリソース・マネージャーとして働きます。
2. フェーズ 2。ここでは、同期点コーディネーターはコミット処理または打ち切り処理を指図し、リソースのコミットまたは打ち切りが必要であることを伝えます。

   DBCTL 環境では、アプリケーション・プログラムが更新 DL/I 呼び出しをまったく行わなかったか、あるいは照会タイプの DL/I 呼び出しだけしか行わなかった場合は、CCTL はフェーズ 1 に対して「FORGET (無視)」応答を要求します (「無視」の処理が使用可能となっている場合)。これは、どのデータベース・リソースも更新されていないために、そのアプリケーション・プログラムについてのフェーズ 2 は限定的なものになることを意味します。「無視」の処理を使用可能にする方法についての詳細は、「*IMS V15* 出口ルーチン」を参照してください。

   同期点コーディネーターがフェーズ 1 を行うことなく打ち切りを要求する場合もあります。

以下の図は、IMS DBCTL 環境用の 2 フェーズの同期点サイクルを示しており、発生する処理が記載されています。



注:

1. リソース・マネージャーから更新をコミットできないことを指示された場合は、同期点コーディネーターは当該のリカバリー単位を打ち切らなければならない。その場合は、この図の残りの部分は適用されません。
2. 同期点コーディネーターがリソース・マネージャーに更新のコミットを指示した場合は、リソース・マネージャーはコミットしなければならない。

図 *58. 2* フェーズ・コミット処理

リカバリー単位

リカバリー単位 (UOR) とは、同期点インターバルの間、つまり、ある同期点から次の同期点までの間にスレッド (リソース・マネージャー制御領域と同期点調整プログラムとの接続) によって行われる作業のことです。

経過中のリカバリー単位

リカバリー単位は、それが作成された時点またはその最後の同期点から、リソース・マネージャーがフェーズ 1 終了をログに記録するまでの間は、経過中 であると言われます。フェーズ 1 以前またはフェーズ 1 の間にリソース・マネージャーに障害が起こってその後に再始動された場合は、IMS はすべてのデータベース更新を打ち切ります。

CCTL に接続された DBCTL の未確定リカバリー単位

リソース・マネージャーが PREPARE 要求への応答を出した時点 (フェーズ 1 の完了) から、このリソース・マネージャーが CCTL から COMMIT 要求または ABORT 要求を受信するまでの間は、リカバリー単位は未確定 であると言われます。リソース・マネージャーは、障害が起こってその後再始動されたときには、どの未確定 UOR が存在しているのかを CCTL に通知します (存在する場合)。すると、CCTL はこれらの未確定 UOR を解決するためのアクションを実行します。これは、未確定の解決処理または再同期と呼ばれます。CCTL がこれらの未確定 UOR のすべては解決できなかった場合は、IMS または CCTL のコマンドを使用し、リカバリー単位を表示させてそれらをコミットまたは打ち切るための適切なアクションを実行できます。

CCTL に接続された DBCTL 用のリカバリー・トークン

リカバリー・トークン は、各リカバリー単位用の 16 バイトの ID です。リソース・マネージャーは、このリカバリー・トークンに対する妥当性検査を行って、リカバリー単位が重複しないように保護します。 DBCTL 環境では、IMS `/DISPLAY CCTL` コマンドを使用してリカバリー・トークンを表示できます。リカバリー・トークンは、リカバリー単位管理を行う DBRC によって使用される 1 次 ID です。DBRC は、バッチ・バックアウト・ユーティリティーが実行するのが適当な各バックアウトに対するトラッキングを行います。

リカバリー可能な未確定構造

IMS DBCTL サブシステムは、以下のいずれかの障害が起こった場合は、常にそれぞれの未確定 UOR ごとにリカバリー可能な未確定構造 (RIS) を作成します。

- CCTL の障害
- CCTL スレッドの障害
- リソース・マネージャーの障害

リソース・マネージャーは、CCTL またはリソース・マネージャーのいずれかに障害が起こったときに未確定 UOR が存在していた場合は、その CCTL への再接続時にリカバリー可能な未確定構造を使用します。 IMS は、システム・チェックポイント時にすべてのリカバリー可能な未確定構造をログに記録します。

リカバリー可能な未確定構造には、以下の情報が含まれています。

- 残りのリカバリー・エレメント (RRE) 内のリカバリー・トークン
- 未確定拡張エラー・キュー・エレメント (IEEQE) 内の変更済みデータ・レコード
- 未解決の未確定 UOR のためにアクセス不能となっているデータの指示
- 拡張エラー・キュー・エレメント (EEQE) キュー・エレメント (EQEL) を使用している他のリカバリー可能な未確定構造へのリンク

DBCTL 単一フェーズ・コミット

1 つのリソース・マネージャー (IMS DBCTL サブシステム) とだけ通信している CCTL は、単一のフェーズのみを使用して同期点を要求できます。CCTL が複数のリソース・マネージャーと通信している場合は、その CCTL は 2 フェーズ・コミット処理を使用する必要があります。

CCTL が UOR をコミットすることにした場合は、単一フェーズ同期点を要求できます。単一フェーズ・コミットは、未確定データのリカバリー可能性に影響を及ぼす可能性があります。トランザクションが未確定となるのは、同期点要求と DBCTL によるコミットとの間の短い時間だけです。 IMS は、単一フェーズ・コミット時のスレッド障害後の未確定データはリカバリーできますが、サブシステム障害後の未確定データはリカバリーできません。

同期点ログ・レコード

2 フェーズ・コミット処理の間に、IMS はデータベース変更のコミットメントを確立するためのログ・レコードを作成します。これらのすべてのログ・レコードは、IMS の変更累積ユーティリティーおよびリカバリー・ユーティリティーから使用可能です。

同期点サイクルに関与するすべてのオンライン・ログ・レコードにはリカバリー・トークンが含まれています。このトークンは、IMS が各リカバリー単位をリカバリーして再始動できることを確認します。リカバリー単位用のログ・レコードのシーケンスを調べれば、そのリカバリー単位がたどった同期点サイクルが明らかになります。

IMS は、同期点処理の間に以下のレコードをログに記録します。

ログ・レコード
: 説明

X'08' スケジュール・レコード

X'07' スケジュールされていない (終了) レコード

X'0A08'
: CPI 通信ドリブン・アプリケーション・プログラムのスケジュール・レコード

X'0A07'
: CPI 通信ドリブン・アプリケーション・プログラムのスケジュールされていない (終了) レコード

X'5945'
: 高速機能 64 ビット・バッファー使用量

X'5937'
: 高速機能開始コミット

X'5938'
: 高速機能開始打ち切り

X'5610'
: フェーズ 1 開始

X'5611'
フェーズ 1 終了

X'3730'
フェーズ 2 開始コミット

X'5612'
フェーズ 2 終了コミット

X'3801'
打ち切り開始

X'4C01'
打ち切り終了

X'5607'
リカバリー単位開始

X'5613'
作成済みリカバリー可能な未確定構造

X'5614'
削除済みリカバリー可能な未確定構造

データ伝搬マネージャーでの同期点

データ伝搬マネージャー (例えば IMS DataPropagator) を使用して Db2 for z/OS データベースを IMS DL/I データベースと同期させて更新している場合、Db2 for z/OS データベースへの更新は、IMS 更新と同時にコミットされます (または打ち切られます)。これにより、データベース管理サブシステム間の整合性が実現されています。 IMSDB/DC、DCCTL、および DBCTL (BMP 領域のみ) は、IMS データ・キャプチャー出口ルーチンをサポートします。

制約事項**:** IMS DBCTL 環境では、データ伝搬マネージャーが使用可能なのは BMP 領域に対してのみです。

IMS DataPropagator について詳しくは、Web URL の http://www.ibm.com/software/data/db2imstools/imstools/imsdprop.html にアクセスしてください。

第 17 章 データベースのリカバリーとデータベース保全性の維持

アプリケーション・プログラムでは、チェックポイントを発行し、プログラムを再始動し、データベースの保全性を維持することができます。

Java バッチ処理 (JBP) 領域で実行されている Java アプリケーションは、IMS Java 従属領域リソース・アダプターを使用することで、シンボリック・チェックポイントおよび再始動呼び出しを発行できます。

関連概念:

837 ページの『IMS Java 従属領域リソース・アダプターを使用した JBP アプリケーションの開発』

チェックポイントの発行

チェックポイント (CHKP) 呼び出しには、基本 CHKP とシンボリック CHKP の 2 種類があります。すべての IMS プログラムおよび CICS 共用データベース・プログラムは、基本 CHKP 呼び出しを使用できます。 BMP およびバッチ・プログラムだけは、どちらの呼び出しを使用することもできます。

「*IMS V15* アプリケーション・プログラミング *API*」には、には、ユーザーのプログラムで、いつ、どのような目的でチェックポイントを発行する必要があるかが説明されています。どちらのチェックポイント呼び出しを使用した場合にも、データベースの位置は失われます。したがって、呼び出し後に GU 呼び出しまたはその他の方法で位置を再設定しなければなりません。非ユニーク・キーを持つセグメントまたはキーのないセグメントの中に位置を再設定することはできません。

制約事項**:** IMS のチェックポイントをとるために、DD ステートメントに CHKPT=EOV を指定してはなりません。

同じ呼び出しシーケンスを、全機能データベースまたは DEDB に対して発行する場合と、MSDB に対して発行する場合とでは、いくらかの違いがあります。

データベース編成によっては、CHKP 呼び出しを出すと PCB のデータベース位置がリセットされることがあります。CHKP 呼び出しが出されると、プログラムによって保持されていたロックは解除されます。したがって、ユーザーのデータベース位置を維持するためにロックが必要な場合、CHKP 呼び出しによって位置がリセットされてしまいます。編成が GSAM の場合 (ロックが使用されません)、あるいは編成が DEDB であって GC 状況コードの後で CHKP 呼び出しが出された場合を除き、位置のリセットは常に行われます。DEDB の場合、位置は操作単位境界で維持されます。

CHKP を出すと、変更可能代替 PCB の宛先がリセットされます。

関連資料**:** CHKP 呼び出しについて詳しくは、「*IMS V15* アプリケーション・プログラミング *API*」のトピックである『CHKP (基本) 呼び出し』および『CHKP (シンボリック) 呼び出し』を参照してください。

関連概念:

379 ページの『MSDB および DEDB におけるコミット・ポイント処理』

最新のチェックポイントからのプログラムの再始動

シンボリック・チェックポイントの代わりに基本チェックポイントを使用する時は、プログラムの異常終了の場合において最新のチェックポイントから再始動するために必要なコードを提供してください。

プログラムを最新のチェックポイントから再始動させる方法として、 HDAM または PHDAM データベースに位置変更情報を保管しておくことができます。この方法により、ユーザーのプログラムは、チェックポイントが出されるたびに、位置変更情報を含むデータベース・レコードをデータベースに書き込みます。プログラムを終了させる前に、このデータベース・レコードを削除する必要があります。

XRST 呼び出しについて詳しくは、「*IMS V15* アプリケーション・プログラミング *API*」のトピック『XRST 呼び出し』を参照してください。

データベース保全性の維持 (IMS バッチ、BMP、および IMS オンライン領域)

データベースの更新をバックアウトするために IMS が使用する DL/I 呼び出しは、ROLB、ROLL、 ROLS、SETS および SETU です。

ROLB および ROLS 呼び出しを使用すると、プログラムの最後のコミット・ポイント以降に行われたデータベース更新をバックアウトしたり、作成された出力メッセージを取り消したりできます。ROLL 呼び出しは、データベースの更新をバックアウトし、プログラムが最後のコミット・ポイント以降に作成した非高速出力メッセージをすべて取り消します。また、現行の入力メッセージも削除します。SETS を使用すると、アプリケーション・プログラム処理中に複数の中間バックアウト・ポイントを記憶できます。 SETU は、PSB 内のサポートされない PCB によってリジェクトされないことを除けば、 SETS と似た働きをします。ユーザーのプログラムで、これらのポイントのいずれかを指定して後続の ROLS 呼び出しを発行すると、そのポイント以降に行われたデータベース更新とメッセージ・アクティビティーがバックアウトされます。

DBCTL を使用すると CICS オンライン・プログラムは、ROLS および SETS または SETU DL/I 呼び出しを使用して、前のコミット・ポイントまたは中間バックアウト・ポイントにデータベースをバックアウトできます。

前のコミット・ポイントへのバックアウト: ROLL、ROLB、および ROLS

プログラムがその処理の一部に誤りがあると判断したときには、いくつかの呼び出しを使用して誤った処理の結果を削除できます。このような呼び出しをロールバック呼び出しといいます。 ROLL、DB PCB を使った ROLS (または入出力域やトークンを使用しない ROLS) および ROLB の各呼び出しがこれに該当します。

ユーザーがこれらのいずれかの呼び出しを出すと、IMS は次のことを行います。

- プログラムの最後のコミット・ポイント以降に行われたデータベース更新をバックアウトする。
- プログラムの最後のコミット・ポイント以降に作成された非高速出力メッセージを取り消す。

これらの呼び出しの相違点は、ROLB は更新のバックアウトと出力メッセージの取り消しのあとでアプリケーション・プログラムに制御を戻し、 ROLS はアプリケーション・プログラムに制御を戻さずに、 ROLL は異常終了コード U0778 でプログラムを終了させます。ROLB は、最新のコミット・ポイント以降の最初のメッセージ・セグメントをプログラムに戻すことができますが、ROLL および ROLS はこれを行うことができません。

ROLL および ROLB 呼び出し、および指定されたトークンを使用しない ROLS 呼び出しは、PSB に GSAM データ・セットの PCB が含まれている場合に有効です。ただし、最後のコミット・ポイント以降に GSAM データ・セットに挿入されたセグメントは、これらの呼び出しではバックアウトされません。拡張チェックポイント再始動を使用して、再始動時に GSAM データ・セットを位置変更することができます。

ROLS 呼び出しは、前のコミット・ポイントにバックアウトするためにも、前の SETS 呼び出しによって設定された中間バックアウト・ポイントにバックアウトするためにも使用できます。この節では、前のコミット・ポイントにバックアウトする ROLS 呼び出しの形式についてのみ説明します。 ROLS のその他の形式については、『中間バックアウト・ポイントへのバックアウト: SETS、SETU、および ROLS』を参照してください。

以下の表では、ROLB、ROLL、および ROLS の各呼び出しの類似点と相違点を要約しています。

表 *42. ROLB*、*ROLL*、および *ROLS* の比較 :

取られる処置	**ROLB**	**ROLL**	**ROLS**
最後のコミット・ポイント以降のデータベース更新をバックアウトする。	X	X	X
最後のコミット・ポイント以降に作成された出力メッセージを取り消す。	X[1]	X[1]	X[1]
処理中のメッセージをキューから削除する。最後のコミット・ポイント以降に処理された以前のメッセージがある場合は、それを再処理のためにキューに戻す。		X	
最後のコミット・ポイント以降に出された最初の入力メッセージの最初のセグメントを戻す。	X[2]		
U3303 異常終了。処理済みの入力メッセージをメッセージ・キューに戻す。			X[3]
U0778 異常終了。ダンプはなし。		X	
異常終了なし。プログラムの処理が続行される。	X		

注**:**

1. プログラムが PURG を出していない限り、 ROLB、ROLL、または ROLS 呼び出しは、高速 PCB を使用して送られた出力メッセージを取り消します。例え

ば、プログラムが、以下のような呼び出しシーケンスを出した場合、MSG1 は宛先に送られます。これは、PURG により MSG1 が完了していて入出力域に次のメッセージ (この例では MSG2) の最初のセグメントが入っていることが IMS に通知されるためです。しかし、MSG2 は取り消されます。

```
ISRT   EXPRESS PCB, MSG1
PURG   EXPRESS PCB, MSG2
ROLB   I/O PCB
```

IMS に完全なメッセージ (MSG1) があるため、さらに高速 PCB を使用しているために、コミット・ポイントの前にメッセージを送ることができます。

2. 入出力域のアドレスを、呼び出しパラメーターの 1 つとして指定した場合のみ返されます。
3. トランザクションは延期され、以降の処理のために再キューイングされます。

ROLL 呼び出し

ROLL 呼び出しは、データベースの更新をバックアウトし、プログラムが最後のコミット・ポイント以降に作成した非高速出力メッセージをすべて取り消します。また、現行の入力メッセージも削除します。最後のコミット・ポイント以降に処理されたその他の入力メッセージがあれば、再処理のためにキューに戻されます。IMS はその後で、異常終了コード U0778 を出してプログラムを終了させます。このタイプの異常終了は、ストレージ・ダンプなしでプログラムを終了します。

ROLL 呼び出しを出すときに指定するパラメーターは、呼び出し機能の ROLL だけです。

バッチ・プログラムでは、ROLL 呼び出しを使用することができます。ユーザーのシステム・ログが DASD 上にあり、BKO 実行パラメーターを使用して動的バックアウトが指定されていると、最後のコミット・ポイント以降に行われたデータベースの変更がバックアウトされます。それ以外の場合には、バックアウトされません。バッチ・プログラムで ROLL を出す理由の 1 つとして、互換性の維持があります。

バックアウトが完了すると、元のトランザクションは (廃棄可能であれば) 廃棄され、再実行されません。 IMS は、リモート・トランザクション・プログラムに通知を出すための TPI を指定して、APPC/MVS verb、ATBCMTP TYPE(ABEND) を出します。APPC/MVS verb を使用すると、アクティブな会話は (アプリケーション・プログラムによって作成されたものも含めて)、すべて DEALLOCATED TYP(ABEND_SVC) になります。

ROLB 呼び出し

ROLB 呼び出しを使用した場合の利点として、ROLB 呼び出しの実行後に IMS がプログラムに制御を戻すため、プログラムの処理を続行できることが挙げられます。 ROLB 呼び出しのパラメーターは次のとおりです。

- 呼び出し機能 ROLB
- I/O PCB または AIB の名前

ROLB 呼び出しの全体的な効果は、その呼び出しを出した IMS アプリケーション・プログラムのタイプによって異なります。

- 現行 IMS アプリケーション・プログラムの場合:

  IMS バックアウトが完了すると、元のトランザクションが IMS アプリケーション・プログラムに返されます。IMS によってロールバックできないリソースは無視されます。例えば、高速代替 PCB に対して出力が送られ、PURG 呼び出しが ROLB 呼び出し前に出された場合などです。
- 修正 IMS アプリケーション・プログラムの場合:

  現行の IMS アプリケーション・プログラムの場合と同じ考慮事項が該当します。アプリケーション・プログラムは、作成済み会話があれば、 ROLB が出されたこと通知しなければなりません。
- CPI-C ドリブン IMS アプリケーション・プログラムの場合:

  IMS リソースのみが影響を受けます。すべてのデータベース変更がバックアウトされます。非高速代替 PCB に挿入されたメッセージは、廃棄されます。また、PURG 呼び出しが出されていない至急 PCB に挿入されたメッセージも廃棄されます。アプリケーション・プログラムは、ROLB 呼び出しが出されたことを、もとのリモート・プログラムおよび作成済み会話に通知する必要があります。

MPP およびトランザクション指向 BMP の場合

プログラムが ROLB パラメーターの 1 つとして入出力域のアドレスを提供した場合、 ROLB 呼び出しはメッセージリトリーブ呼び出しとして働き、最後のコミット・ポイント以降に出された最初の入力メッセージの最初のセグメントを戻します。これが正しいのは、最後のコミット・ポイント以降にプログラムが GU 呼び出しをメッセージ・キューに発行している場合だけです。出していない場合は、ROLB 呼び出しを発行したときメッセージの処理は行われません。

プログラムが、ROLB 呼び出しを出した後でメッセージ・キューに対して GN 呼び出しを出した場合、IMS は、 ROLB 呼び出しが出されたときに処理されていたメッセージの次のセグメントを戻します。そのメッセージにそれ以上のセグメントがない場合には、IMS は QD 状況コードを戻します。

ROLB 呼び出しの後にプログラムが GU 呼び出しをメッセージ・キューに出すと、IMS は、次のメッセージの最初のセグメントをアプリケーション・プログラムに返します。そのメッセージ・キューにプログラムが処理すべきメッセージが他にない場合には、IMS は QC 状況コードを戻します。

入出力域をパラメーターに含めていても、最後のコミット・ポイント以降に GU 呼び出しをメッセージ・キューに適切に発行できなかった場合は、IMS は、QE 状況コードをプログラムに返します。

ROLB 呼び出しで入出力域のアドレスを指定しなかった場合にも、IMS は同じことを行います。プログラムがコミット・インターバル中に GU 呼び出しを正常に出し、そのあとで GN 呼び出しを出した場合、IMS は QD 状況コードを戻します。プログラムが ROLB 呼び出しのあとで GU 呼び出しを出した場合には、 IMS は、次のメッセージの最初のセグメントを戻すか、あるいは、プログラムに関するメッセージがそれ以上ない場合には QC 状況コードを戻します。

最後のコミット・ポイント以降に GU 呼び出しを正常に出していないときに、ROLB 呼び出しで入出力域パラメーターを指定していない場合には、IMS は最後のコミット・ポイント以降のデータベース更新をバックアウトし、作成された出力メッセージを取り消します。

バッチ・プログラム

ユーザーのシステム・ログが DASD 上にあり、BKO 実行パラメーターを試用して動的バックアウトが指定されていた場合には、バッチ・プログラムで ROLB 呼び出しを使用できます。この場合の ROLB 呼び出しは、MPP の場合に行うようなメッセージ処理は行いません。 ROLB は、最後のコミット・ポイント以降のデータベース更新をバックアウトし、ユーザーのプログラムに制御を戻します。呼び出しのパラメーターの 1 つとして入出力域のアドレスを指定することはできません。このような指定を行うと、AD 状況コードがユーザーのプログラムに戻されます。ただし、ユーザーのプログラムのために I/O PCB を指定する必要があります。プログラムの PSB の PSBGEN ステートメントの CMPAT キーワードに CMPAT=YES を指定します。

ROLS 呼び出し

前のコミット・ポイントにバックアウトし、処理済みの入力メッセージをあとで再処理するために IMS に戻すには、ROLS 呼び出しを 2 通りの方法で使用できます。

- I/O PCB を使用して、プログラムで ROLS 呼び出しを発行する。ただし、呼び出しに入出力域とトークンは指定しません。この形式の ROLS 呼び出しのパラメーターは以下のようになります。

  呼び出し機能 ROLS

  I/O PCB または AIB の名前
- データを使用できない状況コードの 1 つを受け取ったデータベース PCB を使用して、プログラムで ROLS 呼び出しを発行する。この場合の結果は、使用不能データが検出され、INIT 呼び出しが出されなかったときと同じになります。ROLS 呼び出しは、その PCB に関する次の呼び出しとして出さなければなりません。他の PCB を使用して呼び出しに介入することもできます。

TOKEN を指定した ROLS 呼び出しの場合、IMS によって処理されたすべてのメッセージを含めて、すべての非至急メッセージについてメッセージ・キューの再位置付けが行われることがあります。この処理は、APPC/MVS 呼び出しを使用して行われ、初期のメッセージ・セグメントも対象に含まれます。元の入力トランザクションは、IMS アプリケーション・プログラムに返すことができます。入力および出力の位置設定は、SETS 呼び出しによって決定されます。この位置付けは、現行および変更 IMS アプリケーション・プログラムに適用され、CPI-C ドリブンの IMS プログラムには適用されません。IMS アプリケーション・プログラムは、すべてのリモート・トランザクション・プログラムに、ROLS について通知しなければなりません。

TOKEN を使用しない ROLS 呼び出しでは、IMS は TPI を指定した APPC/MVS verb、ATBCMTP TYPE(ABEND) を出します。この verb を出すと、アプリケーション・プログラムに関連したすべての会話が DEALLOCATED TYPE(ABEND_SVC) になります。元のトランザクションが LU 6.2 装置から入力

されたものであって、IMS が APPC/MVS からメッセージを受け取った場合、廃棄可能なトランザクションは、廃棄不能トランザクションのような延期キューに入れられるのではなく、廃棄されます。

この形式の ROLS 呼び出しのパラメーターは以下のようになります。

- 呼び出し機能 ROLS
- 状況コード BA または BB を受け取った DB PCB の名前

これらのいずれのパラメーターの場合でも、ROLS 呼び出しを出すと、U3303 異常終了が発生し、制御はアプリケーション・プログラムに戻されません。IMS は、今後の処理のために入力メッセージを保管します。

関連概念:

APPC/IMS および LU 6.2 装置の管理 (コミュニケーションおよびコネクション)

関連資料:

プログラム仕様ブロック (PSB) 生成ユーティリティー (システム・ユーティリティー)

ROLB 呼び出し (アプリケーション・プログラミング API)

中間バックアウト・ポイントへのバックアウト: SETS、SETU および ROLS

ROLS 呼び出しを使用すると、前の SETS または SETU 呼び出しで設定された中間バックアウト・ポイントにバックアウトすることも、前のコミット・ポイントにバックアウトすることもできます。

中間ポイントにバックアウトする ROLS 呼び出しがバックアウトするのは、DL/I の変更だけです。このタイプの ROLS 呼び出しは、CICS ファイル制御または CICS 一時データを使用する CICS 変更に影響を与えません。

SETS 呼び出しおよび ROLS 呼び出しは、アプリケーション・プログラムの呼び出し処理に中間バックアウト・ポイントを設定し、これらのいずれかのポイントにデータベース変更をバックアウトします。最大 9 つの中間バックアウト・ポイントを設定することができます。SETS 呼び出しは、各ポイントにトークンを指定します。そして、IMS はこのトークンを現行の処理ポイントに関連付けます。その後で、同じトークンを使用して ROLS 呼び出しを出すと、そのトークンを使用した SETS 呼び出しの後で行われたすべてのデータベース変更がバックアウトされ、非至急メッセージが廃棄されます。次の図は、SETS 呼び出しと ROLS 呼び出しが連携して機能する仕組みを示しています。

また、アプリケーション・プログラムが ROLS 呼び出しの後で再設定したい他の変数を管理できるように、SETS 呼び出しの入出力域にユーザー・データを含めることもできます。 このデータは、同じトークンを指定した ROLS 呼び出しが出されたときに戻されます。

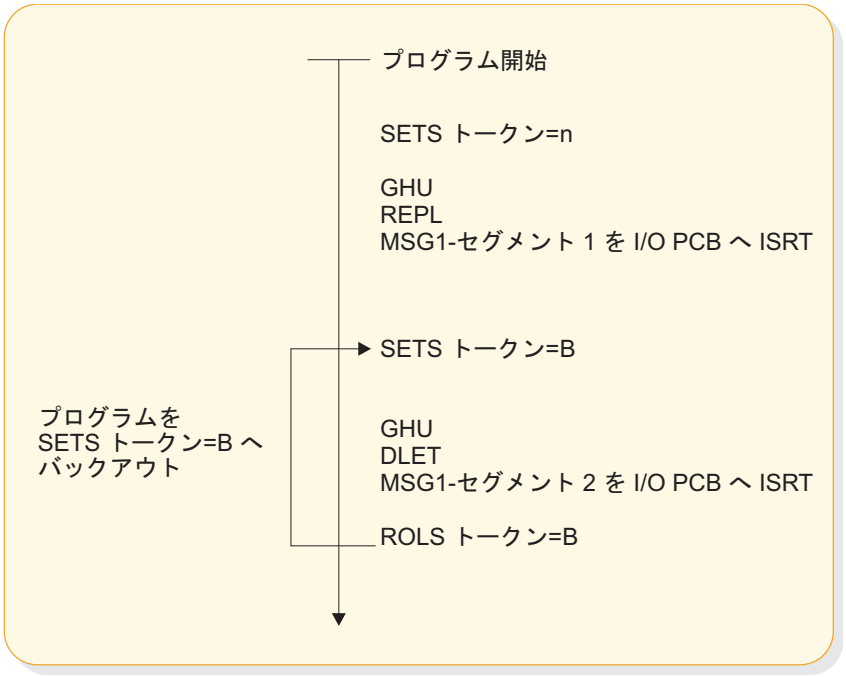


図 *59. SETS* 呼び出しと *ROLS* 呼び出しの連係動作

SETS 呼び出しと SETU 呼び出し

SETS 呼び出しは、9 個までの中間バックアウト・ポイントを設定するか、既存のすべてのバックアウト・ポイントを取り消します。 SETS 呼び出しを使用すると、作業の一部をバックアウトできます。作業の一部を完了するために必要なデータが使用不能である場合には、作業の別の部分を完了してから前の部分に戻ることができます。

中間バックアウト・ポイントを設定するには、I/O PCB を使用して入出力域とトークンを指定した呼び出しを発行してください。入出力域は、 *LLZZ* ユーザー・データ を含む形式になっています。 *LL* は入出力域内のデータの長さを表します (*LLZZ* 部分の長さも含みます)。*ZZ* フィールドには、常に 2 進数のゼロが入ります。入出力域内のデータは、関連する ROLS 呼び出しでアプリケーション・プログラムに返されます。ROLS 呼び出しで戻されるデータの一部を保管する必要がない場合には、入出力域の長さを定義する *LL* を 4 に設定してください。

PLITDLI の場合には、他の言語の場合と異なり、 *LL* フィールドをハーフワードとしてではなく、フルワードとして定義しなければなりません。 PLITDLI の場合の *LL* フィールドの内容は、 *LLZZ* 形式を使用するその他の呼び出しの入出力域と整合性をもちます。この内容は、4 バイトの *LL* フィールドの長さを含む区域の合計長から 2 を引いた値です。

現行の処理ポイントに関連する 4 バイトのトークンも必要です。このトークンは、このプログラム実行のための新しいトークンであっても、前の SETS 呼び出しで出されたトークンと一致するものであってもかまいません。トークンが新しい場合は、先行する SETS 呼び出しは取り消されません。トークンが先行する SETS 呼び出しのトークンと一致する場合、現行の SETS 呼び出しは先行する呼び出しの位置

を使用します。この場合、一致するトークンを持つ SETS 呼び出し以降に出されたすべての SETS 呼び出しが取り消されます。

この形式の SETS 呼び出しのパラメーターは、以下のようになります。

- 呼び出し機能 SETS
- I/O PCB または AIB の名前
- ユーザー・データが入っている入出力域の名前
- トークンが入っている区域の名前

SETS 呼び出しの形式については、「*IMS V15* アプリケーション・プログラミング *API*」のトピック『SETS/SETU 呼び出し』を参照してください。

前のバックアウト・ポイントをすべて取り消すときは、I/O PCB を使用して呼び出しを発行しますが、入出力域またはトークンは指定しません。呼び出しに入出力域が含まれていない場合、前の SETS 呼び出しで設定された中間バックアウト・ポイントはすべて取り消されます。

この形式の SETS 呼び出しのパラメーターは、以下のようになります。

- 呼び出し機能 SETS
- I/O PCB または AIB の名前

コミットされたデータをバックアウトすることはできないため、コミット・ポイント処理を行うと、未解決の SETS はすべて取り消されます。

DEDB、MSDB、および GSAM の各編成の PCB が PSB にある場合、または接続されたサブシステムにプログラムがアクセスする場合には、部分的なバックアウトはできません。この場合、SETS 呼び出しは拒否され、SC 状況コードが戻されます。代わりに SETU 呼び出しを使用すると、PCB がサポートされないという理由で拒否されることはありませんが、サポートされない PCB が PSB に含まれていること、およびこの機能がこれらのサポートされない PCB には適用できないことを警告するために、SC 状況コードが戻されます。

関連資料: SETS 呼び出しの後で返される状況コード、およびそれらの状況コードと必要な応答に関する説明については、「*IMS V15* アプリケーション・プログラミング *API*」を参照してください。

ROLS

ROLS 呼び出しは、直前の SETS または SETU 呼び出しで設定された処理ポイント、あるいは直前のコミット・ポイントまでのデータベース変更をバックアウトします。その後、ROLS 呼び出しは処理済みの入力メッセージをメッセージ・キューに戻します。

前の SETS 呼び出し以降に行われたデータベース変更とメッセージ・アクティビティーをバックアウトするためには、I/O PCB を使用し、ROLS 呼び出しを発行し、その呼び出しの中で入出力域とトークンを指定してください。このトークンが前の SETS 呼び出しで設定されたトークンと一致しない場合には、エラー状況が戻されます。このトークンが前の SETS 呼び出しのトークンと一致する場合には、対応する SETS 呼び出し以降に行われたデータベース変更はバックアウトされ、対応する SETS 呼び出し以降に挿入された非至急メッセージはすべて廃棄されます。バックア

ウトされた処理の一部として出された SETS は、取り消されます。サポートされるすべての PCB に関する既存のデータベース位置がリセットされます。

SETU 呼び出しに対応して ROLS 呼び出しが出されたときに、サポートされない PCB (DEDB、MSDB、または GSAM) が PSB に含まれていると、PCB の位置は影響を受けません。 ROLS 呼び出しによって指定されるトークンは、SETS または SETU 呼び出しのいずれかで設定することができます。非サポート PCB が PSB に含まれず、しかもプログラムが付加サブシステムを使用していない場合に、ROLS 呼び出しの機能は、トークンが SETS または SETU のいずれかの呼び出しで設定されているかにかかわらず同じになります。

SETS 呼び出しに ROLS 呼び出しが応答する場合で、さらに、非サポート PCB が PSB に含まれているか、または先行する SETS 呼び出しが出された時にプログラムが付加サブシステムを使用していた場合には、SETS 呼び出しは受け入れられず、SC 状況コードが戻されます。引き続いて出された ROLS 呼び出しは、RC 状況コードで拒否されるか (この状況コードは、非サポートオプションがあることを示します)、あるいは RA 状況コードで拒否されます (この状況コードは、前に成功した SETS 呼び出しで設定されたトークンと一致するトークンがないことを示します)。

SETU 呼び出しに ROLS 呼び出しが応答する場合、その呼び出しにサポートされないオプションがあることを理由に拒否されることはありません。非サポート PCB が PSB に含まれていても、ROLS 呼び出しの状況コードにはこのことは反映されません。プログラムが付加サブシステムを使用している場合には、ROLS 呼び出しは処理されますが、付加サブシステムを使用して変更を行った場合には、その変更がバックアウトされていないことを警告するために、RC 状況コードが戻されます。

この形式の ROLS 呼び出しのパラメーターは以下のようになります。

- 呼び出し機能 ROLS
- I/O PCB または AIB の名前
- ユーザー・データを受け取る入出力域の名前
- 4 バイトのトークンが入っている区域の名前

関連資料**:** ROLS 呼び出しの後に返される状況コード、およびそれらの状況コードと必要な応答に関する説明については、「*IMS V15* メッセージおよびコード 第 *4* 巻 *: IMS* コンポーネント・コード」を参照してください。

関連概念:

514 ページの『前のコミット・ポイントへのバックアウト: ROLL、ROLB、および ROLS 呼び出し』

ユーザー・プログラムの排他使用に対するセグメントの予約

ユーザーがセグメントを予約して、それを使用している間、他のプログラムによって更新されないようにしたい場合があります。IMS では、リソース・ロック管理を介してある程度までこれを実行できます。Q コマンド・コードを使用すると、これとは別の方法でセグメントを予約できます。

制約事項**:** Q コマンド・コードは、MSDB 編成、あるいはデータベースとして処理される副次索引についてはサポートされません。

リソース・ロック管理と Q コマンド・コードはともに、ユーザー・プログラム用にセグメントを予約する点では同じですが、その働き方は異なり、また相互に独立して機能します。Q コマンド・コードおよび DEQ 呼び出しの使用法および使用目的を理解するには、リソース・ロック管理について理解しておく必要があります。

リソース・ロック管理の機能は、あるプログラムが変更したデータを、その変更を行ったプログラムがコミット・ポイントに達するまで、別のプログラムからアクセスされないようにすることです。したがって、ユーザーがあるセグメントを変更した場合、ユーザー・プログラムがコミット・ポイントに達するまで、別のプログラム (ただし、GO 処理オプションを使用するものは除きます) がそのセグメントにアクセスすることはできません。Q コマンド・コードをサポートするデータベース編成の場合、PCB 処理オプションによって更新が許容されていて、PCB がデータベース・レコード内に位置を保持しているときには、他のプログラムはそのデータベース・レコードにアクセスできません。

Q コマンド・コードを使用すると、セグメントにアクセスした PCB が別のデータベース・レコードに移動しても、ユーザーがアクセスしたセグメントを、他のプログラムが更新することはできなくなります。

関連資料: Q コマンド・コードについて詳しくは、「*IMS V15* アプリケーション・プログラミング *API*」のトピック『Q コマンド・コード』を参照してください。

第 18 章 副次索引付けおよび論理関係

副次索引と論理関係を使用すると、アプリケーション・プログラムのデータ・ビューを変更できます。これらのオプションを使用するかどうかは、DBA が決定します。

これらの技法を使用するための方法を以下の例で示します。

- アプリケーション・プログラムがキー・フィールドで指定された以外の順序でセグメント・タイプにアクセスする必要がある場合には、副次索引を使用できます。副次索引を使用すると、従属セグメントで指定されている条件にもとづいた、アプリケーション・プログラムのデータ・アクセスまたはデータ視点を変更することもできます。
- アプリケーション・プログラムが、異なるデータベースからのセグメントを収める論理構造を必要とする場合は、論理関係が使用されます。

関連概念:

210 ページの『SSA のガイドライン』

副次索引がユーザー・プログラムに与える影響

副次索引を使用する例として、アプリケーション・プログラムがルート・キーにより定義されるもの以外の順序でデータベース・レコードを選択する必要がある場合が挙げられます。

IMS はルート・セグメントをキー・フィールドの順序で保管します。キー・フィールドの順序と異なる順序でルート・セグメントにアクセスするプログラムは、効率よく作動しません。

ユーザーは、データベースの DBD 内にあるフィールドに関する XDFLD ステートメントを定義することにより、セグメント内の任意のフィールドに索引を付けることができます。Get 呼び出しが、このキーでは修飾されずに別のフィールドを使用する場合、IMS は、正しいレコードを検出するために、すべてのデータベース・レコードを検索しなければなりません。副次索引を使用すると、IMS は、キー・フィールドにないフィールド値に基づいて直接レコードを検索できます。

副次索引の詳細および例については、「*IMS V15* データベース管理」を参照してください。

副次索引を使用した SSA

ユーザーのプログラムが副次索引を使用している場合、SSA に索引付きフィールドの名前を使用できます。これを使用した場合、IMS は副次索引に直接アクセスし、指定した値のポインター・セグメントを検出します。その後、IMS は、索引セグメントが指し示すセグメントを基本データベース内に見つけ、そのセグメントをユーザーのプログラムに戻します。

SSA で索引付きフィールド名を使用するためには、以下のガイドラインに従ってください。

- 索引付きフィールドは、基本データベースに関する DBD 内で定義しなければなりません。これは、その DBD の生成時に XDFLD ステートメントを使用して定義します。
- XDFLD ステートメントで指定した名前を、修飾ステートメントのフィールド名として使用してください。
- PSB 生成時に、処理順序として副次索引を指定してください。これを行うには、PSB の生成時に、全機能副次索引データベースの PROCSEQ パラメーターまたは高速機能副次索引データベースの PROCSEQD パラメーターで、副次索引データベースの名前を指定します。

(REPL 呼び出しを使用して) 索引付きセグメントの XDFLD を変更すると、その REPL 呼び出しを出す前にユーザーが設定した親子関係はすべて失われます。REPL 呼び出しが成功すると、キー・フィードバック域は無効になります。

例えば、NAME フィールドで PATIENT セグメントを索引付けするためには、医療データベースについての DBD の XDFLD ステートメントでそのセグメントを定義しておかなければなりません。副次索引データベースの名前が INDEX である場合には、PCB で PROCSEQ=INDEX を指定してください。 PATNO フィールドではなく NAME フィールドによって PATIENT を識別する修飾を出す場合には、XDFLD ステートメントで指定した名前を使用してください。XDFLD の名前が XNAME である場合には、次のように SSA で XNAME を使用してください。

DBD の中**:**

```
XDFLD NAME=XNAME
```

PSB の中**:**

```
PROCSEQ=INDEX (全機能副次索引データベースの場合) または
PROCSEQD=INDEX (高速機能副次索引データベースの場合)
```

プログラムの中**:**

```
GU PATIENTb(XNAMEbbb=bJBBROKEbbb)
```

PROCSEQD= パラメーターを指定した PCB を使用して、副次索引により基本 DEDB データベースにアクセスする場合、「ターゲット = ルート・セグメント」の主キー・フィールドを使用した SSA の修飾 GU/GN セグメント名がサポートされています。

「ターゲット = 従属セグメント」の主キー・フィールドを使用した SSA の修飾 GU/GN セグメント名はサポートされていません。PROCSEQD= パラメーターを指定した PCB を使用して副次索引により基本 DEDB データベースにアクセスした場合、修飾 GET 呼び出しに対して AC 状況コードが返されます。

副次索引を使用した複数の修飾ステートメント

索引付きフィールドの名前を使用して呼び出しを修飾する場合には、複数修飾ステートメントを含むことができます。

修飾ステートメントを結合するために使用できる AND 演算子は 2 つあります。

* または **&**

副次索引で使用する場合には、この AND は従属 AND と呼ばれます。呼び出しの条件を満たすために、IMS は索引を 1 回スキャンし、両方の修飾ステートメントの条件を満たす 1 つのポインター・セグメントを検索します。

これは、独立 AND と呼ばれます。この演算子は副次索引でのみ使用してください。呼び出しの条件を満たすために独立 AND を使用した場合、IMS は索引を 2 回スキャンし、同じターゲット・セグメントを指し示す複数の異なるポインター・セグメントを検索します。

2 種類の AND の区別は、すべての修飾の中で索引付きフィールド (DBD で XDFLD として定義されたフィールド) が使用された場合にのみ行われます。修飾ステートメントのうちの 1 つが別のフィールドを使用しているときには、どちらの AND も従属 AND のように機能します。

次の 2 つのセクションでは、従属 AND と独立 AND の例を示します。例の中では SSA 内の 2 つの修飾ステートメントだけしか示していませんが、3 つ以上の修飾ステートメントを使用することもできます。 1 つの SSA に含むことのできる修飾ステートメントの数に制限がありませんが、SSA の最大サイズには制限があります。このサイズは、PSBGEN ステートメントの SSASIZE パラメーターで指定します。このパラメーターについては、「*IMS V15* システム・ユーティリティー」を参照してください。

従属 AND

従属 AND を使用すると、IMS は索引を 1 回だけスキャンします。この呼び出しの条件を満たすには、IMS は、両方の修飾ステートメントの条件を満たす **1** つのポインター・セグメントを検出しなければなりません。

例えば、請求金額が $500 から $1000 までの患者をリストしたいものとします。そのためには、BILLING セグメントに基づいて PATIENT セグメントを検索し、副次索引を処理順序として使用するように IMS に指示します。次の図には 3 つの副次索引セグメントが示されています。

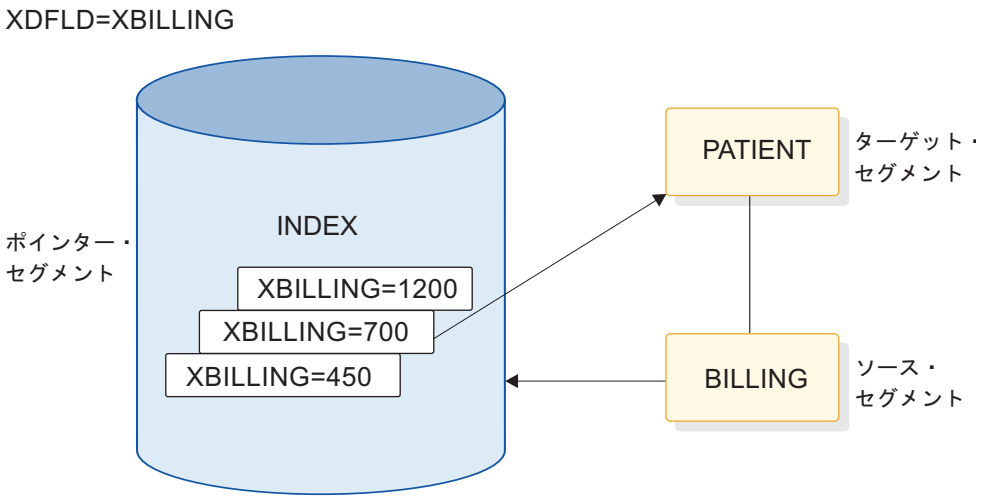


図 *60.* 従属 *AND* の使用例

この場合、次の呼び出しを使用できます。

```
GU   PATIENT (XBILLING>=00500*XBILLING<=01000)
```

この呼び出しの条件を満たすために、IMS は 500 から 1000 までの間の値をもつ 1 つのポインター・セグメントを検索します。 IMS はそのセグメントで指し示された PATIENT セグメントを戻します。

独立 AND

例えば、扁桃腺炎と咽喉炎の両方にかかっている患者のリストを必要とするものとします。この情報を得るためには、ILLNESS セグメントの ILLNAME フィールドに基づいて PATIENT セグメントを索引し、副次索引を順序処理として使用するように IMS に指示します。この例では、従属セグメント (ILLNESS セグメント) の修飾に基づいて PARENT セグメントをリトリーブします。次の図には 4 つの副次索引セグメントが示されています。

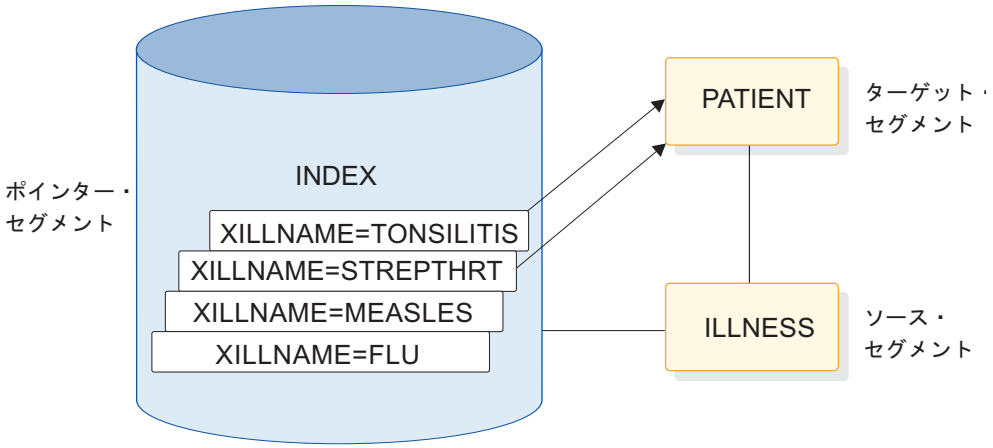


図 *61.* 独立 *AND* の使用例

IMS に、同じ PATIENT セグメントを指す 2 つのポインター・セグメントを索引から検出させる必要があります。 1 つは TONSILLITIS に等しい ILLNAME をも

つセグメントで、もう 1 つは STREPTHRT に等しい ILLNAME をもつセグメントです。次の呼び出しを使用します。次の呼び出しを使用します。

```
GU   PATIENTb(XILLNAME=TONSILITIS#XILLNAME=bSTREPTHRT)
```

この呼び出しは、ILLNESS セグメントの値が咽頭炎と扁桃腺炎になっている、最初の PATIENT セグメントをリトリーブします。この呼び出しを出すと、IMS は扁桃腺炎の索引項目を検索します。そして、同じ PATIENT セグメントを指し示す咽頭炎の索引項目を検索します。

GN および GNP 呼び出しで独立 AND を使用すると、特殊な状況が起こることがあります。同じ修飾を使用して GN または GNP 呼び出しを繰り返すと、IMS が同じセグメントをユーザーのプログラムに 2 回以上戻すことがあります。ユーザーは、キー・フィードバック域を調べることによりIMS が既にセグメントを戻したかどうかを判別できます。

未検出 (GE) 状況コードを受け取るまで GN 呼び出しを出し続けると、IMS は、それぞれの独立 AND グループごとに 1 回ずつセグメント・オカレンスを戻します。 既に戻されているセグメントと同じセグメントが IMS によって戻されたときには、PCB キー・フィードバック域は異なっています。

関連概念:

211 ページの『複数の修飾ステートメント』

副次索引に関する DL/I 戻り

「ポインター・セグメントのキー」という用語は、アプリケーション・プログラムによって認識されているキーのことを指しています。すなわち、このキーには後続フィールドが含まれていません。 IMS はこのキーを、副次索引を使用しなかった場合にルート・キーが置かれている位置 — キー・フィードバック域の左端のバイトに入れます。

IMS がアプリケーション・プログラムの入出力域に戻す PATIENT セグメントは、副次索引を使用しなかった場合とまったく同じです。しかし、キー・フィードバック域の内容は若干異なります。IMS が戻す連結キーは、ユーザーが要求したセグメントのキー (PATIENT セグメントのキー) を戻す代わりに IMS が副次索引のキー (INDEX データベース内のセグメントのキー) の検索部分を戻すことを除いて、同じです。

副次索引に既に反映されているソース・フィールドと重複する副次索引ソース・フィールドが入っているセグメントの挿入または置き換えを試みると、IMS は NI 状況コードを戻します。 NI 状況コードは、直接アクセス記憶装置に記録するバッチ・プログラムの場合にのみ戻されます。それ以外の場合、アプリケーション・プログラムは異常終了します。重複する索引ソース・フィールドがないことを確認することにより、ユーザー・プログラムの終了を回避できます。セグメントを挿入する前に、副次索引ソース・フィールドを修飾として使用し、そのセグメントをリトリーブしてみてください。

副次索引に関する状況コード

あるセグメントに副次索引が定義されていて、その定義で副次索引にユニーク・キーが指定されている場合 (ほとんどの副次索引では重複キーを使用できます)、アプリケーション・プログラムは、通常の状況コードの他に、NI 状況コードを受け取ることがあります。

この状況コードは、副次索引を処理列として使用する PCB の場合にも、また使用しない PCB の場合にも、戻されることがあります。 NI 状況コードの詳細については、「*IMS V15* メッセージおよびコード 第 *4* 巻*: IMS* コンポーネント・コード」を参照してください。

論理関係にあるセグメントの処理

アプリケーション・プログラムは、複数の別個のデータベース階層に既に存在しているセグメントからなる階層を処理しなければならないことがあります。論理関係を使用すると、これらのセグメント間の階層関係を設定できます。論理関係を使用した場合には新しい階層が作成されます。この階層は物理ストレージに存在しているものではありませんが、アプリケーション・プログラムは、これが物理ストレージに存在している場合と同様に処理できます。このタイプの階層は論理構造と呼ばれます。

論理関係を使用する利点は、実際には 1 個所にだけ保管されているデータに対して、複数の階層に存在するときと同じようにプログラムがアクセスできることです。2 つのアプリケーション・プログラムが異なるパスを介して同じセグメントにアクセスする必要がある場合には、論理関係を使用するもう 1 つの方法としてそのセグメントを両方の階層に保管する方法があります。この方法の問題点は、カレント・データを保持するために、2 個所でデータ更新を行わなければならないことです。

論理関係にあるセグメントの処理は、他のセグメントの処理とあまり変わりません。次の例は、在庫アプリケーション・プログラム用のシナリオからの抜粋です。このアプリケーション・プログラムは、購買データベース内のデータを処理しながら、患者データベース内のセグメントにもアクセスする必要があります。

例えば、在庫アプリケーション・プログラムが処理のために必要とする階層には、次の 4 つのセグメント・タイプが含まれています。

- 病院で使用されている薬剤の名前と識別番号が入っている ITEM セグメント
- その薬剤を納入している業者の名前と住所が入っている VENDOR セグメント
- その病院に納入されるそれぞれの薬剤の納入量や納入日などの情報が入っている SHIPMENT セグメント
- 病院におけるその薬剤の消費に関する、使用量、日付、および処方した医師などの情報が入っている DISBURSE セグメント

医療データベースの TREATMNT セグメントは、在庫アプリケーション・プログラムが DISBURSE セグメントでの処理に必要とするものと同じ情報を含んでいます。この情報を両方の階層で保管する代わりに、TREATMNT セグメントに情報を保管し、ITEM 階層内の DISBURSE セグメントと PATIENT 階層内の TREATMNT セグメントとの論理関係を定義できます。これにより、ITEM 階層を

介して TREATMNT セグメントを SHIPMENT セグメントの子セグメントのように処理できます。この場合、DISBURSE は 2 つの親を持つことになります。SHIPMENT は DISBURSE の物理親で、TREATMNT は DISBURSE の論理親です。

この論理関係には、DISBURSE、SHIPMENT、および TREATMNT の 3 つのセグメントが含まれます。次の図の右の部分には ITEM 階層が示されています。DISBURSE セグメントは、左側の PATIENT 階層の TREATMNT セグメントを指し示します。(PATIENT 階層は医療データベースの一部です。)

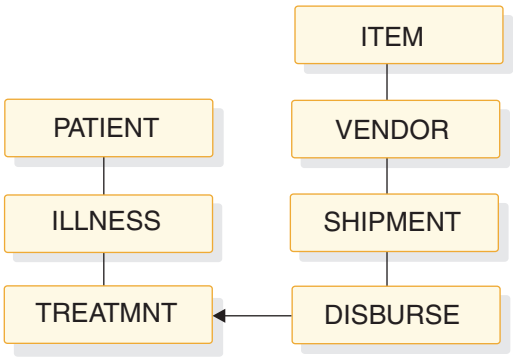


図 *62. PATIENT および ITEM 階層*

論理関係には、次の 3 つのタイプのセグメントが含まれています。

- TREATMNT は論理親セグメントと呼ばれます。これは ILLNESS の物理従属セグメントですが、論理子セグメント DISBURSE によってパスが設定されているため、ITEM 階層を介して処理できます。論理親セグメントは、両方の階層を介してアクセスすることができますが、1 個所だけにしか保管されません。
- SHIPMENT は、物理親セグメントと呼ばれます。物理親は、物理データベース階層における論理子セグメントの親です。
- DISBURSE は論理子セグメントと呼ばれます。これは、ITEM 階層の中の SHIPMENT セグメントから PATIENT 階層の TREATMNT セグメントまでのパスを確立します。

論理子セグメントは論理親を指し示すため、プログラムが論理親セグメントにアクセスするためには、2 つのパスを使用できます。

- プログラムが物理パスを介して論理親セグメントにアクセスする時には、そのセグメントの物理親を介して論理親に到達します。ILLNESS を介して TREATMNT セグメントにアクセスする時には、物理パスを介しての論理親へのアクセスとなります。
- プログラムが論理パスを介して論理親セグメントにアクセスする時には、そのセグメントの論理子を介して論理親に到達します。SHIPMENT を介して TREATMNT セグメントにアクセスする時には、論理パスを介しての論理親へのアクセスとなります。

論理親セグメントがその論理子セグメントを介してアクセスされる場合は、論理子セグメントは次のように、論理親セグメントからのデータと、単一セグメント入出力域の中でユーザーがこのペアリング (交差データ) に関連付けた任意のデータの両

方に連結されます。

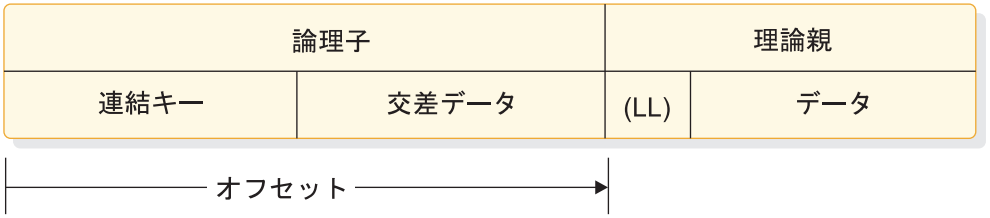


図 63. 連結セグメント

LL は、論理親が可変長セグメントである場合の長さフィールドです。

論理関係がプログラミングに与える影響

論理関係にあるセグメントを処理するためにユーザーが出す呼び出しは、他のセグメントを処理するために使用する呼び出しと同じです。ただし、次の点で、処理方法が異なります。すなわち、ユーザーの入出力域での論理セグメントの外観、リトリーブ呼び出し後の DB PCB マスクの内容、物理親セグメントおよび論理親セグメントの置き換え、削除、および挿入を行うための方法が異なります。

論理関係にあるセグメントには論理パスからでも物理パスからでもアクセスできるため、未許可プログラムによってセグメントが更新されないように保護する必要があります。

DBA が論理関係を定義する時には、セグメントの削除、置き換え、および挿入がどのようにして行えるかを決める一連の規則を定義します。 これらの規則の定義は、データベース設計の決定に属します。ユーザーのプログラムが論理関係にあるセグメントを処理する場合には、DBA (またはインストール先におけるデータベース設計の責任者) から以下の情報を得る必要があります。

- ユーザーがセグメントをリトリーブするときに、入出力域の中でセグメントがどのようになっているか
- ユーザーのプログラムが、セグメントの更新および挿入を行うことができるか
- DX、IX、または RX 状況コードを受け取ったときに何を行うべきか

論理子セグメントの挿入の要件は、以下のとおりです。

- ロード・モードでは、論理子セグメントは物理親のもとでのみ挿入できます。論理親は、入出力域では提供されません。
- 更新モードでは、論理子セグメントの形式は、物理親または論理親のどちらを介してアクセスするかによって異なります。
  - 物理親からアクセスする場合、論理子セグメントの形式は、論理親の連結キーの後に交差データが続いたものになります。
  - 論理親からアクセスする場合、論理子セグメントの形式は、物理親の連結キーの後に交差データが続いたものになります。
- 論理子は、論理親または物理親の挿入規則にもとづいて挿入または置き換えできます。論理または物理親の挿入規則が PHYSICAL でない限り、論理親または物理親は、入出力域の中で論理子セグメントの後に指定しなければなりません。

関連概念:

211 ページの『複数の修飾ステートメント』

論理関係に関する状況コード

論理関係にあるセグメントに対して特に適用される状況コードとして、以下のものがあります。

論理子セグメントまたは物理あるいは論理親を処理する際に受け取る可能性のある状況コードは、これだけではありません。これらの状況コードを受け取った場合、許されない方法でデータベースの更新を試みたことを意味します。問題を判別するためには、DBA またはインストール先における論理関係の決定責任者に連絡してください。

DX 物理削除規則に違反したため、IMS がセグメントを削除しませんでした。セグメントが論理親セグメントである場合、まだアクティブな論理子セグメントがあります。セグメントが論理子セグメントである場合、論理パス経由では削除されていません。

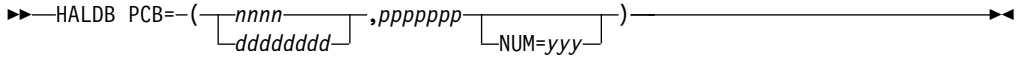
IX 論理子セグメントまたは連結セグメントを挿入しようとしました。セグメントが論理子セグメントである場合、対応する論理または物理親セグメントが存在していません。このセグメントが連結セグメントである場合、挿入規則が物理的であるにもかかわらず論理または物理親が存在していないか、あるいは挿入規則が仮想であるにもかかわらず入出力域内の論理または物理親のキーがその論理または物理親の連結キーに一致していません。

RX 物理置き換え規則に違反しています。目標親に物理置き換え規則が指定されているときに、そのデータを変更しようとしました。物理置き換え規則が適用されている宛先親は、物理パスを経由してでなければ置き換えることはできません。

第 19 章 HALDB 区画選択処理

DL/I 呼び出しの処理を単一 HALDB 区画あるいは HALDB 区画の範囲に制限することができます。このためには、制御ステートメントを渡すために DD 名 DFSHALDB を指定した DD ステートメントを使用します。DFS HALDB は、バッチ・ジョブ、BMP (バッチ・メッセージ処理従属オンライン領域)、または JBP (Java バッチ処理従属オンライン領域) の JCL で供給する必要があります。

HALDB 区画選択処理のための制御ステートメント



各 HALDB 制御ステートメントには、必要パラメーターが指定された PCB キーワードを組み込む必要があります。それぞれの制御ステートメントに必要なパラメーターは 1 行ごとに入力し、同一行に継続して入力することはできません。複数の HALDB 制御ステートメントを入力できます。DB PCB 番号を重複させないでください。重複すると、最後に読み取られた制御ステートメントによって、前のステートメントがオーバーライドされます。

HALDB 制御ステートメントが構文的に正しければ、テーブルに項目として記載されます。テーブル内の最大項目数は 20 です。それ以降読み込まれるすべてのステートメントは、既にテーブルに記載された項目の重複ステートメントでない限りは構文的に正しくても無視され、U0201 異常終了となります。

HALDB 区画選択処理のためのパラメーターの記述

nnnn
PSB で定義した DB PCB の相対番号としての DB PCB 番号。

dddddddd
DB PCB のラベルまたは名前。

ppppppp
区画名。このパラメーターは必須です。

NUM=*yyy*
この PCB の使用に限定されている、指定された区画から始まる連続した区画の範囲。連続区画の範囲は、区画選択の順序として定義され、これは DFSHALDB ステートメントで指定された宛先区画から始まる、次の選択された区画です。次の区画は、HALDB のために定義されたハイ・キー、または区画選択出口によって定義された処理順序を使って決定されます。このパラメーターはオプションです。

以下の例では、HALDB 区画選択処理ステートメントの使用方法を示しています。

単一区画制限のための DFSHALDB

```
HALDB PCB=(4,POHIDKA)
HALDB PCB=(PCBNUM2,POHIDJA)
```

範囲区画制限のための DFSHALDB

```
HALDB PCB=(3,PVHDJ5A,NUM=4)
HALDB PCB=(PCBNUM7,PVHDJ5B,NUM=3)
```

HALDB 区画選択処理で生成される報告書

HALDB 区画選択処理を使用すると、SYSHALDB データ・セット内に「HALDB Selective Partition Processing」という報告書が生成されます。この報告書には、発行された制御ステートメントと、各ステートメントの受諾もしくはリジェクトの理由が示されています。妥当性検査され、受け入れられた制御ステートメントについては、「Syntactically correct」と表示されます。次の表では、構文的に正しいステートメントに対して出される可能性があるその他のメッセージ、およびそれに付随するメッセージを示しています。

表 *43. HALDB* 区画選択処理で生成された報告書にあるメッセージ

メッセージ	説明
Duplicate, overrides previous statement	同じ PCB に対する HALDB ステートメントは既に検出されました。 現行のステートメントは、前の HALDB ステートメントをオーバーライドします。
Ignored, number of valid statements exceeds 20	20 ステートメントしか許されていないのに、20 を超える HALDB ステートメントがありました。HALDB ステートメントの数を 20 以下に減らしてから、ジョブを再実行してください。このメッセージが表示された場合、U0201 で異常終了します。
NUM parameter must be non-zero numeric	NUM キーワードで指定する区画範囲は、ゼロでない 1 から 999 までの値でなければなりません。
NUM value exceeds three digits	NUM キーワードで指定する区画範囲は、ゼロでない 1 から 999 までの値でなければなりません。
An equal sign must follow NUM keyword	HALDB ステートメントで、NUM キーワードの後には等号が続かなければなりません。HALDB ステートメントに等号を追加してください。
The NUM keyword is missing	区画名の後にコンマが検出されましたが、NUM キーワードは存在しませんでした。HALDB ステートメントの定位置パラメーターの構文を検査するか、あるいは制限のための NUM キーワードと区画範囲を追加するか、してください。
NUM parameter is missing	NUM キーワードが検出されましたが、NUM パラメーター値は存在しませんでした。HALDB ステートメントの定位置パラメーターの構文を検査するか、あるいは制限のための NUM キーワードと区画範囲を追加するか、してください。

構文的に正しくない HALDB 制御ステートメント (処理されてリジェクトされたステートメント) については、発行されるメッセージと説明を次の表に示しています。

表 *44.* 構文が不適切な *HALDB* ステートメントで生成された報告書内のメッセージ

メッセージ	説明
No HALDB statement type	DFSHALDB データ・セットは HALDB ステートメントを含んでいません。このエラーを回避するために HALDB ステートメントを追加してください。
A space must follow HALDB statement type	HALDB ステートメントでは HALDB の後と PCB キーワードの前にスペースが必要です。
PCB keyword missing	必要なキーワード PCB が見つかりませんでした。 HALDB ステートメントを正しく処理するには PCB キーワードが必要です。
Equal sign must follow PCB keyword	PCB キーワードに続く等号がありません。HALDB ステートメントを正しく処理するには、PCB キーワードに続いて等号が必要です。
Open parenthesis must follow equal sign	PCB= の後に左括弧 "(" がありません。HALDB ステートメントを正しく処理するには、PCB= に続いて左括弧 "(" が必要です。
Second parameter may be missing	HALDB 区画を指定する必要があります。区画名を追加するか、あるいは定位置パラメーターの構文が正しいことを検証してください。
First parameter exceeds four digits	DB PCB 番号は 4 桁の値を超えることができません。DB PCB 番号を正しいものに変更してください。
Delimiter is not a comma	パラメーター値間のコンマが欠落しています。コンマは定位置パラメーターの区切り文字として使用します。コンマを追加するか、あるいは定位置パラメーターの構文が正しいことを検証してください。
Partition name must start with an alpha	HALDB 区画名は英字で始まる必要があります。区画名を追加するか、あるいは定位置パラメーターの構文が正しいことを検証してください。
Delimiter is not a close parenthesis	HALDB ステートメントで、右括弧 ")" が欠落しています。 PCB パラメーターを括るよう右括弧 ")" を追加してください。
Partition name exceeds seven characters	HALDB 区画名は 7 文字以下でなければなりません。区画名を追加するか、あるいは定位置パラメーターの構文が正しいことを検証してください。

表 44. 構文が不適切な *HALDB* ステートメントで生成された報告書内のメッセージ (続き)

メッセージ	説明
Invalid character in partition name	HALDB 区画名に無効文字が含まれています。区画名を追加するか、あるいは定位置パラメーターの構文が正しいことを検証してください。
Statement contains all spaces	HALDB ステートメントが欠落しています。有効な HALDB ステートメントを追加してください。
Invalid statement input	HALDB ステートメントが見つかりましたが、完全なものではありません。HALDB ステートメントの構文および指定した定位置パラメーターを検査してください。
Space must follow close parenthesis	右括弧 ")" の後にスペースが必要です。右括弧 ")" の後にスペースを追加してください。
First parameter missing	PCB 番号またはラベルが欠落しています。PCB 名またはラベルを追加するか、あるいは定位置パラメーターの構文が正しいことを検証してください。
Comma and part name missing	HALDB ステートメントに PCB 番号かラベルしかありません。区画名を追加するか、あるいは定位置パラメーターの構文が正しいことを検証してください。
Partition name is missing	HALDB ステートメントに HALDB 区画名を指定する必要があります。区画名を追加するか、あるいは定位置パラメーターの構文が正しいことを検証してください。
Partition name starts with numeric	HALDB 区画名は英字で始まる必要があります。区画名を追加するか、あるいは定位置パラメーターの構文が正しいことを検証してください。
First parameter must not be zero	PCB 番号はゼロ以外の番号でなければなりません。DB PCB 番号にゼロ以外の番号を追加してください。
Comment statement	HALDB ステートメントの 1 桁目にアスタリスクがあります。このステートメントはコメントとみなされ、スキップされます。

すべてのステートメントの検証が済むと、ジョブは異常終了コード U0201 で異常終了します。

第 20 章 GSAM データベースの処理

GSAM データベースは、バッチ・メッセージ処理 (BMP) 領域、トランザクション指向 BMP、またはJava バッチ処理 (JBP) 領域でバッチ・プログラムとして実行できるアプリケーション・プログラムで使用できます。

ユーザーのアプリケーション・プログラムが GSAM データベースにアクセスする場合、プログラムの設計時に以下の点を考慮する必要があります。

- IMS プログラムは、GSAM データベースからレコードをリトリーブしたり、データベースの終わりにレコードを追加したりすることができますが、データベース内のレコードを削除したり置き換えたりすることはできません。
- 複数の GSAM データベースにアクセスするには、別々の呼び出しを使用します。(GSAM の使用に関しては、チェックポイントと再始動に関する追加の考慮事項があります。)
- ユーザーのプログラムは、GSAM を使用する場合には、シンボリック `CHKP` および `XRST` 呼び出しを使用しなければなりません。基本 `CHKP` 呼び出しでは、GSAM データベースのチェックポイントをとることはできません。
- IMS プログラムは、GSAM データ・セットを使用するときには、GSAM データ・セットを順次非階層データベースと同じように扱います。GSAM で使用可能な z/OS アクセス方式は、直接アクセス装置、ユニット・レコード装置、およびテープ装置の場合には BSAM であり、直接アクセス記憶装置の場合には VSAM です。VSAM データ・セットは、DASD 上にあるキーなし、索引なしの入力順データ・セット (ESDS) でなければなりません。VSAM は、一時ファイル、SYSIN、SYSOUT、およびユニット・レコード・ファイルをサポートしません。
- GSAM は順次非階層データベースであるため、セグメント、キー、および親子関係をもちません。

JBP 領域で実行されている Java アプリケーション・プログラムは、IMS Java 従属領域リソース・アダプターを使用することで GSAM データベースにアクセスできます。

関連概念:

278 ページの『GSAM データベースのデータ域』

関連資料:

841 ページの『JBP アプリケーションから GSAM データへのアクセス』

GSAM データベースへのアクセス

汎用順次アクセス方式 (GSAM) データベースにアクセスするために使用する呼び出しは他の IMS データベースにアクセスするために使用する呼び出しとは異なります。また、GSAM データベースを入力用にも出力用にも使用できます。

例えば、プログラムはある GSAM データベースから入力を順次に読み取り、その出力データを別の GSAM データベースにロードできます。 GSAM データベース

から入力をリトリーブするプログラムは、通常は GSAM レコードを順次にリトリーブしてからそれらを処理します。 GSAM データベースに出力を送るアプリケーションは、プログラムがレコードを処理する際、データベースの終わりに出力レコードを追加しなければなりません。 GSAM データベース内のレコードは削除または置き換えを行うことができず、レコードを追加するときにはデータベースの終わりに追加しなければなりません。

GSAM データベース用の PCB マスク

一般的に、GSAM データベースは他の IMS データベースと同じ方法で処理されます。ユーザーは DL/I 呼び出しと非常に似た呼び出しを使用して要求を伝えます。GSAM はこれらの呼び出しの結果を GSAM DB PCB に記述します。

GSAM データベースに対する呼び出しは、AIBTDLI インターフェースを使用することも、PCB インターフェースを使用することもできます。

GSAM データベース用の DB PCB マスクは、他の IMS データベース用の場合と同じ用途に使用できる。プログラムは GSAM DB PCB マスクを介して DB PCB のフィールドを参照します。この GSAM DB PCB マスクには、GSAM DB PCB と同じフィールドが含まれている必要があり、また各フィールドは GSAM DB PCB の対応フィールドと同じ長さでなければなりません。

GSAM データベース用の DB PCB と他の IMS データベース用の DB PCB との間には、いくつかの相違点があります。いくつかのフィールドが相違しており、また GSAM DB PCB には、他の PCB に含まれていないフィールドが 1 つあります。GSAM は階層データベースではないため、他の IMS データベース用の PCB マスクのフィールドの中には、GSAM PCB マスク内では意味をもたないものがあります。GSAM データベースをアクセスする場合に使用されないフィールドは、以下のとおりです。

- 2 番目のフィールド: セグメント・レベル番号
- 6 番目のフィールド: セグメント名
- 8 番目のフィールド: センシティブ・セグメントの数

GSAM がこれらのフィールドを使用することはありませんが、以下の表に示されたとおりの順序および長さで、これらのフィールドを GSAM DB PCB マスク内に定義する必要があります。

DB PCB マスク内で、フィールドをコーディングするときには、DB PCB の場合と同じように、すべてのフィールドが入っている区域に名前を付けます。入り口ステートメントは、PSB 内の PCB の順序にもとづいて、ユーザー・プログラム内の各 DB PCB マスクをユーザー・プログラムの PSB 内の DB PCB と関連付けます。入り口ステートメントはプログラム内の DB PCB マスクを、そのマスクの名前またはポインターによって参照します。

入り口ステートメントをコーディングする際に、次のことを行います。

- COBOL、Java、Pascal、C、およびアセンブラー言語プログラムで、入り口ステートメントは、プログラム内の DB PCB マスクの名前をリストしなければなりません。

- PL/I プログラムで、入り口ステートメントは、プログラム内の DB PCB マスクを指すポインターをリストしなければなりません。

入り口ステートメントの最初の PCB 名またはポインターは、最初の PCB に対応しています。入り口ステートメントの 2 番目の PCB 名またはポインターは、2 番目の PCB に対応しています (以下同様)。

表 *45. GSAM DB PCB* マスク

記述子	長さ (バイト数)	DB/DC	DBCTL	DCCTL	DB バッチ	TM バッチ
データベース名 [1]	8	X	X	X	X	X
セグメント・レベル番号 [2]	2	N/A	N/A	N/A	N/A	N/A
状況コード [3]	2	X	X	X	X	X
処理オプション [4]	4	X	X	X	X	X
IMS に予約済み[5]	4	X	X	X	X	X
セグメント名 [6]	8	N/A	N/A	N/A	N/A	N/A
キー・フィードバック域と不定長レコード域の長さ [7]	4	X	X	X	X	X
センシティブ・セグメントの数 [8]	4	N/A	N/A	N/A	N/A	N/A
キー・フィードバック域 [9]	8 または 12 (ラージ・データ・セットの場合)	X	X	X	X	X
不定長レコードの長さ[10]	4	X	X	X	X	X

注:

1. データベース名。GSAM DBD の名前です。このフィールドは 8 バイト長で、文字データが入ります。
2. セグメント・レベル番号。GSAM では使用されませんが、コード化はする必要があります。このフィールドは、2 バイト長です。
3. 状況コード。GSAM データベースに対する各呼び出しの後で、IMS はこのフィールドに 2 文字の状況コードを入れます。この状況コードは、呼び出しの結果を表します。各呼び出しが出されるたびに、IMS はこのフィールドを更新します。呼び出しと呼び出しの間、その値は消去されません。アプリケーション・プログラムは、各呼び出しの後でこのフィールドを検査して、その呼び出しが成功したかどうかを確認する必要があります。呼び出しが正常に完了した場合には、このフィールドにはブランクが入ります。
4. 処理オプション。プログラムが出すことのできる呼び出しのタイプを IMS に通知するコードが入る 4 バイト・フィールドです。このフィールドは、特定のプログラムにデータベースの読み取りだけを行わせて、そのデータベースを更新できないようにする、セキュリティー機構として機能します。この値は、アプリケーション・プログラムの PSB を生成するときに、PCB ステートメント

の PROCOPT パラメーターでコーディングされます。この値は、変更されることはありません。 GSAM の場合、この値は G、GS、L、または LS です。

5. **IMS** に予約済み。この 4 バイト・フィールドは、IMS により内部結合のために使用されます。アプリケーション・プログラムによって使用されることはありません。
6. セグメント名。このフィールドは GSAM では使用されませんが、GSAM DB PCB マスクの一部としてコーディングする必要があります。このフィールドは 8 バイト長です。
7. キー・フィードバック域と不定長レコード域の長さ。10 進値の 12 (ラージ・フォーマット・データ・セットの場合は 16) が入る 4 バイト・フィールドです。これはキー・フィードバック域と不定長レコード域の長さの合計です。
8. センシティブ・セグメントの数。このフィールドは GSAM では使用されませんが、GSAM DB PCB マスクの一部としてコーディングする必要があります。このフィールドは 4 バイト長です。
9. キー・フィードバック域。リトリーブ呼び出しが成功すると、GSAM は、プログラムに返されてきたレコードのアドレスをこのフィールドに入れます。これは、レコードの検索引数 (RSA) と呼ばれます。後で、このレコードを直接リトリーブする必要がある場合、この値を GU 呼び出しのパラメーターの 1 つとして指定できます。このフィールドは、基本形式のデータ・セットの場合は 8 バイトで、ラージ・フォーマット・データ・セットの場合は 12 バイトです。
10. 不定長レコード域。不定長レコード (RECFM=U) を使用する場合、処理するレコードの長さの 2 進表現が、このフィールドに入れられてユーザー・プログラムと GSAM との間で受け渡されます。このフィールドは 4 バイト長です。GU または GN 呼び出しを出すと、GSAM は、リトリーブされたレコードの長さの 2 進表現をこのフィールドに入れます。 ISRT 呼び出しを出すときには、挿入したいレコードの長さの 2 進表現をこのフィールドに入れてから、 ISRT 呼び出しを出してください。

関連概念:

279 ページの『AIBTDLI インターフェース』

357 ページの『GSAM レコードの形式』

GSAM レコードのリトリーブと挿入

GSAM レコードは、順次または直接にリトリーブすることができます。新しいデータ・セットに GSAM レコードを追加することや、データベース内の既存のデータ・セットの終わりに新しいレコードを追加することもできます。

GSAM レコードを順次にリトリーブするには、GN 呼び出しを使用します。必要なパラメーターは GSAM PCB とセグメントの入出力域のみです。データベース全体を処理するには、GSAM PCB に GB 状況コードが戻されるまで GN 呼び出しを発行してください。この状況コードは、データベースの終わりに達したことを意味しています。ユーザーがデータベースの終わりに達すると、GSAM はデータベースを自動的にクローズします。新しいデータ・セットにレコードを追加したり、データベース内の既存のデータ・セットの終わりに新しいレコードを追加したりするには、ISRT 呼び出しを使用してください。GSAM は、ユーザーが指定したとおりの順序でレコードを追加します。

GSAM データベースからレコードを直接にリトリーブする方法もありますが、その場合は、GSAM データベースに対するレコードリトリーブ引数 (RSA) を指定します。RSA はセグメントリトリーブ引数 (SSA) に似ていますが、リトリーブしたいレコードの正確なアドレスを含んでいます。 RSA の具体的な内容と形式は、GSAM が使用するアクセス方式によって異なります。BSAM テープ・データ・セットおよび VSAM データ・セットの場合、RSA には相対バイト・アドレス (RBA) が入ります。BSAM ディスク・データ・セットの場合、RSA はディスク・アドレスを含んでおり、相対トラックおよびレコード形式を使用します。

アプリケーション・プログラムを変更して、ラージ・フォーマット・データ・セットのレコードをリトリーブするときに余分な 4 バイトに対応できるようにすることができます。それには、文字ストリング `RSA12` を含む入出力域を指定した INIT 呼び出しを使用します。 INIT RSA12 呼び出しは、GSAM データベースへの呼び出しをコーディングする前に、GSAM アプリケーション・プログラムにコーディングします。GSAM アプリケーションが INIT RSA12 呼び出しを発行すると、IMS に、ラージ・フォーマット・データ・セットのレコードをリトリーブする際にプログラムが 12 バイトの RSA を受け入れ可能であることが通知されます。INIT RSA12 呼び出しは、ラージ・フォーマット・データ・セットを使用するすべてのアプリケーションで発行する必要があります。ラージ・フォーマット・データ・セットに対する INIT RSA12 呼び出しの発行に失敗すると、予期せぬ結果が生じることがあります。INIT RSA12 呼び出しがない場合、IMS は、基本フォーマット・データ・セットのレコードをリトリーブする際に 8 バイトの RSA を返し続けます。

以下の表に、基本フォーマット・データ・セット用の RSA の形式を詳しく示します。

表 *46.* 基本フォーマット・データ・セット用の *RSA* の形式

位置	アドレス
位置 1 から 4	• バッファー内のブロックの BSAM (DASD) 相対トラックおよびレコード (TTRZ)。 • BSAM RBA。 • VSAM RBA。
位置 5	連結データ・セットの相対データ・セット。最初のデータ・セットの番号は 1 です。
位置 6	データ・セットの相対ボリューム。データ・セットの最初のボリュームは 1 です。
位置 7 および 8	現在の変位。

以下の表に、ラージ・フォーマット・データ・セット用の RSA の形式を詳しく示します。

表 47. ラージ・フォーマット・データ・セット用の *RSA* の形式

位置	アドレス
位置 1 から 4	• バッファー内のブロックの BSAM (DASD) 相対トラックおよびレコード (TTTR)。 • BSAM RBA。
位置 5	ゾーン・バイト
位置 6	連結データ・セットの相対データ・セット。最初のデータ・セットの番号は 1 です。
位置 7	データ・セットの相対ボリューム。データ・セットの最初のボリュームは 1 です。
位置 8 から 10	NULL バイト。未使用
位置 11 から 12	現在の変位。

GSAM データベースに対する GU 呼び出しで RSA を指定するには、その RSA が前に GN または ISRT 呼び出しの結果として返されている必要があります。 GSAM が RSA を返すようにするには、プログラム内の 8 バイト (基本フォーマット・データ・セット) または 12 バイト (ラージ・フォーマット・データ・セット) の RSA 保管域を指す第 4 パラメーターを指定して、GN または ISRT 呼び出しを発行する必要があります。該当の特定レコードをリトリーブする必要が生じるまで、この RSA を保管してください。

特定のレコードをリトリーブするには、そのレコードについての GU 呼び出しを発行し、そのレコードの RSA のアドレスを、GU 呼び出しの第 4 パラメーターとして指定します。 GSAM は、ユーザーが呼び出しパラメーターの一部として提供した名前の入出力域に、そのレコードを戻します。

制約事項: DASD 上の GSAM データベースからのみ、レコードを直接リトリーブしてください。バッファー入出力を使用する場合、出力 PCB のバッファー定義がパフォーマンスに影響する場合があります。

GSAM データベース内の位置のリセット

GU 呼び出しを使用して、GSAM データベース内の位置をリセットできます。

位置は、GSAM データベースの始めか、または GSAM データベース内の特定のセグメントにリセットできます。

- 基本フォーマット・データ・セットを使用して、位置を GSAM データベースの始めにリセットするには、2 進値 1 のフルワードの直後に 2 進値 0 のフルワードを続けた RSA を指定して、GU 呼び出しを発行します。
- ラージ・フォーマット・データ・セットを使用して、位置を GSAM データベースの始めにリセットするには、2 進値 1 のフルワードの直後に 2 進値 0 のフルワードを 2 つ続けた RSA を指定して、GU 呼び出しを発行します。
- 位置を GSAM データベース内の特定のセグメントにリセットするには、そのセグメントについての前回の ISRT または GN 呼び出しから、保管された RSA 値を含む RSA を指定して、GU 呼び出しを発行します。

関連資料:

360 ページの『GSAM のコーディングの考慮事項』

INIT 呼び出し (アプリケーション・プログラミング API)

GSAM に対する明示的なオープンおよびクローズ呼び出し

IMS は、最初の呼び出しが行われたときに GSAM データ・セットをオープンし、アプリケーション・プログラムが終了したときにこのデータ・セットをクローズします。したがって、常にアプリケーション・プログラムが GSAM に対して明示的なオープンまたはクローズ呼び出しを出す必要はありません。

ただし、次の状況では明示的な OPEN および CLSE 呼び出しが役立ちます。

- アプリケーション・プログラムが GSAM データ・セットをロードし、同じステップの中で (例えば、データ・セットをソートするために) GSAM を使用してそのデータ・セットを読み取る場合。アプリケーション・プログラムは、ロードが完了した後で GSAM CLSE 呼び出しを出す必要があります。
- GSAM データ・セットが出力データ・セットであって、プログラムの実行時に GSAM ISRT 呼び出しが出されない可能性がある場合。データ・セットは作成されません。その後で、存在しないデータ・セットを (GSAM を使用するか、あるいはこれを使用しないで) 読み取ろうとすると、エラーが発生する可能性があります。これを避けるためには、データ・セットを明示的にオープンしてください。このステップが終了すると、DL/I がデータ・セットをクローズします。データ・セットをクローズすると、空のデータ・セットに対する読み取りの試行を回避できます。
- GSAM データ・セットが出力データ・セットであって、データ・セット制御ブロック (DSCB) 内で EOF アドレスを過ぎた場所にデータが存在する場合。DSCB を更新できるようになる前に、直前のジョブ/ステップが異常終了した可能性があります。プログラムが再始動されても、プログラムが GSAM ISRT 呼び出しを行わない場合、DL/I がデータ・セットをクローズする際、ジョブ/ステップの終了時に EOF が更新されません。これにより、EOF アドレスを過ぎた場所に存在するデータが取り残される可能性があります。この状況を回避するには、データ・セットを明示的にオープンして、DSCB が正しい EOF アドレスによって更新されるようにしてください。

明示的な OPEN または CLSE 呼び出しには、入出力域パラメーターを含める必要はありません。PCB の処理オプション次第で、データ・セットは入力用または出力用にオープンされます。出力データ・セットに ASA または機械制御文字を入れるように指定できます。この呼び出しに入出力域パラメーターを含め、入出力域に OUTA を指定すると、ASA 制御文字が指示されます。OUTM を指定すると、機械制御文字が指示されます。

GSAM レコードの形式

GSAM レコードにはキーがありません。可変長レコードの場合、レコードの最初の 2 バイトでレコード長を指定しなければなりません。不定長レコードには、固定長レコードには、と同じように、データ (および、必要があれば制御文字) だけが含まれます。

不定長レコードを使用する場合、レコード長は GSAM DB PCB のキー・フィードバック域に続く 4 バイト・フィールドに入れられて、ユーザーのプログラムと GSAM の間で受け渡されます。このフィールドは不定長レコード域と呼ばれます。ISRT 呼び出しを出すときには、この長さを指定してください。 GN または GU 呼び出しを出すと、GSAM は戻されたレコードの長さをこのフィールドに入れます。不定長レコードを使用する利点は、レコードの初めにレコード長をもつ必要がなく、しかもレコードを固定長にする必要がないことです。レコードの長さは、ブロック・サイズ (BLKSIZE) 以下で、かつ 11 バイトよりも大きい長さ (z/OS の規則) でなければなりません。

VSAM を使用する場合、ブロック式または非ブロック式の固定長レコードまたは可変長レコードを使用できます。 BSAM を使用する場合、ブロック式または非ブロック式の固定長レコード、可変長レコード、または不定長レコードを使用できます。どの形式のレコードを使用する場合にも、GSAM DBD の DATASET ステートメントでその形式を RECFM キーワードに指定してください。この値は、JCL の DCB パラメーターの RECFM ステートメントで指定変更できます。また、どの形式の場合にも、JCL にキャリッジ制御文字を使用できます。

関連概念:

352 ページの『GSAM データベース用の PCB マスク』

361 ページの『GSAM データ・セット特性の指定元』

GSAM 入出力域

オプションの入出力域を指定する場合は、次の値のうちの 1 つを入れる必要があります。

- 入力データ・セットを表す INP
- 出力データ・セットを表す OUT
- ASA 制御文字を使用する出力データ・セットを表す OUTA
- 機械制御文字を使用する出力データ・セットを表す OUTM

GN、ISRT、および GU 呼び出しの場合の入出力域の形式は、レコードが固定長、不定長 (これは BSAM でのみ有効です)、または可変長のいずれであるかによって異なります。どの種類のレコードについても、制御文字を使用するように選択することできます。

固定長または不定長レコードの場合の入出力域の形式は、次のとおりです。

- 制御文字を使用しないときには、入出力域にはデータだけが含まれます。データはバイト 0 から始まります。
- 制御文字を使用するときには、バイト 0 に制御文字が入り、データはバイト 1 から始まります。

不定長レコードを使用する場合には、レコード長は、キー・フィードバック域に続く PCB フィールドに入れられて、ユーザーのプログラムと GSAM の間で受け渡されます。ISRT 呼び出しを出すときには、この長さを指定してください。 GN または GU 呼び出しを出すと、GSAM は戻されたレコードの長さをこのフィールドに入れます。このデータ長フィールドは 4 バイト長です。

可変長レコードには他のレコードにはない長さフィールドが含まれているため、形式は異なります。長さフィールドは 2 バイトです。可変長の入出力域には、固定長および不定長の入出力域と同じように、制御文字が含まれることがあります。

- 制御文字が含まれない場合には、バイト 0 および 1 には 2 バイトのデータ長フィールドが入り、データはバイト 2 から始まります。
- 制御文字が含まれる場合には、バイト 0 および 1 には同じく長さフィールドが入りますが、バイト 2 には制御文字が入り、データはバイト 3 から始まります。

GSAM 状況コード

ユーザーのプログラムは、それぞれの GSAM 呼び出しが出されたあとで、DL/I 呼び出しやシステム・サービス呼び出しが出された場合と同じように、状況コードを検査する必要があります。

状況コードを検査して、エラーが発生していたことが分かったときに、ユーザーのプログラムを終了させる場合には、プログラムを終了させる前にエラーのある PCB を必ずメモするようにしてください。問題判別には、この GSAM PCB アドレスが役立ちます。GSAM を使用するプログラムが異常終了すると、GSAM は内部的に PURGE および CLSE 呼び出しを出し、これによって PCB 情報が変更されます。

GSAM の場合に特有の意味をもつ状況コードは、次のとおりです。

AF GSAM が、無効な形式の BSAM 可変長レコードを検出しました。ユーザー・プログラムを終了させてください。

AH GU 呼び出しに RSA が指定されていません。

AI データ管理 OPEN エラーが起こっています。

AJ RSA で指定したパラメーターのうちの 1 つが無効です。

AM ユーザーが GSAM データベースに対して無効な要求を出しました。

AO データ・セットがアクセスされたとき、またはクローズされたときに入出力エラーが発生しました。

GB ユーザーがデータベースの終わりに到達し、GSAM がデータベースをクローズしました。次の位置は、このデータベースの先頭です。次の位置は、このデータベースの先頭です。

IX AI または AO 状況コードを受け取った後で、ユーザーが ISRT 呼び出しを出しました。ユーザー・プログラムを終了させてください。

GSAM でのシンボリック CHKP および XRST

GSAM データベースのチェックポイントをとるためには、シンボリック CHKP および XRST 呼び出しを使用してください。

GSAM を使用してデータ・セットの読み取りまたは書き込みを行うと、シンボリック CHKP および XRST 呼び出しを使用して再始動時にデータ・セットの位置変更ができ、ユーザーのプログラムを再始動可能にします。XRST 呼び出しを使用すると、IMS は処理のために GSAM データベースを位置変更します。CHKP および XRST

呼び出しは、バッチ・プログラム、バッチ指向 BMP、またはトランザクション指向 BMP として実行できるアプリケーション・プログラムで使用可能です。

制約事項: GSAM データベースを再始動する場合、以下のことに注意してください。

- シンボリック CHKP または XRST 呼び出しを指定して一時データ・セットを使用することはできません。
- 再始動時の SYSOUT データ・セットは、重複した出力データを含んでいることがあります。
- GSAM または VSAM データベースをロード中のプログラムを再始動することはできません。
- GSAM データベースのデータ・セットは、シンボリック CHKP 呼び出しが発行されたときと同じデータ・セット・フォーマット (BASIC または LARGE) でなければなりません。

IMS は、XRST 呼び出しで指定されているデータ域をリストアするときに、シンボリック CHKP 呼び出しを出した時点でユーザー・プログラムが使用していた GSAM データベースの位置変更も行います。シンボリック CHKP 呼び出しが出された時にユーザーのプログラムが GSAM データベースをロード中の場合、(それらのデータベースに BSAM でアクセスできる時は) IMS はデータベースを位置変更します。再始動処理の入力として使用するために GSAM データ・セットのコピーを作成するときには、例えば、SYSUT1 に RECFM=U を指定した IEBGENER を使用するなどの方法で、新規データ・セットに短ブロックが短ブロックとして書き込まれるようにしてください。元の GSAM データ・セットを使用して再始動を開始してください。

GSAM XRST の処理中に、位置変更する GSAM 出力データ・セットが空であるか、および、異常終了したジョブがそのデータ・セットにレコードを挿入したことがあったかを確認します。

GSAM のコーディングの考慮事項

GSAM データベースにアクセスするためにユーザー・プログラムで使用される呼び出しは、DL/I 呼び出しと同じではありません。ユーザーが GSAM で使用するシステム・サービス呼び出しは、シンボリック CHKP および XRST です。

次の表では、GSAM データベース呼び出しを要約しています。 GSAM データベースの処理に使用できる呼び出しは、次の 5 つです。

- CLSE
- GN
- GU
- ISRT
- OPEN

COBOL、PL/I、Pascal、C、およびアセンブラー言語での呼び出し形式およびパラメーターは同じであり、次表に記載しています。 GSAM 呼び出しは DL/I 呼び出しと大きな違いはありませんが、GSAM では GSAM PCB を参照する必要があり、また SSA を使用しない点で異なります。

Java バッチ処理 (JBP) 領域で実行する Java アプリケーション・プログラムは、IMS Java 従属領域リソース・アダプターを使用することで、GSAM データベースにアクセスできます。

表 *48. GSAM* 呼び出しの要約

呼び出し形式	意味	用途	オプション	パラメーター
CLSE	クローズ	GSAM データベースを明示的にクローズする。	ありません。	function, gsam pcb
GNbb	後続取り出し	次の順番のレコードをリトリーブする。	戻される RSA のアドレスを指定することができる。	function, gsam pcb, i/o area [,rsa name]
GUbb	初回取り出し	データベース内での位置を設定するか、あるいは固有レコードをリトリーブする。	ありません。	function, gsam pcb, i/o area, rsa name
ISRT	挿入	データベースの終わりに新しいレコードを追加する。	戻される RSA のアドレスを指定することができる。	function, gsam pcb, i/o area [,rsa name]
OPEN	オープン	GSAM データベースを明示的にオープンする。	印刷または穿孔制御文字を指定することができる。	function, gsam pcb [, open option]

関連資料:

841 ページの『JBP アプリケーションから GSAM データへのアクセス』

GSAM データ・セット特性の指定元

入力データ・セットの場合、レコード形式 (RECFM)、論理レコード長 (LRECL)、およびブロック・サイズ (BLKSIZE) は入力データ・セット・ラベルに基づいて決まります。

これらの情報がデータ・セット・ラベルで提供されない場合には、DD ステートメントまたは DBD 仕様が使用されます。この場合、DD ステートメントが優先されます。

出力データ・セットには、以下の特性があります。

- レコード・フォーマット
- 論理レコード長
- ブロック・サイズ
- その他の JCL DCB パラメーター
- DNS タイプ

レコード形式は、GSAM DBD の DATASET ステートメントで指定してください。オプションは次のとおりです。

- V (可変長)
- VB (可変長ブロック式)
- F (固定長)

- FB (固定長ブロック式)
- U (不定長)

V、F、または U の定義が適用され、これらは DD ステートメントの DCB=RECFM= 指定で上書きされません。 ただし、DD RECFM でブロック化が指定され、DBD でこれが指定されていない時には、RECFM はブロック化にセットされます。DD RECFM に A または M 制御文字が指定されている時には、その指定も適用されます。

不定形式レコードを使用するとき以外は、DBDGEN の DATASET ステートメントの RECORD= パラメーターを使用して論理レコードを指定するか、あるいは DD ステートメントで DCB=LRECL=xxx を使用してください。両方の方法で論理レコードを指定した場合には、DD ステートメントが優先されます。最大レコード長については、以下の表を参照してください。

表 *49.* レコード・フォーマット別の *GSAM* データ・セットの *BSAM* および *VSAM* の論理レコード長

レコード・フォーマット	**BSAM** 論理レコード長	**VSAM** 論理レコード長
固定/固定長ブロック式	32760 バイト	32760 バイト
可変/可変長ブロック式	32756 バイト	32756 バイト
不定長	32760 バイト	サポートされない

DBDGEN の DATASET ステートメントの BLOCK= または SIZE= パラメーターを使用してブロック・サイズを指定するか、あるいは DD ステートメントで DCB=BLKSIZE=xxx を使用してください。両方の方法でブロック・サイズを指定した場合には、DD ステートメントが優先されます。DBD または DD ステートメントでブロック・サイズが指定されない場合、不定形式レコードが使用されている場合を除き、システムは装置タイプにもとづいてサイズを決定します。

これ以外に、次のような JCL DCB パラメーターを使用できます。

- CODE
- DEN
- DNSTYPE
- TRTCH
- MODE
- STACK
- PRTSP (RECFM に A または M が指定されていないときに使用可能です)
- DCB=BUFNO=X (これを指定すると、GSAM は X 個のバッファーを使用するようになります)

制約事項**:** BFALN、BUFL、BUFOFF、FUNC、NCP、および KEYLEN は使用しないでください。

関連概念:

357 ページの『GSAM レコードの形式』

GSAM データ・セット用の DD ステートメントの DISP パラメーター

DD ステートメントの DISP パラメーターは、入力または出力のいずれのデータ・セットを作成するかによって、また、データ・セットをどのように使用する予定であるかによって異なります。

重要: 空でないデータ・セットを使用した通常の開始に DISP=OLD または DISP=SHR パラメーターを指定すると、既存のレコードがデータ・セットの先頭から上書きされます。

- 入力データ・セットの場合には、DISP=OLD パラメーターを使用します。
- 出力データ・セットの場合には、以下のオプションを考慮します。
  - DD ステートメントで割り振られた出力データ・セットを作成するには、DISP=NEW を設定してください。
  - 通常の開始時または障害後の再始動時に、空のデータ・セットに新規レコードを追加するには、DISP=MOD、DISP=SHR、または DISP=OLD を設定します。
  - ステップを再開する場合、既存のデータ・セットには DISP=OLD を設定し、空のデータ・セットには DISP=MOD を設定します。
  - 障害後の再始動時に、空でない既存のデータ・セットに新規レコードを追加するには、DISP=MOD、DISP=SHR、または DISP=OLD を設定します。これらのパラメーターは、既存のデータ・セット上の再始動点から新規レコードを追加します。
  - 通常の開始時に、空でない既存のデータ・セットの末尾に新規レコードを追加するには、DISP=MOD を設定します。

GSAM データ・セット用の拡張チェックポイント再始動

GSAM データ・セットに拡張チェックポイント再始動を使用する場合の推奨事項は、次のとおりです。

- 通過データ・セットを使用しないでください。
- 先行のステップでデータ・セットに対する逆方向参照を行わないでください。
- 既存のテープ・データ・セットにレコードを追加するために DISP=MOD を使用しないでください。
- DISP=DELETE または DISP=UNCATLG を使用しないでください。
- DFSMS ストライプ・データ・セットは、以下の条件のもとで使用してください。
  - データ・セットが SMS によって管理される場合。
  - データ・セットが、ボリュームについてのシステムのエクステント限度を超える可能性がある場合。
- さらに、次の事項に留意してください。
  - SYSIN、SYSOUT、または一時データ・セットの再配置が行われないようにしてください。
  - 連結 DD ステートメント用のいずれかのデータ・セットが SYSIN または SYSOUT の場合、それらのどの連結データ・セットの再配置も行われないようにしてください。

- 連結データ・セットを使用する場合には、再始動時とチェックポイント時で同じ数と順序のデータ・セットを指定します。
- GSAM/VSAM ロード・モードの制約事項は、非ストライプ・データ・セットとストライプ・データ・セットのどちらにも適用されます。
- シンボリック・チェックポイント呼び出しを出す際に、オープンされている GSAM VSAM 出力データ・セットが PSB に含まれていると、システムは、データベースの PCB に AM 状況コードを警告として戻します。これは、データ・セットが再始動時に再配置されておらず、チェックポイントが正常に完了したことを意味します。
- CLSE 呼び出しの後で ISRT 呼び出しが出され、GSAM データ・セットが DISP=OLD として定義されている場合、CLSE 呼び出しより前に行われたすべての CHKP 呼び出しには無効な位置変更情報が含まれます。DISP=OLD を使用する場合はすべて、拡張再始動 (XRST) 後の異常終了 U0271 を避けるために、CLSE の後で必ず CHKP 呼び出しを出してください。

チェックポイントと再始動の間での GSAM データ・セットのコピー

非ストライプ GSAM DASD データ・セットの再始動時に GSAM データ・セットを配置するには、相対トラックおよびレコード形式 (TTRZ、またはラージ・フォーマット・データ・セット用の TTTRZ) を使用します。

GSAM は再始動時に、TTRZ または TTTRZ をボリューム上で使用して、非ストライプ GSAM DASD データ・セットを配置します。テープ・データ・セットの場合には、ボリュームの相対レコードが使用されます。テープ・ボリューム上の相対レコードを変更することはできません。

非ストライプ DASD データ・セットをチェックポイントと再始動の間でコピーするには、次のようにします。

- データ・セットを同じ装置タイプにコピーします。
- 入力と出力のどちらのデータ・セットの場合でも、不定形式レコード (RECFM=U)を使用した再ブロックは行わないでください。

コピーされた各ボリュームには、元のボリュームと同じ数のレコードが含まれます。

注**:** GSAM は、相対ブロック番号 (RBN) を使用してストライプ DASD データ・セットの再配置を行います。SMS によって管理されるデータ・セットが GSAM データベースで使用されている場合、各ボリュームのコピー方法をユーザーが制御することはできません。非ストライプ DASD データ・セットとは異なり、データ・セットのコピー後には、再始動レコードの TTRZ または TTTRZ が変更されていないことを確認する必要はありません。

非ストライプ・データ・セットからストライプ・データ・セットへのデータ・セットの変換

システム割り振り限度を超えるかシステム X'37' エラー条件が発生した場合、拡張再始動を行うことが必要になる前に、GSAM/BSAM 非ストライプ・データ・セットをストライプ・データ・セットに変換してください。 SMS によって管理されていない非ストライプ・データ・セットは、ボリュームのみによる初期の 1 次または

2 次割り振りを超えて拡張されますが、SMS によって管理される非ストライプ GSAM/BSAM 複数ボリューム・データ・セットでは、結果としての新しいスペース割り振りがデータ・セット内のすべてのボリュームについて有効になります。

SMS によって管理されている非ストライプ・データ・セットをスペース割り振り値の変更後にコピーした場合、新しいボリュームのレコード数は古いボリュームのレコード数と異なります。新しい 1 次および 2 次割り振り値は、非ストライプ・データ・セットで使用されます。データがコピーされるとき、新規ボリューム上に割り振られたすべてのスペースが使用されてから、次ボリュームへデータがコピーされます。

GSAM/BSAM 非ストライプ・データ・セットの処理時にエラー条件が発生し (システム X'37' が発生したかシステム割り振り限度を超えた)、エラー発生後にそのデータ・セットがストライプ・データ・セットに変換された場合、障害後の再始動は正常に完了しません。チェックポイント呼び出しが発行されると、データベースの再始動時にアプリケーション・プログラムを再配置するために、チェックポイント操作により TTRZ または TTTRZ 値がログ・レコードに保管されます。GSAM 再始動時にストライプ・データ・セット用のログ・レコードが使用されますが、その際、再配置を行うために相対ブロック番号 (RBN) が必要になります。

GSAM で使用される連結データ・セット

GSAM では連結データ・セットを使用できます。これらのデータ・セットは、類似しない装置タイプ (例えば、DASD とテープ) にあるものでも、異なる DASD 装置上のものでもかまいません。論理レコード長とブロック・サイズは、異なっていてもかまいません。この場合、ブロック・サイズが、最大のデータ・セットを最初に連結する必要はありません。

単一 DD ステートメントで指定できる連結データ・セットの最大数は 255 です。最初に連結されたデータ・セットに関して判別されたバッファー数が、後続のすべてのデータ・セットに使用されます。世代別データ・グループを使用すると、連結データ・セットが作成されることがあります。

GSAM データ・セット属性の指定

GSAM データ・セット属性を指定する場合には、以下の設定を推奨します。

- DBD で RECFM を指定します。(この指定は必須です。)
- DATASET ステートメントで、RECORD= を使用して論理レコード長を指定します。 DASD ボリューム上でデータ・セットが 65535 トラックより大きくなる可能性があり、データ・セットが複数のボリュームにまたがらないようにしたい場合、DSNTYPE=LARGE パラメーターを指定します。
- DD ステートメントでは、LRECL、RECFM または BLKSIZE を指定しないでください。RECFM=U の場合を除き、システムがブロック・サイズを判別します。システムは DBD にもとづいて論理レコード長を判別します。
- PSB については、出力用には PROCOPT=LS を指定し、入力用には GS を指定してください。S を指定すると、GSAM はパフォーマンスを向上させるために単一バッファーではなく複数バッファーを使用します。

IMS は、DBD に指定されたレコード長の値に 2 バイトを加えます。これは、BSAM RDW を構成するために必要な ZZ フィールドを収容するためです。データ

ベースが GSAM または BSAM であり、レコードが可変である (V または VB) 場合は必ず、IMS は、アプリケーションから渡された GSAM レコードのレコード長値に 2 バイトを加えます。この追加によって IMS は、BSAM RDW (レコード記述子ワード) を構成する ZZ フィールドを収容することができます。

レコードが可変の GSAM および BSAM の例

```
 //IDASD DD DUMMY
//ODASD DD UNIT=SYSDA,VOL=SER=000000,DISP=(,KEEP),
// SPACE=(TRK,(5,1)),DSN=GSAM.VARIABLE1,
// DCB=(RECFM=VB,BLKSIZE=32760,LRECL=32756)
//SYSIN DD *,DCB=BLKSIZE=80
S 1 1 1 1 1 DBDNAME
L ISRT
L V8187 DATA 1ST RECORD LOADED TO GSAM <---RDW
L ISRT
L V8187 DATA 2ND RECORD LOADED TO GSAM
L ISRT
L V8187 DATA 3RD RECORD LOADED TO GSAM
L ISRT
L V8187 DATA 4TH RECORD LOADED TO GSAM
```

上記の例では、1 つの 32756 バイト (MVS) のレコードに、4 つの GSAM レコード (IMS セグメント) を入れることができます。

DLI、DBB、および BMP 領域タイプと GSAM

GSAM データベースをアクセスするために、IMS は、PSBLIB、DBDLIB、および ACBLIB の PSB および DBD 情報を使用して DLI 制御ブロックを作成します。PSB および DBD 情報のソースは、領域タイプによって異なります。

DLI オフライン・バッチ領域の場合、IMS は、PSBLIB および DBDLIB から PSB および DBD 情報を入手します。DBB オフライン・バッチ領域の場合、IMS データベース管理は、ACBLIB から PSB および DBD 情報を入手します。オンライン・バッチ領域 (BMP) の場合、IMS は、ACBLIB の情報を使用して DLI 制御ブロックを作成します。アプリケーションが BMP 領域でスケジュールされ、アプリケーションに関連付けられた PSB に 1 つ以上の GSAM PCB がある場合、IMS スケジューリングは、ACBLIB および PSBLIB から PSB 情報を入手します。この場合、ACBLIB と PSBLIB の PSB は同じでなければなりません。GSAM データベース管理は、ACBLIB から PSB および DBD 情報を入手しません。代わりに、GSAM データベース管理は、PSBLIB および DBDLIB から PSB および DBD 情報を入手します。

GSAM で DLI、DBB、BMP 領域を初期設定するときは、//IMS DD ステートメントと GSAM DD ステートメントを含めなければなりません。DBB 領域または BMP 領域で GSAM を使用しない場合、//IMS DD ステートメントを含める必要はありません。PSB および DBD をロードして GSAM 制御ブロックを作成するには、//IMS DD ステートメントを含める必要があります。次の図では、65535 トラックより大きいデータ・セットを持つ //IMS DD ステートメントの例を示します。

図 *64. //IMS DD* ステートメントの例

```
//STEP       EXEC     PGM=DFSRRC00,PARM=[BMP|DBB|DLI],...'
//STEPLIB    DD       DSN=executionlibrary-name,DISP=SHR
//           DD       DSN=pgmlib-name,DISP=SHR
//IMS        DD       DSN=psblib-name,DISP=SHR
//           DD       DSN=dbdlib-name,DISP=SHR
//IMSACB     DD       DSN=acblib-name,disp=shr (required for DBB)
//SYSPRINT   DD       SYSOUT=A
//SYSUDUMP   DD       SYSOUT=A
//ddnamex    DD       (add DD statements for required GSAM databases)
//ddnamex    DD       (add DD statements for non-GSAM IMS databases
                      for DLI/DBB)
//ddnamex    DD       DSNTYPE=LARGE,...
                      .
                      .
                      .
/*
```

第 21 章 高速機能データベースの処理

アプリケーション・プログラムを、主記憶データベースおよび高速処理データベースを含む、高速機能データベースにアクセスするように作成できます。

高速機能データベースには、次の 2 つの種類があります。

- 主記憶データベース (MSDB) は、DB/DC 環境で使用できます。 MSDB には、ユーザーが最も頻繁にアクセスするデータを記憶するルート・セグメントのみが収められます。
- 高速処理データベース (DEDB) は、最高 15 の階層レベルと 127 のセグメント・タイプから構成することのできる階層データベースです。 DEDB は、IMS ユーザーも、DBCTL を使用する CICS ユーザーも使用できます。

制約事項**:** この DEDB 情報は、DBCTL を使用する CICS ユーザーに適用されます。 CICS ユーザーは、読み取りモードの DBCTL で MSDB にアクセスできますが、更新モードはサポートされていません。

VSO の考慮事項

VSO はアプリケーションの処理に影響を与えません。データが常駐する場所は、アプリケーションにとって重要なことではありません。

MSDB と DEDB 用のデータ・ロック

FLD 呼び出しを含むすべての MSDB 呼び出しは、セグメント・レベルでデータをロックできます。呼び出しが処理され、呼び出しの最後で解放されるときにロックが取得されます。HSSP 呼び出しを除くすべての DEDB 呼び出しは VSAM CI レベルでロックされます。単一セグメント、ルートのみ、固定長 VSO 区域の場合、PROCOPT R または G を指定すると、アプリケーション・プログラムはすべての呼び出し用にセグメント・レベルのロックを取得します。その他の PROCOPT を指定した場合、アプリケーション・プログラムは VSAM CI ロックを取得します。

セグメント・レベル・ロッキング (SLL) は、 2 層のロック構造を提供します。まず、CI 全体に対して共用 (SHR) ロックが獲得されます。その後、要求されたセグメントに対して、排他的 (EXCL) セグメント・ロックが獲得されます。この構造は、CI の SLL ユーザーと CI の EXCL リクエスターの間の競合検出を考慮しています。既存の EXCL CI ロック・ユーザーと SHR CI ロック・リクエスターの間に競合が発生した場合、SHR CI ロックが EXCL CI ロックにアップグレードされます。 EXCL CI ロックが保持されている間は、それ以後の SHR CI ロック要求は、次のコミット・ポイントで EXCL CI が解放されるまで待機しなければなりません。

DEDB FLD 呼び出しは、呼び出し時にロックされません。その代わり、コミット・ポイントでロックされます。

同期点処理の間、ロックは再度取得され (まだ保留されていない場合)、変更内容が検査されます。検査に障害があると、メッセージが再処理される (メッセージ・ド

リブンのアプリケーションの場合) か、あるいは FE 状況コードが戻されます (非メッセージ・ドリブンのアプリケーションの場合)。 FLD 呼び出しで使用されたセグメントが同期点より前で削除あるいは置換されていると、検査が失敗することがあります。

FLD 呼び出しのセグメント・リトリーブは、GU 呼び出しのセグメント・リトリーブと同じです。非修飾 GU 呼び出しと同じように、非修飾 FLD 呼び出しは現行区域内の最初のセグメントを戻します。FLD 呼び出しが処理された後、先行する呼び出しによって変更されていない場合、現行の CI のすべてのロックは解放されます。

圧縮ルーチンが、ルート専用構造をもつ DEDB のルート・セグメントに定義されているとき、しかもそのルート・セグメントが固定長セグメントであるときは、圧縮後のその長さが変数になります。圧縮されたセグメントを置き換えるには、削除と挿入を行わなければなりません。この場合は、セグメント・レベルの制御とロッキングは使用可能になりません。

関連概念:

- 高速処理データベース (データベース管理)
- 主記憶データベース (MSDB) (データベース管理)
- 高速順次処理 (HSSP) (データベース管理)

高速機能データベース呼び出し

高速機能データベースにアクセスするには、アプリケーション・プログラムで高速機能データベース呼び出しを使用します。

次の表では、高速機能データベースで使用できるデータベース呼び出しを要約しています。

表 *50.* 高速機能データベース呼び出しの要約：

	MSDB のタイプ			
機能コード	非端末関連	端末関連固定	端末関連動的	**DEDB**
DEQ				X
FLD	X	X	X	X
GU、GHU	X	X	X	X
GN、GHN	X	X	X	X
GNP, GHNP DLET			X	X
ISRT			X	X
POS				X
REPL	X	X	X	X
RLSE				X

DEDB に対する DL/I 呼び出しには、既存の階層内のレベル (最高 15) と同数の SSA を含むことができます。また、コマンド・コードと複数の修飾ステートメントを含めることもできます。

制約事項:

- 高速機能は、順次従属セグメントで使用されたコマンド・コードを無視します。
- 使用する呼び出しに適用されないコマンド・コードを指定すると、高速機能はそのコマンド・コードを無視します。
- 設定された親を超えるレベルの SSA で F または L を使用すると、高速機能はその F または L コマンド・コードを無視します。
- DEDB に対する DL/I 呼び出しには、独立 AND は指定できません。この AND 演算子は、副次索引でのみ使用されるものです。

DEDB に対する呼び出しでは、すべてのコマンド・コードを使用できます。サブセット・ポインターを使用する DEDB への呼び出しのみが、R、S、Z、W、および M コマンド・コードを使用できます。次の表では、これらのコマンド・コードと一緒に使用できる呼び出しを示しています。

表 *51.* サブセット・ポインター・コマンド・コードおよび呼び出し

コマンド・コード	DLET	GU GHU	GN GHN	GNP GHNP	ISRT	REPL
M		X	X	X	X	X
R		X	X	X	X	
S		X	X	X	X	X
W		X	X	X	X	X
X	X	X	X	X	X	X

主記憶データベース (MSDB)

MSDB には、ルート・セグメントのみが含まれています。各セグメントは、特定の対象に関するすべての情報を含んでいる点でデータベース・レコードに似ています。

DL/I 階層では、データベース・レコードはルート・セグメントと、それに従属するすべてのセグメントからなります。例えば、医療階層では、特定の PATIENT セグメントおよびその PATIENT セグメントの下のすべてのセグメントによって、その患者に関するデータベース・レコードが構成されます。MSDB では、当該セグメントによってデータベース・レコード全体が構成されます。データベース・レコードには、そのセグメントに含まれている以外のフィールドは含まれません。 MSDB セグメントは、固定長です。

MSDB には、端末関連と非端末関連の 2 つの種類があります。端末関連 MSDB では、各セグメントが 1 つの論理端末によって所有されています。端末は、所有しているセグメントを更新できます。関連 MSDB には、固定関連と動的関連があります。動的関連 MSDB ではセグメントの追加および削除を行うことができます。固定関連 MSDB では、セグメントの追加や削除を行うことはできません。

もう 1 つの種類の MSDB は、非端末関連 MSDB あるいは簡潔に非関連 MSDB と呼ばれます。非関連 MSDB 内のセグメントは、論理端末によって所有されません。

MSDB に関する呼び出しの使用上の制約事項

MSDB からセグメントをリトリーブするには、他の IMS データベースからセグメントをリトリーブするときと同じように、Get 呼び出しを発行できます。 MSDB にはルート・セグメントのみが含まれているため、GU と GN 呼び出し (および、セグメントを更新するときには GHU と GHN 呼び出し) のみが使用されます。 SSA 内のセグメント名フィールドに *MYLTERM が含まれている場合、GU、GHU、および FLD 呼び出しでは LTERM 所有のセグメントが戻され、SSA の残りの部分は無視されます。

MSDB を処理するときには、MSDB に対する呼び出しとその他の IMS データベースに対する呼び出しのと間の、以下の相違点を考慮してください。

- 1 つの MSDB に対する呼び出しでは、1 つの SSA だけしか使用できません。
- MSDB 呼び出しでコマンド・コードを使用することはできません。
- MSDB 呼び出しで複数の修飾ステートメント (ブール演算子) を使用することはできません。
- MSDB セグメント・キーの最大長は、(他の IMS データベースの場合のように 255 バイトではなく) 240 バイトです。
- データベース記述 (DBD) で指定された算術フィールド (タイプ P 、H、または F) を SSA で指名すると、データベース検索は、(DL/I 呼び出しで使用される論理比較ではなく) 算術比較を使用して行われます。
- 16 進数フィールドを指定すると、データベース・フィールド内の各バイトは、SSA では 2 文字の 16 進数表記で表されます。この表記法が使用されることにより、検索引数はデータベース・フィールドの 2 倍の長さになります。

  16 進数タイプの SSA 修飾ステートメント内の文字は、データベース形式に変換される前に妥当性検査されます。0 から 9 までの数字と A から F までの文字だけが受け入れられます。
- 端末関連および非端末関連の LTERM キー付き MSDB は、ETO または LU 6.2 端末ではサポートされません。これらの MSDB へのアクセスを試みると、データはリトリーブされず、AM 状況コードが戻されます。ETO および LU 6.2 の詳細については、「*IMS V15* コミュニケーションおよびコネクション」を参照してください。
- MSDB は、シスプレックス・グループ内の IMS サブシステム間で共用することはできません。高速機能急送メッセージ・ハンドラー (EMH) を使用すると、端末関連および非端末関連の端末キー付き MSDB は静的端末からのみアクセスできます。これらの静的端末は、DBFHAGU0 (入力編集ルーター出口ルーチン) に指定されるようにローカル・オンリーのシスプレックス処理コード (SPC) でトランザクションを実行します。

上記の制約事項は、CICS ユーザーには適用されません。

高速処理データベース (DEDB)

DEDB には、ルート・セグメントと、最大 127 個の従属セグメント・タイプが含まれます。うち 1 つは順次従属セグメントであってかまいませんが、残り 126 個は直接従属セグメントです。順次従属セグメントは日時順に保管され、直接従属セグメントは階層的に保管されます。

DEDB を使用すると、データ使用可能性が向上します。各 DEDB は、複数の「エリア」に区分化または分割することができます。 各区域には、データベース・レコードの別々の集合が入ります。さらに、エリア・データ・セットごとに最大 7 つのコピーを作成できます。区域の 1 つのコピーに誤りが存在する場合、アプリケーション・プログラムはその区域の他のコピーを使用してデータへのアクセスを継続することができます。区域のコピーの使用は、アプリケーション・プログラムに影響を与えません。DEDB 内のデータにエラーが発生しても、IMS は、データベースを停止しません。 IMS は、エラーのあるデータを使用不能にしますが、アプリケーション・プログラムのスケジュールと処理は継続します。エラーのあるデータを必要としないプログラムには、影響がありません。

別々の IMS システム内のアプリケーション・プログラム間で、DEDB を共用することができます。DEDB を共用するということは、実質的に全機能データベースを共用することと同じなので、同じ規則のほとんどが適用されます。 複数の IMS システムが、エリア・レベル (全機能データベースにおけるデータベース・レベルではない) で DEDB を共用するか、あるいはブロック・レベルで DEDB を共用することができます。

関連資料: DEDB データ共用の詳細については、「*IMS V15* システム管理」に記載される、データを共用する IMS システムの管理の説明を参照してください。

セグメントの更新: REPL、DLET、ISRT、および FLD

MSDB または DEDB を更新するために使用できる呼び出しのうちの 3 つは、他の IMS データベースの更新に使用される呼び出しと同じで、REPL、DLET、および ISRT です。

REPL 呼び出しは関連 MSDB または非関連 MSDB に対して出すことができ、非端末関連 MSDB (端末関連キーは除く) と DEDB に対しては 3 つの呼び出しをすべて出すことができます。 REPL または DLET 呼び出しを出す場合には、MSDB や DEDB に対して出す場合でも、他の IMS データベース内のセグメントの置き換えまたは削除を行うときと同様に、更新したいセグメントに関する Get Hold 呼び出しを先に出しておく必要があります。

MSDB および DEDB に対して使用でき、その他のタイプの IMS データベースに対して使用できない呼び出しは、フィールド (FLD) 呼び出しです。これを使用すると、セグメント内のフィールドの内容へのアクセスとその変更が可能になります。FLD 呼び出しは、次の 2 つのことを行います。

- FLD/VERIFY

  このタイプの呼び出しは、ターゲット・セグメント内のフィールドの値を、ユーザーが FSA に提供した値と比較します。

- FLD/CHANGE

このタイプの呼び出しは、ユーザーが FSA に指定した方法でターゲット・セグメント内のフィールドの値を変更します。FLD/CHANGE 呼び出しは、前の FLD/VERIFY 呼び出しが成功した場合にのみ成功します。

FLD は、Get Hold と REPL の 2 つの呼び出しが行うことを、 1 つの呼び出しで行います。例えば、『非関連 MSDB における口座セグメント』のトピックで示した ACCOUNT セグメントを使用する場合、ある銀行で、特定の顧客が口座からある金額を引き出すことができるかどうかを調べるために、次の処理を行う必要があります。

1. 顧客の口座のセグメントをリトリーブする。
2. 口座にある残高が、顧客が引き出そうとしている金額よりも多いかどうかを調べる。
3. 残高が引き出し額よりも多い場合には、その引き出し額に応じて残高を更新する。

FLD 呼び出しを使用しない場合、プログラムは、GU 呼び出しを出してセグメントを検査し、プログラム論理を使用してその内容を検査し、そのうえで REPL 呼び出しを出して、引き出し額に応じて残高を更新することになります。ルート SSA を使用して FLD 呼び出しを使用すると、希望するセグメントをリトリーブできます。FLD 呼び出しの SSA の形式は、他の呼び出しの場合と同じです。SSA が指定されていない場合には、 MSDB 内または DEDB 内の最初のセグメントがリトリーブされます。FLD/VERIFY を使用して、BALANCE フィールドと引き出し額を比較します。この比較が満たされると、FLD/CHANGE によって BALANCE フィールドを更新できます。

FLD 呼び出しによってリトリーブされたセグメントは、GHU 呼び出しによってリトリーブできるセグメントと同じです。 FLD 呼び出しが出された後、その位置は消失します。 FLD 呼び出しの後に非修飾 GN 呼び出しを出すと、現行区域内の次のセグメントが戻されます。

制約事項: 以下の環境では、呼び出しの削除や置き換えを長さの変更と同時に実行しないでください。

- いずれかの DEDB AREA に対する FLD 呼び出し
- ルート専用 DEDB AREA に対する、VIEW=MSDB または VIEW=MSDBL を指定した GU 呼び出し

アプリケーションは ABEND U1026 となります。または GN 呼び出しで一部のセグメントがスキップされる場合があります。

フィールドの内容の検査: FLD/VERIFY

FLD/VERIFY 呼び出しは、セグメント内の特定のフィールドの内容をユーザーが指定した値と比較します。 FLD/VERIFY 呼び出しが 2 つの値を比較する方法は、ユーザーが指定した演算子によって異なります。

フィールドの名前と、比較対象の値を指定する場合には、次のいずれかの方法で比較を行うことができます。

- フィールドの値が、ユーザーが指定した値に等しいか。
- フィールドの値が、ユーザーが指定した値よりも大きいか。

- フィールドの値が、ユーザーが指定した値よりも大きいかまたは等しいか。
- フィールドの値が、ユーザーが指定した値よりも小さいか。
- フィールドの値が、ユーザーが指定した値よりも小さいかまたは等しいか。
- フィールドの値が、ユーザーが指定した値と等しくないか。

IMS は、ユーザーが要求した比較を行った後で、 (PCB に戻す状況コードの他に) 比較の結果を示す状況コードを戻します。

フィールドの名前、およびそのフィールド値と比較したい値は、フィールド検索引数 (FSA) で指定します。FSA は、IMS が状況コードを戻すときにも使用されます。 FLD 呼び出しを出す前に FSA を入出力域に入れて、呼び出しの中でその入出力域を参照します (これは、DL/I 呼び出しにおける SSA の使用法と同じです)。FSA は、データベースからリトリーブしたい情報に関する情報を IMS に提供するために使用される点では、SSA に似ています。ただし、FSA には SSA よりも多くの情報が入ります。次の表では、FSA の構造および形式を示しています。

表 *52. FSA* 構造

FSA コンポーネント	フィールド長
FLD 名	8
SC	1
OP	1
FLD 値	可変
CON	1

FSA には次の 5 つのフィールドがあります。

フィールド名 (FLD 名)
これは、更新するフィールドの名前です。このフィールドは、DBD で定義されていなければなりません。

状況コード (SC)
これは、IMS がこの FSA の状況コードを戻す場所です。IMS による FSA の処理が正常に行われると、ブランクの状況コードが戻されます。IMS は、FSA の処理に失敗すると、FSA にブランク以外の状況コードがあることを示すために PCB に FE 状況コードを戻し、ブランク以外の FSA 状況コードを戻します。IMS が FLD/VERIFY 呼び出しでユーザーに戻す可能性のある FSA 状況コードは、以下のとおりです。

B フィールド値に指定したデータの長さが無効です。またはデータベース内のデータのセグメント長が、DBD で指定したフィールド長を格納するには十分ではありません。

D 検証チェックが正常に行われませんでした。すなわち、ユーザーの照会に対する答は NO です。

E フィールド値に無効なデータが含まれています。ユーザーがこのフィールドに指定したデータが、このフィールドに関して DBD で定義されているデータのタイプと異なっています。

H セグメント内に要求されたフィールドがありません。

演算子 (OP)

これは、2 つの値を比較する方法を IMS に指示します。FLD/VERIFY 呼び出しの場合、次のように指定できます。

E フィールド内の値が、ユーザーが FSA で指定した値と等しいかどうか検査する。

G フィールド内の値が、ユーザーが FSA で指定した値よりも大きいかどうか検査する。

H フィールド内の値が、ユーザーが FSA で指定した値以上かどうか検査する。

L フィールド内の値が、ユーザーが FSA で指定した値よりも小さいかどうか検査する。

M フィールド内の値が、ユーザーが FSA で指定した値以下かどうか検査する。

N フィールド内の値が、ユーザーが FSA で指定した値と等しくないかどうか検査する。

フィールド値 (FLD 値)

この区域には、IMS にセグメント・フィールドの値と比較させる値が入っています。この区域に指定するデータは、FSA の最初のフィールドで名前を指定したフィールド内のデータと同じタイプでなければなりません。データのタイプとしては、 16 進数、パック 10 進数、英数字 (あるいは各種のデータ・タイプの組み合わせ)、2 進数フルワード、および 2 進数ハーフワードの 5 つがあります。この区域内のデータ長は、DBD 内のこのフィールドに対して定義される長さと同じである必要があります。

例外:

- 16 進データを処理するときには、FSA 内のデータは 16 進数でなければなりません。すなわち、FSA 内のデータの長さは、このデータベースのフィールド内のデータの長さの 2 倍になります。IMS は、16 進数フィールドの中の文字データをデータベース形式に変換する前に、データの妥当性をチェックします。(有効な文字は、0 から 9 までと A から F までだけです。)
- パック 10 進数データの場合、フィールド値に先行ゼロを付ける必要はありません。パック 10 進数データの場合、フィールド値に先行ゼロを付ける必要はありません。すなわち、FSA 内の桁数は、対応するデータベース・フィールドの桁数より少なくても構いません。このフィールドに指定するデータは、有効なパック 10 進数形式でなければならず、また、符号桁で終わっていなければなりません。このフィールドに指定するデータは、有効なパック 10 進数形式でなければならず、また、符号桁で終わっていなければなりません。

IMS は、FSA を処理するときに、英数字フィールドと 16 進数フィールドについては論理比較を行い、パック 10 進数フィールドと 2 進数フィールドについては算術比較を行います。

結合子 (CON)

この FSA が呼び出し内の唯一または最後の FSA である場合には、この区域にブランクが入ります。この FSA の後に別の FSA が続く場合には、この区域にアスタリスク (*) が入ります。 FSA が参照するすべてのフィールドが同一セグ

メントに入っている場合には、複数の FSA を 1 つの FLD 呼び出しに含めることができます。FLD 呼び出しに関してエラー状況コードが出された場合には、FLD 呼び出しの各 FSA の状況コードを調べ、どこにエラーがあるかを判別してください。

データベース内のフィールドの内容を検査したあとで、同じ呼び出しの中でそのフィールドの内容を変更できます。そのためには、そのフィールドに関する変更操作を指定する FSA を提供してください。

フィールドの内容の変更: FLD/CHANGE

あるフィールドの内容を変更することを IMS に通知するには、FLD/VERIFY 呼び出しの場合のように、FSA を使用します。

ただし、指定できる演算子と、呼び出し後に IMS から返されてくる FSA 状況コードが多少違います。 FLD/CHANGE は、以下のように使用します。

- ユーザーは、変更したいフィールドの名前を FSA の最初のフィールド (フィールド名) に指定します。
- ユーザーは、FSA の 3 番目のフィールド (演算子) に演算子を指定します。これにより、そのフィールドをどのように変更するかが IMS に指示されます。
- ユーザーは、FSA の最後の区域 (フィールド値) に、フィールドを変更するために IMS が使用する値を指定します。

FLD/CHANGE 呼び出しでさまざまな演算子を指定することにより、データベース内のフィールドを次のように変更できます。

- FSA の中で指定した値を、フィールドの値に加える。
- FSA の中で指定した値を、フィールドの値から減算する。
- データベース・フィールドの値を、FSA の中で指定した値にセットする。

これらの演算子は、次の記号を使用して FSA でコーディングします。

- 加算: +
- 減算: -
- フィールド値を新しい値にセット: =

値の加算と減算が行えるのは、データベース内のフィールドに算術 (パック 10 進数、2 進数フルワード、または 2 進数ハーフワード) データが含まれている場合に限られます。

FLD/CHANGE FSA に戻される可能性のある状況コードは、次のとおりです。

A 操作の誤り。例えば、文字データが入っているフィールドに関してユーザーが + 演算子を指定した場合。

B データ長の誤り。ユーザーが FSA で指定したデータの長さが、そのデータに関して DBD で指定されている長さと異なっています。

C セグメント内のキー・フィールドを変更しようとしました。キー・フィールドの変更は許されません。

E FSA 内のデータの誤り。ユーザーが FSA で指定したデータが、そのデータに関して DBD で指定されているデータのタイプと異なっています。

F ユーザーが非所有セグメントを変更しようとしました。この状況コードは、関連 MSDB にだけ適用されます。

G ユーザーがデータ・フィールドを変更したときに算術オーバーフローが起こりました。

H セグメント内に要求されたフィールドがありませんでした。

FLD/VERIFY および FLD/CHANGE の使用例

ここでは、「銀行口座の例」データベースにある銀行口座セグメントを使用し、顧客が当座預金から 100 ドル引き出そうとしているところを想定しています。口座番号は 24056772 です。顧客がこの金額を引き出せるかどうか調べるには、現在残高を確認する必要があります。現在残高が 100 を超えていれば、残高から 100 ドルを引き、セグメント内の取引カウントに 1 を加えます。

以上のすべての処理を、1 つの FLD 呼び出しと 3 つの FSA によって行うことができます。この 3 つの FSA を以下に示します。

1. BALANCE フィールド内の値が 100 ドルよりも大きいかあるいは等しいかどうかを調べる。この検査を行うためには、BALANCE フィールド、「より大きいか等しい」かを表す H 演算子、および金額を指定します。この金額は、小数点なしで指定します。8 文字より短いフィールド名は、8 文字に等しくなるまで末尾にブランクを埋め込む必要があります。また、フィールド名と演算子の間に FSA 状況コードを収めるためのブランクを 1 つ入れなければなりません。この FSA は次のようになります。

   ```
   BALANCEbb
   H10000*
   ```

   この FSA の後には別の FSA が続くため、FSA 内の最後の文字はアスタリスクになっています。

2. 最初の FSA が正常に行われた場合には、BALANCE フィールドの値から 100 ドルを減算します。最初の FSA が正常に行われなかった場合には、IMS は処理を続けません。口座残高から引き出し額を減算するには、次の FSA を使用します。

   ```
   BALANCEbb
   -10000*
   ```

   ここでも、この FSA の後に 3 つ目の FSA が続くため、FSA 内の最後の文字はアスタリスクになっています。

3. この口座に関する取引カウントに 1 を加えます。このためには、次の FSA を用います。

   ```
   TRANCNTbb
   001b
   ```

   この FSA は呼び出しにおける最後の FSA であるため、FSA 内の最後の文字はブランク (b) になっています。

   この FLD 呼び出しを出すときには、各 FSA を個別に参照する必要はありません。すべての FSA が入っている入出力域を参照してください。

MSDB および DEDB におけるコミット・ポイント処理

ユーザーの既存のアプリケーション・プログラムは、MSDB コミット表示とデフォルトの DEDB コミット表示のどちらかを使用できます。

MSDB コミット表示

ユーザーが MSDB のセグメントを更新した場合、IMS はその更新をすぐには適用しません。ユーザーのプログラムがコミット・ポイントに達するまでは、更新は有効になりません。

更新の処理方法に応じた結果として、同じ呼び出しシーケンスでも、全機能データベースまたは DEDB に対して発行した場合と、MSDB に対して発行した場合とでは、異なる結果を受け取ることがあります。例えば、MSDB のセグメントに対して GHU および REPL 呼び出しを発行し、続いて、同一コミット・インターバル内でこれと同じセグメントに対し、別の Get 呼び出しを発行した場合、IMS が戻すセグメントは、「古い」値であり、更新済みの値ではありません。ただし、同じ呼び出しシーケンスを全機能データベースまたは DEDB のセグメントに対して発行した場合には、2 番目の Get 呼び出しは、更新済みのセグメントを戻します。

プログラムがコミット・ポイントに達すると、IMS は FLD VERIFY/CHANGE 呼び出しも処理します。VERIFY 検査が正常に行われた場合には、変更がデータベースに適用されます。VERIFY 検査が失敗した場合、前のコミット・ポイント以降に行われた変更は取り消され、トランザクションは処理し直されます。

MSDB コミット表示を使用する DEDB

DEDB 用の MSDB コミット表示を使用するには、PCB ステートメント上で VIEW=MSDB を指定します。VIEW=MSDB を指定しないと、DEDB は、デフォルトの DEDB コミット表示を使用します。したがって、ユーザーの MSDB を DEDB にマイグレーションする場合、既存のアプリケーション・プログラムを変更する必要はありません。

PCB に VIEW=MSDB を指定し、アプリケーション・プログラムが DEDB に対して GHU と REPL 呼び出しを出して、その後に同じコミット・インターバルでセグメント用に別の GHU を出すものと仮定します。この場合、アプリケーション・プログラムは REPL 呼び出しからそのデータの新しい値ではなく古い値を受け取ります。VIEW=MSDB を指定しない場合、DEDB あるいはその他の DL/I データベースから受け取るのと同じように、アプリケーション・プログラムはそのデータの新しく更新された値を受け取ります。

任意の DEDB PCB 用に VIEW=MSDB を指定できます。これが非 DEDB データベース用に指定された場合、ACBGEN 中にユーザーはメッセージ DFS0904 を受け取ります。

VIEW=MSDB を指定した PCB で REPL 呼び出しを発行する場合は、セグメントにはキーが必要です。この要件は、コマンド・コード「D」がパスに指定されているあらゆるセグメントに適用されます。それ以外の場合は AM 状況コードが戻されます。この状況コードについては、「*IMS V15* メッセージおよびコード 第 *4* 巻*: IMS* コンポーネント・コード」を参照してください。

次のコードは、VIEW オプションを指定した PCB の例です。

View=MSDB を指定する PCB の例

```
PCB  ,                                                  *00000100
     TYPE=DB,                                           *00000200
     NAME=DEDBJN21,                                     *00000300
     PROCOPT=A,                                         *00000400
     KEYLEN=30,                                         *00000500
     VIEW=MSDB,                                         *00000600
     POS=M                                               00000700
```

関連資料:

325 ページの『チェックポイントの発行』

400 ページの『DEDB におけるコミット・ポイント処理』

DEDB の処理 (IMS、および DBCTL を使用する CICS)

DEDB を処理するために、アプリケーション・プログラム内でサブセット・ポインター、副次索引、POS 呼び出し、データ・ロッキング、および処理オプションP と H を使用することができます。

サブセット・ポインター・コマンド・コードを使用した高速機能 DEDB の処理

サブセット・ポインターおよびそれとともに使用するコマンド・コードは、長いセグメント・チェーンを処理する必要があるときにユーザー・プログラムの効率を大幅に向上させる、最適化ツールです。

サブセット・ポインターは、同じ親のもとにあるセグメント・オカレンスのチェーンを 2 つ以上のグループ、すなわちサブセットに分割するための手段です。 1 つのセグメント・タイプについて、8 つまでのサブセット・ポインターを定義できます。そのうえで、アプリケーション・プログラム内でそのサブセット・ポインターを定義します。各サブセット・ポインターは、新しいサブセットの開始位置を指します。 例えば、次のトピックで、最後の 3 つのセグメント・オカレンスを最初の 4 つのセグメント・オカレンスから分けるために、1 つのサブセット・ポインターを定義したと仮定します。これにより、ユーザーのプログラムは、コマンド・コードを介してそのサブセット・ポインターを参照し、最後の 3 つのセグメント・オカレンスを直接にリトリーブできるようになります。

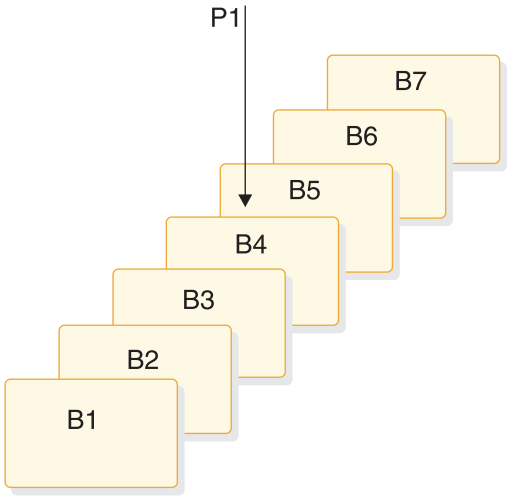


図 65. サブセット・ポインターを使用したセグメント・オカレンスの長いチェーンの処理

サブセット・ポインターは、ルート・レベルを除いて、データベース階層のどのレベルでも使用することができます。 ルート・レベルでサブセット・ポインターを使用しようとしても、無視されます。

次の図では、サブセット・ポインターを設定する方法を示しています。サブセット・ポインターは、互いに独立しているので、チェーンの中のどのセグメントに対しても 1 つ以上のポインターを設定することができます。例えば、次の図に示すように、1 つのセグメントに複数のサブセット・ポインターを設定できます。

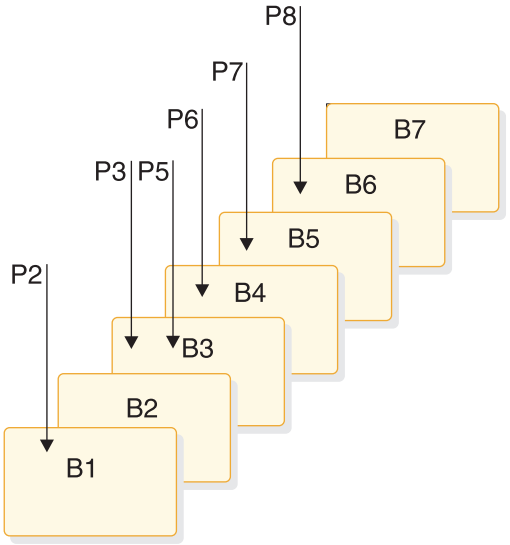


図 66. サブセット・ポインターの設定例

また、次の図に示すようにポインターとセグメントの間に 1 対 1 の関係を定義することもできます。

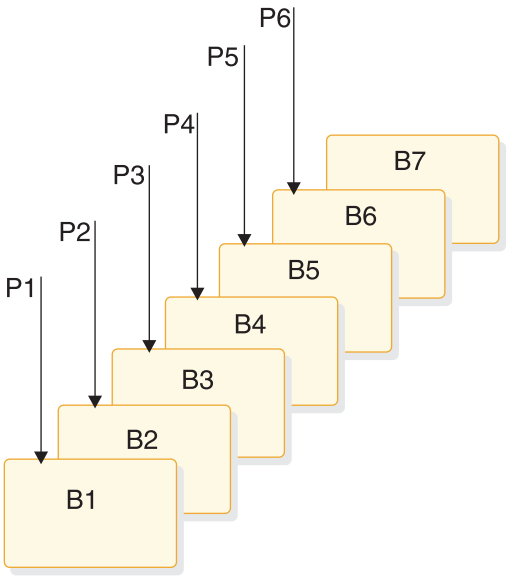


図 *67.* サブセット・ポインター設定の追加例

次の図は、同じ親の下にあるセグメント・オカレンスのチェーンが、サブセット・ポインターの使用によりどのように分けられるかを示しています。各サブセットはチェーン全体の最後のセグメントで終わります。 例えば、サブセット・ポインター 1 で定義されているサブセットの最終セグメントは、B7 です。

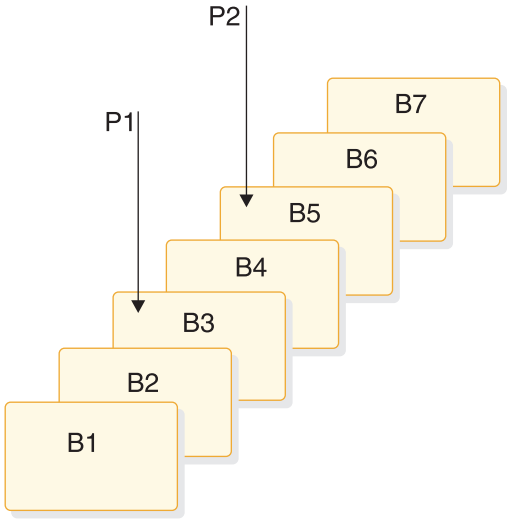


図 *68.* サブセット・ポインターがチェーンをサブセットに分ける方法

サブセット・ポインターを使用する前に

ユーザーのプログラムでサブセット・ポインターを使用するためには、 DEDB に関する DBD とユーザー・プログラムの PSB の中でポインターを定義しておかなければなりません。

- DBD には、1 つのセグメント・チェーンに使用されるポインター数を指定します。 どのセグメント・チェーンに対しても最高 8 つまでポインターを指定することができます。

- PSB では、ユーザーのプログラムがどのポインターを使用するかを指定します。これは、SENSEG ステートメントで定義してください。(各ポインターは 1 から 8 までの整数で定義します。) さらに、ユーザーのプログラムが、使用するポインターを設定できるかどうかを、SENSEG ステートメントで指示してください。ユーザーのプログラムが読取センシティビティーを備えている場合、このプログラムはポインターを設定することができず、すでに設定されているサブセット・ポインターによるセグメントのリトリーブだけを行うことができます。ユーザーのプログラムが更新センシティビティーを備えている場合、このプログラムは、S、W、M、および Z コマンド・コードを使用してサブセット・ポインターを更新することもできます。

DBD と PSB の中でポインターを定義した後、アプリケーション・プログラムは、ポインターをチェーンの中のセグメントに設定することができます。 アプリケーション・プログラムの実行が終わると、そのプログラムで使用されたサブセット・ポインターは、プログラムで設定されたときのままになり、リセットされません。

サブセット・ポインターの指定

プログラムの中でサブセット・ポインターを使用するためには、PSB の中で定義されたポインターの番号が分かっていなければなりません。サブセット・ポインター・コマンド・コードを使用するときには、使用したい各サブセット・ポインターの番号をコマンド・コードの後に続けて指定します。例えば、サブセット・ポインター 3 で定義されているサブセット内の最初のセグメント・オカレンスをリトリーブしたいことを示すためには、R3 を使用します。デフォルト値はありません。したがって、1 から 8 までの間の番号を指定しないと、IMS は SSA が無効であると見なし、AJ 状況コードを戻します。

サブセット・ポインター・コマンド・コード

サブセットを活用するために、アプリケーション・プログラムでは 5 つのコマンド・コードが使用されます。サブセット内の最初のセグメントをリトリーブしたい場合には、R コマンド・コードを使用します。「*IMS V15* アプリケーション・プログラミング *API*」のトピック『DL/I 用の DEDB コマンド・コード』で説明されている以下の 4 つのコマンド・コードは、サブセット・ポインターを修正することによってサブセットを再定義します。

Z サブセット・ポインターを、ゼロに設定します

M サブセット・ポインターを、現行セグメントの次のセグメントに設定します。

S 無条件で、サブセット・ポインターを現行セグメントに設定します。

W 条件付きで、サブセット・ポインターを現行セグメントに設定します。

プログラムがサブセット・ポインターを設定するためには、その前にデータベース内の位置を設定しておかなければなりません。 サブセット・ポインターを設定する前に、呼び出しは条件を完全に満たしている必要があります。ポインターが設定されるセグメントは、呼び出しの完了時におけるユーザーの現在位置によって異なります。セグメントをリトリーブするための呼び出しが条件を完全に満たしていない場合に、位置が設定されていないと、サブセット・ポインターは呼び出しが出される前のままになります。サブセット・ポインター・コマンド・コードは、非修飾

SSA でも修飾 SSA でも使用できます。非修飾 SSA を使用する呼び出しでコマンド・コードを使用するためには、セグメント名の後に、必要なサブセット・ポインターの番号を指定したコマンド・コードを、コーディングしてください。これは次の図に示されています。

表 *53.* サブセット・ポインター・コマンド・コードを使用した非修飾 *SSA*

セグメント名	*	コマンド・コード	サブセット・ポインター	b
8	1	可変	可変	1

修飾された SSA でサブセット・ポインター・コマンド・コードを使用するには、次の図に示すように、修飾ステートメントの左括弧の直前でコマンド・コードとサブセット・ポインター番号を使用します。

表 *54.* サブセット・ポインター・コマンド・コードを使用した修飾 *SSA*

セグメント名	*	コマンド・コード	サブセット・ポインター	(	フィールド名	R.O.	フィールド値	)
8	1	可変	可変	1	8	2	可変	1

サブセットへのセグメントの挿入

キーのないセグメントをサブセットに挿入するために R コマンド・コードを使用すると、サブセットの最初のセグメント・オカレンスの前に新しいセグメントが挿入されます。しかし、サブセット・ポインターが、新しいセグメント・オカレンスに自動的に設定されるわけではありません。

例えば、次の呼び出しは、新しい B セグメント・オカレンスをセグメント B5 の前に挿入しますが、サブセット・ポインター 1 をこの新しい B セグメント・オカレンスを指し示すようには設定しません。

```
ISRT   Abbbbbbb
(Akeybbbb=bA1)
Bbbbbbbb
*R1
```

サブセット・ポインター 1 を新しいセグメントに設定するには、次の例に示すように、R コマンド・コードとともに S コマンド・コードを使用します。

```
ISRT   Abbbbbbb
(Akeybbbb=bA1)
 Bbbbbbbb
*R1S1
```

このサブセットが存在しない場合 (サブセット・ポインター 1 が 0 に設定されている場合) には、セグメントはセグメント・チェーンの終わりに追加されます。

サブセット・ポインターに指示されているセグメントの削除

サブセット・ポインターにより指示されているセグメントを削除すると、サブセット・ポインターは、チェーンの中の次のセグメント・オカレンスを指示します。 削除したセグメントがチェーン内の最後のセグメントであった場合には、サブセット・ポインターは 0 に設定されます。

コマンド・コードの組み合わせ

相互に矛盾がない限り、S、M、および W コマンド・コードを他のコマンド・コードとともに使用することができ、サブセット・ポインター・コマンド・コードを相互に組み合わせることができます。例えば、R と S は一緒に使用することができますが、S と Z は機能が矛盾するため、一緒に使用できません。矛盾するコマンド・コードを組み合わせると、IMS は、ユーザーのプログラムに AJ 状況コードを戻します。

SSA ごとに 1 つの R コマンド・コードを使用することができ、またサブセット・ポインターごとに 1 つの更新コマンド・コード (Z、M、S、または W) を使用できます。

関連概念:

214 ページの『SSA とコマンド・コード』

402 ページの『DEDB に対する従属セグメントを使用した呼び出し』

サブセット・ポインターの状況コード

サブセット・ポインター・コマンド・コードを含む SSA でエラーが発生すると、IMS は、以下のいずれかの状況コードをプログラムに戻すことがあります。

AJ DBD 内でサブセット・ポインターが定義されていないセグメントに関して、SSA で R、S、Z、W、または M コマンド・コードを指定しました。

SSA に含まれているサブセット・コマンド・コードが矛盾しています。例えば、1 つの SSA で同じサブセット・ポインターに関して S コマンド・コードと Z コマンド・コードを指定すると、IMS は AJ 状況コードを戻します。S は、ポインターを現在位置に設定することを意味し、Z は、ポインターを 0 に設定することを意味します。これらのコマンド・コードを 1 つの SSA で使用することはできません。

SSA には複数の R コマンド・コードが含まれます。

サブセット・ポインター・コマンド・コードに続くポインター番号が無効です。番号が指定されなかったか、あるいは無効な文字が含まれています。コマンド・コードに続く番号は 1 から 8 までの間でなければなりません。

AM SSA で参照されたサブセット・ポインターが、プログラムの PSB で指定されていません。例えば、ユーザー・プログラムの PSB で、そのプログラムがサブセット・ポインター 1 および 4 を使用できることを指定した場合に、ユーザーの SSA がサブセット・ポインター 5 を参照していると、IMS は、AM 状況コードを戻します。

ユーザーのプログラムが、ポインターを更新するコマンド・コード (S、W、または M) を使用しようとしましたが、プログラムの PSB でポインター更新センシティビティーが指定されていません。

ご使用のプログラムは、IOAREA を指定せずに GSAM データベースを開こうとしました。

副次索引を使用した DEDB の処理

HISAM または SHISAM のいずれかのデータベース構造の場合、アプリケーション・プログラムで DEDB データベースの副次索引を処理することができます。

HISAM 副次索引データベースまたは SHISAM 副次索引データベースでは、順次キーの副次索引をサポートしています。

順次従属 (SDEP) セグメントを使用した DEDB データベースでは副次索引データベースを使用できます。SDEP セグメントを索引フィールドとして使用することはできません。そのため、SDEP セグメントでは、その SEGM ステートメントで LCHILD または XDFLD ステートメントを定義することはできません。SDEP セグメントは一時データであり、SDEP SCAN および SDEP DELETE ユーティリティーを使用して削除されるので、SDEP セグメントに対する高速機能副次索引サポートは制限されます。つまり、SDEP セグメントを副次索引データベースのターゲット・セグメントまたはソース・セグメントにすることはできません。ターゲット・セグメントがルート・セグメントの場合、代替シーケンスを介してアクセスされる DEDB データベースについては SDEP セグメントを返すことができます。

高速機能副次索引は、ユニーク・キーと非ユニーク・キーの両方をサポートしています。HISAM 副次索引データベースは、ユニーク・キーと非ユニーク・キーの両方をサポートし、SHISAM 副次索引はユニーク・キーのみをサポートしています。

HISAM 副次索引データベースは、ユニーク・キーと非ユニーク・キーの両方をサポートしています。HISAM 副次索引データベースの場合、非ユニーク・キーのサポートは ESDS オーバーフロー・データ・セットを使用して提供されます。重複キーは、後入れ先出し (LIFO) 順で保管されます。最初に挿入された重複キーが KSDS データ・セットに保管され、残りの重複キーは ESDS オーバーフロー・データ・セットに LIFO 順で保管されます。

ターゲット・セグメントは基本 DEDB データベースに入っています。ターゲット・セグメントは、アプリケーション・プログラムがリトリーブする必要のあるセグメントです。ターゲット・セグメントは、基本 DEDB データベースの 15 のレベルのうち、どのレベルにもなります。SDEP セグメントは、副次索引データベースのターゲット・セグメントまたはソース・セグメントにすることはできません。

1 セグメント当たり最大 32 の副次索引があり、1 DEDB データベース当たり最大 255 の副次索引があります。

高速機能副次索引データベースには、以下としてアクセスできます。

- 専用のデータベース。
- 基本 DEDB データベースへの副次索引。オプションで単一 (デフォルト) または複数 (DEDB 実装の場合のみ)の副次索引セグメントを使用できます。
- 高速機能副次索引ユーザー区画。
- BMP アプリケーションの索引保守を抑止するためのオプション。
- 高速機能副次索引ユーザー区画データベースに 1 つの別個の論理データベースとしてアクセスするためのオプション。

高速機能副次索引は、完全機能の DL/I 呼び出しと同様のブール修飾をサポートします。サポートされるブール演算子は、以下のとおりです。

- 論理 AND (* または & でコーディング)
- 論理 OR (+ または | でコーディング)

制約事項

副次索引を使用した DEDB の処理に適用される制約事項は以下のとおりです。

- 順次従属 (SDEP) セグメントを使用した DEDB データベースでは副次索引データベースを使用できますが、SDEP セグメントを索引フィールドとして使用することはできません。そのため、SDEP セグメントでは、その SEGM ステートメントで LCHILD または XDFLD ステートメントを定義することはできません。SDEP セグメントは一時データであり、SDEP SCAN および SDEP DELETE ユーティリティーを使用して削除されるので、SDEP セグメントに対する高速機能副次索引サポートは制限されます。つまり、SDEP セグメントを副次索引データベースのターゲット・セグメントまたはソース・セグメントにすることはできません。
- 高速機能副次索引では、共用副次索引をサポートしていません。複数の副次索引セグメントがサポートされるのは DEDB 実装の場合のみで、共用副次索引サポートの場合とは異なります。
- 高速機能副次索引データベースは、シンボリック・ポインターのみをサポートします。直接ポインター・サポートはありません。索引先セグメントにシンボリック・ポインターを使用しても、基本 DEDB データベースの再編成時に高速機能副次索引データベースに影響を与えることはありません。
- PROCSEQD= パラメーターを指定した PCB を使用して、副次索引により基本 DEDB データベースにアクセスする場合、「ターゲット = ルート・セグメント」の主キー・フィールドを使用した SSA の修飾 GU/GN セグメント名がサポートされています。
- 「ターゲット = 従属セグメント」の主キー・フィールドを使用した SSA の修飾 GU/GN セグメント名はサポートされていません。PROCSEQD= パラメーターを指定した PCB を使用して副次索引により基本 DEDB データベースにアクセスした場合、修飾 GET 呼び出しに対して AC 状況コードが返されます。
- 独立 AND (#) ブール演算子は、サポートされません。
- XDFLD を使用する SSA およびターゲット・セグメントからのフィールドでは、ブールはサポートされません。ブールのサポートは、XDFLD に対してのみです。

高速機能副次索引を使用した基本 DEDB データベースへのアクセス: 例 1

GU および GN DL/I 呼び出しを使用して副次索引により基本 DEDB データベース上の COURSE セグメントにアクセスするための DL/I 呼び出し。わかりやすくするために、各従属セグメントには 1 つだけセグメント・インスタンスがあるものとします。

PCB2NDX は、高速機能副次索引データベース NAMESXDB を使用するように PROCSEQD= パラメーターが定義された PCB です。COURSE セグメントは、ターゲット・セグメントであり、かつルート・セグメントです。ソース・セグメントは、ターゲット・セグメントと同じです。

```
PCB2NDX
 PCB TYPE=DB,DBDNAME=EDUCDB,PROCOPT=GR,KEYLEN=100,
     PROCSEQD=NAMESXDB
 SENSEG   NAME=COURSE,PARENT=0      <<- (target seg=root)
 SENSEG   NAME=CLASS,PARENT=COURSE
```

```
 SENSEG   NAME=INSTRUCT,PARENT=CLASS
 SENSEG   NAME=STUDENT,PARENT=CLASS
 PSBGEN   PSBNAME=NAMEXPSB,LANG=COBOL
 END
PCB   PCB2NDX
GU    COURSE(NAMEINDX=CHEMISTRY)
GN    COURSE
```

GU COURSE は、副次索引キー NAMEINDX=CHEMISTRY を使用して、基本 DEDB データベース内の CHEMISTRY に関する COURSE セグメントを返します。

ポインター・セグメントのキー CHEMISTRY が、キー・フィードバック域に返されます。

GN COURSE は、高速機能副次索引データベース NAMESXDB 内でセグメント CHEMISTRY の後にある次のポインター・セグメントが指している、基本 DEDB データベースの COURSE セグメントを返します。

高速機能副次索引データベース NAMESXDB 内で CHEMISTRY の後にある次のポインター・セグメントのキー (副次索引キー CHEMISTRY の後にある次の順次キー) が、キー・フィードバック域に返されます。

高速機能副次索引を使用した基本 DEDB データベースへのアクセス: 例 2

GU および GN DL/I 呼び出しを使用して副次索引により基本 DEDB データベース上の COURSE セグメントにアクセスするための DL/I 呼び出し。わかりやすくするために、各従属セグメントには 1 つだけセグメント・インスタンスがあるものとします。

PCB2NDX は、高速機能副次索引データベース NAMESXDB を使用するように PROCSEQD= パラメーターが定義された PCB です。COURSE セグメントは、ターゲット・セグメントであり、かつルート・セグメントです。ソース・セグメントは、ターゲット・セグメントと同じです。

```
PCB   PCB2NDX
GU    COURSE(NAMEINDX=CHEMISTRY)
GN                                    1st GN
GN                                    2nd GN
GN                                    3rd GN
GN                                    4th GN
```

GU COURSE は、副次索引キー NAMEINDX=CHEMISTRY を使用して、基本 DEDB データベース内の CHEMISTRY に関する COURSE セグメントを返します。

ポインター・セグメントのキー CHEMISTRY が、キー・フィードバック域に返されます。

最初の GN 呼び出しは、基本 DEDB データベース内の CHEMISTRY COURSE セグメントの DEDB 反転構造のセグメントを返します。COURSE セグメントはターゲット・セグメントであり、かつルート・セグメントであるため、物理構造内のすべてのセグメントは PCB PCB2INDX に定義された方法でアクセス可能です。

GN は、基本 DEDB データベース内の GU 呼び出しによってリトリーブされた CHEMISTRY COURSE セグメントの下のデータベース・レコードの CLASS セグメントを返します。

ポインター・セグメントのキー CHEMISTRY が、CHEMISTRY COURSE セグメントの下の CLASS セグメントのキーと連結され、キー・フィードバック域に返されます。

2 番目の GN 呼び出しは、CHEMISTRY COURSE セグメントの下のデータベース・レコードの INSTRUCT セグメントを返します。

ポインター・セグメントのキー CHEMISTRY が、CHEMISTRY COURSE セグメントの下の CLASS セグメントのキーおよび INSTRUCT セグメントのキーと連結されて、キー・フィードバック域に返されます。

3 番目の GN 呼び出しは、CHEMISTRY COURSE セグメントの下のデータベース・レコードの STUDENT セグメントを返します。

ポインター・セグメントのキー CHEMISTRY が、CHEMISTRY COURSE セグメントの下の CLASS セグメントのキーおよび STUDENT セグメントのキーと連結されて、キー・フィードバック域に返されます。

STUDENT セグメントは基本 DEDB データベースの COURSE データベース・レコードの最後のセグメントであるため、4 番目の GN 呼び出しは、副次索引データベース NAMESXDB 内で CHEMISTRY セグメントの後にある次の副次索引キーを使用して、基本 DEDB データベースの COURSE セグメントを返します。

高速機能副次索引データベース NAMESXDB 内で CHEMISTRY の後にある次のポインター・セグメントのキー (副次索引キー CHEMISTRY の後にある次の順次キー) が、キー・フィードバック域に返されます。

高速機能副次索引を使用した基本 DEDB データベースへのアクセス: 例 3

GU および GNP DL/I 呼び出しを使用して副次索引により基本 DEDB データベース上の COURSE セグメントにアクセスするための DL/I 呼び出し。わかりやすくするために、各従属セグメントには 1 つだけセグメント・インスタンスがあるものとします。

PCB2NDX は、高速機能副次索引データベース NAMESXDB を使用するように PROCSEQD= パラメーターが定義された PCB です。COURSE セグメントは、ターゲット・セグメントであり、かつルート・セグメントです。ソース・セグメントは、ターゲット・セグメントと同じです。

```
PCB   PCB2NDX
GU    COURSE(NAMEINDX=CHEMISTRY)
GNP                                    1st GNP
GNP                                    2nd GNP
GNP                                    3rd GNP
GNP                                    4th GNP
```

GU COURSE は、副次索引キー NAMEINDX=CHEMISTRY を使用して、基本 DEDB データベース内の CHEMISTRY に関する COURSE セグメントを返します。

ポインター・セグメントのキー CHEMISTRY が、キー・フィードバック域に返されます。

最初の GNP 呼び出しは、基本 DEDB データベース内の CHEMISTRY COURSE セグメントの DEDB 反転構造の下にある最初のセグメントを返します。COURSE セグメントはターゲット・セグメントであり、かつルート・セグメントであるため、物理構造内のすべてのセグメントは PCB2INDX で定義された方法でアクセス可能です。 GNP は、基本 DEDB データベース内の GU 呼び出しによってリトリーブされた CHEMISTRY COURSE セグメントの下のデータベース・レコードの CLASS セグメントを返します。

ポインター・セグメントのキー CHEMISTRY が、CHEMISTRY COURSE セグメントの下の CLASS セグメントのキーと連結され、キー・フィードバック域に返されます。

2 番目の GNP 呼び出しは、CHEMISTRY COURSE セグメントの下のデータベース・レコードの INSTRUCT セグメントを返します。

ポインター・セグメントのキー CHEMISTRY が、CHEMISTRY COURSE セグメントの下の CLASS セグメントのキーおよび INSTRUCT セグメントのキーと連結されて、キー・フィードバック域に返されます。

3 番目の GNP 呼び出しは、CHEMISTRY COURSE セグメントの下のデータベース・レコードの STUDENT セグメントを返します。

ポインター・セグメントのキー CHEMISTRY が、CHEMISTRY COURSE セグメントの下の CLASS セグメントのキーおよび STUDENT セグメントのキーと連結されて、キー・フィードバック域に返されます。

STUDENT セグメントは基本 DEDB データベースの COURSE データベース・レコードの最後のセグメントであるため、4 番目の GNP 呼び出しは GE 状況コードを返します。

高速機能副次索引を使用した基本 DEDB データベースへのアクセス: 例 4

GU および GN DL/I 呼び出しを使用して副次索引により基本 DEDB データベース上の INSTRUCT セグメントにアクセスするための DL/I 呼び出し。わかりやすくするために、各従属セグメントには 1 つだけセグメント・インスタンスがあるものとします。

PCB3NDX は、高速機能副次索引データベース INSTSXDB を使用するように PROCSEQD= パラメーターが定義された PCB です。CLASS セグメントは、ターゲット・セグメントであり、ルート・セグメントではありません。ソース・セグメント (INSTRUCT セグメント) は、ターゲット・セグメント (CLASS セグメント) と同じではありません。

CLASS SEGM ステートメントには複数の LCHILD/XDFLD ペアがありますが、この例では 1 つのみを使用します。

EDUCDB DEDB DBD 内の CLASS および INSTRUCT SEGM ステートメントの DBDGEN の抜粋:

```
SEGM   NAME=CLASS,BYTES=50,PARENT=COURSE
FIELD  NAME=(CLASSNO,SEQ,U),BYTES=4,START=7
FIELD  NAME=CLASNAME,BYTES=10,START=15

LCHILD NAME=(CLASXSEG,CLASSCDB),PTR=SYMB
XDFLD  NAME=CLASINDX,SRCH=CLASNAME

LCHILD NAME=(INSTXSEG,INSTSXDB),PTR=SYMB
XDFLD  NAME=INSTINDX,SEGMENT=INSTRUCT,SRCH=INSTNAME
SEGM   NAME=INSTRUCT,BYTES=50,PARENT=CLASS
FIELD  NAME=(INSTNO,SEQ,U),BYTES=6,START=1
FIELD  NAME=INSTPHNO,BYTES=10,START=11
FIELD  NAME=INSTNAME,BYTES=20,START=21

...
```

PCB3NDX の PSBGEN 定義:

ターゲット・セグメントがルート・セグメントではない場合、ルート・セグメントからのターゲット・セグメントの直接の親をすべて、PROCSEQD パラメーターを使用して PCB で定義する必要があります。ターゲット・セグメントがルート・セグメントではない場合は、ルート・セグメントからターゲット・セグメントおよびそのターゲット・セグメントのすべての子セグメントへの物理パスに沿って、直接の親のみにアクセスすることができます。CLASS のすべての兄弟セグメントはアクセス不可能です。セグメントが常に論理シーケンスで (例えば、ターゲットまたは論理ルートから物理ルートへと) リトリーブされる場合でも、必須の SENSEG のコーディング・シーケンスは、セグメントの物理パスの (例えば、物理ルートからターゲットへの) シーケンス内になければなりません。

```
PCB3NDX
 PCB TYPE=DB,DBDNAME=EDUCDB,PROCOPT=GR,KEYLEN=100,
     PROCSEQD=INSTSXDB
 SENSEG   NAME=COURSE,PARENT=0          <<- mandatory SENSEG
 SENSEG   NAME=CLASS,PARENT=COURSE      <<- mandatory SENSEG (target seg)
 SENSEG   NAME=INSTRUCT,PARENT=CLASS    <<- optional SENSEG
 PSBGEN   PSBNAME=NAMEXPSB,LANG=COBOL
 END

 PCB   PCB3NDX
 GU    CLASS (INSTINDX=TOMJONES)
 GN                                     1st GN
 GN                                     2nd GN
 GN                                     3rd GN
```

GU CLASS は、副次索引キー INSTINDX=TOMJONES を使用して、基本 DEDB データベース内の、クラスを教える講師に関する CLASS セグメントを返します。

ポインター・セグメントのキー TOMJONES が、キー・フィードバック域に返されます。

最初の GU 呼び出しは、基本 DEDB データベースのターゲット・セグメントを返します。ターゲット・セグメント (CLASS) はルート・セグメントではないため、後続の GN は、基本 DEDB データベース内の GU 呼び出しによってリトリーブ

された CLASS セグメントの DEDB 反転構造内での順番が次のセグメントを返します。例えば、GN が返す COURSE セグメントは、直接の物理親であり、DEDB 反転構造内の CHEMISTRY COURSE セグメントを教える CLASS セグメントの論理子でもあります。

ポインター・セグメントのキー TOMJONES が、COURSE セグメントのキーと連結されて、キー・フィードバック域に返されます。

2 番目の GN 呼び出しは、データベース・レコードの INSTRUCT セグメントを返します。これは CLASS セグメントの論理子で、DEDB 反転構造内の COURSE セグメントの論理兄弟です。

ポインター・セグメントのキー TOMJONES が、INSTRUCT セグメントのキーと連結されて、キー・フィードバック域に返されます。

PCB PCB3NDX の INSTRUCT セグメントには子セグメントも兄弟セグメントも定義されていないため、3 番目の GN 呼び出しは、副次索引データベース内で INSTINDX=TOMJONES の後にある次のセグメントを使用して、基本 DEDB データベースの CLASS セグメントを返します。

高速機能副次索引データベース INSTSXDB 内で TOMJONES の後にある次のポインター・セグメントのキー (副次索引キー TOMJONES の後にある次の順次キー) が、キー・フィードバック域に返されます。

高速機能副次索引を使用した基本 DEDB データベースへのアクセス: 例 5

GU および GNP DL/I 呼び出しを使用して副次索引により基本 DEDB データベース上の INSTRUCT セグメントにアクセスするための DL/I 呼び出し。わかりやすくするために、各従属セグメントには 1 つだけセグメント・インスタンスがあるものとします。

PCB3NDX は、高速機能副次索引データベース INSTSXDB を使用するように PROCSEQD= パラメーターが定義された PCB です。CLASS セグメントは、ターゲット・セグメントであり、ルート・セグメントではありません。ソース・セグメント (INSTRUCT セグメント) は、ターゲット・セグメント (CLASS セグメント) と同じではありません。

CLASS SEGM ステートメントには複数の LCHILD/XDFLD ペアがありますが、この例では 1 つのみを使用します。

EDUCDB DEDB DBD 内の CLASS および INSTRUCT SEGM ステートメントの DBDGEN の抜粋:

```
...
SEGM   NAME=CLASS,BYTES=50,PARENT=COURSE
FIELD  NAME=(CLASSNO,SEQ,U),BYTES=4,START=7)
FIELD  NAME=CLASNAME,BYTES=10,START=15

LCHILD NAME=(CLASXSEG,CLASSXDB),PTR=SYMB
XDFLD  NAME=CLASINDX,SRCH=CLASNAME

LCHILD NAME=(INSTXSEG,INSTSXDB),PTR=SYMB
XDFLD  NAME=INSTINDX,SEGMENT=INSTRUCT,SRCH=INSTNAME
```

```
SEGM   NAME=INSTRUCT,BYTES=50,PARENT=CLASS
FIELD  NAME=(INSTNO,SEQ,U),BYTES=6,START=1
FIELD  NAME=INSTPHNO,BYTES=10,START=11
FIELD  NAME=INSTNAME,BYTES=20,START=21

...
```

PCB3NDX の PSBGEN 定義:

ターゲット・セグメントがルート・セグメントではない場合、ルート・セグメントからのターゲット・セグメントの直接の親をすべて、PROCSEQD パラメーターを使用して PCB で定義する必要があります。ターゲット・セグメントがルート・セグメントではない場合は、ルート・セグメントからターゲット・セグメントおよびそのターゲット・セグメントのすべての子セグメントへの物理パスに沿って、直接の親のみにアクセスすることができます。

```
PCB3NDX
 PCB TYPE=DB,DBDNAME=EDUCDB,PROCOPT=GR,KEYLEN=100,
     PROCSEQD=INSTSXDB
 SENSEG   NAME=COURSE,PARENT=0
 SENSEG   NAME=CLASS,PARENT=COURSE      <<-- Target segment
 SENSEG   NAME=INSTRUCT,PARENT=CLASS
 PSBGEN   PSBNAME=NAMEXPSB,LANG=COBOL
 END

 PCB   PCB3NDX
 GU    CLASS(INSTINDX=TOMJONES)
 GNP                                    1st GNP
 GNP                                    2nd GNP
 GNP                                    3rd GNP
```

GU CLASS は、副次索引キー INSTINDX=TOMJONES を使用して、基本 DEDB データベース内の特定のクラスを教える講師に関する CLASS セグメントを返します。

ポインター・セグメントのキー TOMJONES が、キー・フィードバック域に返されます。

最初の GU 呼び出しは、DEDB の CLASS セグメントを返します。ターゲット・セグメント (CLASS) はルート・セグメントではないため、最初の GNP は、DEDB 反転構造内での順番が次のセグメントを返します。例えば、最初の GNP が返す COURSE セグメントは、DEDB 反転構造内の CLASS セグメントの直接の物理親で、直接の論理子です。

ポインター・セグメントのキー TOMJONES が、COURSE セグメントのキーと連結されて、キー・フィードバック域に返されます。

2 番目の GNP 呼び出しは、データベース・レコードの INSTRUCT セグメントを返します。これは、CLASS の論理子で、DEDB 反転構造階層では COURSE セグメントの論理兄弟でもあります。

ポインター・セグメントのキー TOMJONES が、INSTRUCT セグメントのキーと連結されて、キー・フィードバック域に返されます。

PCB PCB3NDX の INSTRUCT セグメントには子セグメントも兄弟セグメントも定義されていないので、副次索引キー TOMJONES を持つ CLASS セグメントの下に

ある 3 番目の GNP 呼び出しは、GE 状況コードを返します。

高速機能副次索引を使用した基本 DEDB データベースへのアクセス: 例 6

GU および GN DL/I 呼び出しを使用して副次索引により基本 DEDB データベース上の CLASS セグメントにアクセスするための DL/I 呼び出し。わかりやすくするために、各従属セグメントには 1 つだけセグメント・インスタンスがあるものとします。

PCB4NDX は、高速機能副次索引データベース CLASSXDB を使用するように PROCSEQD= パラメーターが定義された PCB です。CLASS セグメントは、ターゲット・セグメントであり、ルート・セグメントではありません。ソース・セグメントは、ターゲット・セグメントと同じです。

PCB4NDX の PSBGEN 定義:

ターゲット・セグメントがルート・セグメントではない場合、ルート・セグメントからのターゲット・セグメントの直接の親をすべて、PROCSEQD パラメーターを使用して PCB で定義する必要があります。ターゲット・セグメントがルート・セグメントではない場合は、ルート・セグメントからターゲット・セグメントおよびそのターゲット・セグメントのすべての子セグメントへの物理パスに沿って、直接の親のみにアクセスすることができます。

```
PCB4NDX
 PCB TYPE=DB,DBDNAME=EDUCDB,PROCOPT=GR,KEYLEN=100,
     PROCSEQD=CLASSXDB
 SENSEG   NAME=COURSE,PARENT=0
 SENSEG   NAME=CLASS,PARENT=COURSE    <<-- Target segment
 SENSEG   NAME=INSTRUCT,PARENT=CLASS
 SENSEG   NAME=STUDENT,PARENT=CLASS
 PSBGEN   PSBNAME=NAMEXPSB,LANG=COBOL
 END

 PCB   PCB4NDX
 GU    CLASS(CLASINDX=CHEM1A)
 GN                                    1st GN
 GN                                    2nd GN
 GN                                    3rd GN
 GN                                    4th GN
```

GU CLASS は、副次索引キー INSTINDX=CHEM1Aを使用して、基本 DEDB データベース内のクラス名 CHEM1A に関する CLASS セグメントを返します。

ポインター・セグメントのキー CHEM1A が、キー・フィードバック域に返されます。

最初の GN 呼び出しは、基本 DEDB データベース内の CHEM1A CLASS セグメントの DEDB 反転構造の COURSE セグメントを返します。CLASS セグメントはターゲット・セグメントであり、ルート・セグメントではないため、GN は、基本 DEDB データベース内の GU 呼び出しによってリトリーブされた CLASS セグメントの DEDB 反転構造内での順番が次のセグメントを返します。GN が CHEMISTRY の場合に返す COURSE セグメントは、DEDB 反転構造の CHEM1A CLASS セグメントの直接の親です。

ポインター・セグメントのキー CHEM1A が、COURSE セグメントの主キーと連結されて、キー・フィードバック域に返されます。

2 番目の GN 呼び出しは、CHEM1A CLASS セグメントの下のデータベース・レコードの INSTRUCT セグメントを返します。

ポインター・セグメントのキー CHEM1A が、CHEM1A CLASS セグメントの下の INSTRUCT セグメントのキーと連結されて、キー・フィードバック域に返されます。

3 番目の GN 呼び出しは、CHEM1A CLASS セグメントの下のデータベース・レコードの STUDENT セグメントを返します。

ポインター・セグメントのキー CHEM1A が、CHEM1A CLASS セグメントの下の STUDENT セグメントのキーと連結されて、キー・フィードバック域に返されます。

STUDENT セグメントはデータベース・レコードの CLASS セグメントの最後のセグメントであるため、4 番目の GN 呼び出しは、副次索引データベース CLASINDX 内で CHEM1A セグメントの後にある次の副次索引キーを使用して、基本 DEDB データベースの CLASS セグメントを返します。

高速機能副次索引データベース CLASSXDB 内で CHEM1A の後にある次のポインター・セグメントのキー (副次索引キー CHEM1A の後にある次の順次キー) が、キー・フィードバック域に返されます。

高速機能副次索引を使用した基本 DEDB データベースへのアクセス: 例 7

GU および GN DL/I 呼び出しを使用して副次索引により基本 DEDB データベース上のクラス名にアクセスするための DL/I 呼び出し。わかりやすくするために、各従属セグメントには 1 つだけセグメント・インスタンスがあるものとします。

PCB4NDX は、高速機能副次索引データベース CLASSXDB を使用するように PROCSEQD= パラメーターが定義された PCB です。CLASS セグメントは、ターゲット・セグメントであり、ルート・セグメントではありません。ソース・セグメントは、ターゲット・セグメントと同じです。

```
PCB   PCB4NDX
GU    CLASS(CLASINDX=CHEM1A)
GN    CLASS
```

GU CLASS は、副次索引キー NAMEINDX=CHEM1A を使用して、基本 DEDB データベース内の CHEM1A に関する CLASS セグメントを返します。

ポインター・セグメントのキー CHEM1A が、キー・フィードバック域に返されます。

GN CLASS は、高速機能副次索引データベース CLASSXDB 内でセグメント CHEM1A の後にある次のポインター・セグメントが指している、基本 DEDB データベースの CLASS セグメントを返します。

高速機能副次索引データベース CLASSXDB 内で CHEM1A の後にある次のポインター・セグメントのキー (副次索引キー CHEM1A の後にある次の順次キー) が、キー・フィードバック域に返されます。

高速機能副次索引を使用した基本 DEDB データベースへのアクセス: 例 8

GU および GN DL/I 呼び出しと C コマンド・コードを使用して副次索引により基本 DEDB データベース上のクラス名にアクセスするための DL/I 呼び出し。わかりやすくするために、各従属セグメントには 1 つだけセグメント・インスタンスがあるものとします。

PCB4NDX は、高速機能副次索引データベース CLASSXDB を使用するように PROCSEQD= パラメーターが定義された PCB です。CLASS セグメントは、ターゲット・セグメントであり、ルート・セグメントではありません。ソース・セグメントは、ターゲット・セグメントと同じです。

```
PCB   PCB4NDX
GU    INSTRUCT *C (CHEM1AI12345)
GN                                    1st GN
GN                                    2nd GN
```

GU INSTRUCT は、クラス名 CHEM1A の CLASS セグメントの下にある講師番号 I12345 に関する INSTRUCT セグメントを返します。

ポインター・セグメントのキー CHEM1A が、INSTRUCT セグメントのキー I12345 と連結されて、キー・フィードバック域に返されます。

最初の GN 呼び出しは、基本 DEDB データベース内の CHEM1A CLASS セグメントの DEDB 反転構造の STUDENT セグメントを返します。CLASS セグメントはターゲット・セグメントであり、ルート・セグメントではないため、GN は、基本 DEDB データベース内の GU 呼び出しによってリトリーブされた INSTRUCT セグメントの DEDB 反転構造内での順番が次のセグメントを返します。GN が返す STUDENT セグメントは、DEDB 反転構造の CHEM1A CLASS セグメントの子セグメントです。

ポインター・セグメントのキー CHEM1A が、STUDENT セグメントのキーと連結されて、キー・フィードバック域に返されます。

STUDENT セグメントはデータベース・レコードの CLASS セグメントの最後のセグメントであるため、2 番目の GN 呼び出しは、副次索引データベース CLASINDX 内で CHEM1A セグメントの後にある次の副次索引キーを使用して、基本 DEDB データベースの CLASS セグメントを返します。

高速機能副次索引データベース CLASSXDB 内で CHEM1A の後にある次のポインター・セグメントのキー (副次索引キー CHEM1A の後にある次の順次キー) が、キー・フィードバック域に返されます。

高速機能副次索引を使用した基本 DEDB データベースへのアクセス: 例 9

副次索引キー CHEM1A を持つ CLASS セグメントの副次索引を使用して、基本 DEDB データベースに INSTRUCT セグメントを挿入する DL/I 呼び出し。

PCB4NDX は、高速機能副次索引データベース CLASSXDB を使用するように PROCSEQD= パラメーターが定義された PCB です。CLASS セグメントは、ターゲット・セグメントであり、ルート・セグメントではありません。ソース・セグメントは、ターゲット・セグメントと同じです。

```
PCB4INDX
ISRT CLASS(CLASINDX = CHEM1A)
     INSTRUCT I23456 JOHN SMITH
```

この ISRT 呼び出しは、副次索引キー CHEM1A を持つ CLASS セグメントの下に、キー I23456 を持つ INSTRUCT セグメントを挿入します。

ポインター・セグメントのキー CHEM1A が、INSTRUCT セグメントのキー I23456 と連結されて、キー・フィードバック域に返されます。

関連概念:

副次索引の作成 (データベース管理)

関連タスク:

DEDB への副次索引の追加 (データベース管理)

POS 呼び出しを使用した位置のリトリーブ (DEDB のみ)

POS (位置) 呼び出しを使用して、特定の順次従属セグメントの位置のリトリーブ、最後に挿入された順次従属セグメントの位置、そのタイム・スタンプ、および IMS ID のリトリーブ、あるいは順次従属セグメントまたは Logical Begin のタイム・スタンプのリトリーブを行います。また、POS 呼び出しを使用して、各 DEDB エリア内の未使用スペースの量を通知することもできます。例えば、 POS 呼び出しに対応して IMS が戻した情報を使用して、特定の期間に関する順次従属セグメントをスキャンまたは削除できます。

「*IMS V15* アプリケーション・プログラミング *API*」のトピック『POS 呼び出し』は、POS 呼び出しのコーディング方法、および POS 呼び出しのための入出力域の様子を示しています。POS 呼び出しで指定された区域が使用不能な場合には、入出力域は変更されず、FH 状況コードが戻されます。

特定の順次従属セグメントの位置指定

あるルート・セグメントに位置指定されているユーザーは、そのルートの特定順次従属セグメントの位置情報および区域名をリトリーブできます。順次従属セグメント上に位置を設定した場合には、検索はその位置から始まります。 IMS は、呼び出しの条件を満たした最初の順次従属セグメントの位置情報を戻します。この情報をリトリーブするには、順次従属セグメントのセグメント名を含む修飾された SSA または非修飾 SSA を指定して、POS 呼び出しを出してください。この種の POS 呼び出しの後の現在位置は、GNP 呼び出しの後の位置と同じになります。

POS 呼び出しが成功した後の入出力域の内容は、以下のとおりです。

LL　入出力域内のデータ全体の長さを示す 2 バイトのフィールド (2 進数)

区域名
　AREA ステートメントで指定された DD 名を示す 8 バイトのフィールド

位置　要求されたセグメントに対する位置情報が入っている 8 バイトのフィールド

例外: POS 呼び出しの目標である順次従属セグメントが同じ同期インターバルで挿入された場合には、位置情報は戻されません。バイト 11 から 18 には X'FF' が入っています。その他のフィールドには、通常のデータが入ります。

未使用の **CI**
順次従属部分にある未使用の CI 数が入っている 4 バイトのフィールド

未使用の **CI**
独立したオーバーフロー部分にある未使用の CI 数が入っている 4 バイトのフィールド

最後に挿入された順次従属セグメントの位置指定

特定のルート・セグメントの順次従属セグメントの中で、最後に挿入されたセグメントの位置情報をリトリーブすることもできます。これを行うためには、ルート・セグメントをセグメント名として含む非修飾または修飾された SSA を指定して、POS 呼び出しを出してください。このタイプの呼び出しの後の現在位置は、GU 呼び出しの後の位置と同じ規則に従います。

さらに、SDEP の位置、そのタイム・スタンプ、そのセグメントを所有する IMS の ID をリトリーブすることもできます。これを行うためには、修飾された SSA を指定して POS 呼び出しを発行し、入出力域の位置 1 に、POS 呼び出しへの入力としてキーワード PCSEGTSP を指定してください。このキーワードは、SDEP の位置、そのタイム・スタンプ、およびそのセグメントを所有する IMS の ID を戻すことを POS 呼び出しに要求します。

要件: 戻される追加データ用に、入出力域のサイズを 42 バイトに増やす必要があります。入出力域には、次の表に示されていない 2 バイトの LL フィールドが含まれています。この LL フィールドは、表の後に説明されています。

表 *55.* 修飾 *POS* 呼び出し*:* キーワードと戻される入出力域のマップ

キーワード	ワード **0**	ワード **1**	ワード **2**	ワード **3**	ワード **4**	ワード **5**	ワード **6**	ワード **7**	ワード **8**	ワード **9**
<null>	フィールド 1		フィールド 2		フィールド 3	フィールド 4	N/A		N/A	
PCSEGTSP	フィールド 1		フィールド 2		フィールド 5		フィールド 6		フィールド 7	

表 *55.* 修飾 *POS* 呼び出し: キーワードと戻される入出力域のマップ *(*続き*)*

キーワード	ワード **0**	ワード **1**	ワード **2**	ワード **3**	ワード **4**	ワード **5**	ワード **6**	ワード **7**	ワード **8**	ワード **9**

フィールド **1**
エリア名

フィールド **2**
修飾された SSA からの順次従属位置

フィールド **3**
順次従属部分の未使用 CI

フィールド **4**
独立オーバーフロー部分の未使用 CI

フィールド **5**
コミット済み順次従属セグメントのタイム・スタンプ

フィールド **6**
IMS ID

フィールド **7**
埋め込み

POS 呼び出しが成功した後の入出力域の内容は、以下のとおりです。

LL (表には示されていません) 入出力域のデータの全体の長さを 2 進数で示す、2 バイト・フィールド。

(フィールド **1)**

区域名
AREA ステートメントで指定された DD 名を示す 8 バイトのフィールド

(フィールド **2)**

位置 最後に挿入された順次従属セグメントに関する位置情報が入っている 8 バイトのフィールド。 このルートに順次従属セグメントがない場合、このフィールドにはゼロが入ります。

修飾された **SSA** からの順次従属位置
POS 呼び出しが成功すると、IMS は 2 つのデータをこの 8 バイト・フィールドに入れます。最初の 4 バイトにはサイクル・カウントが入り、 2 番目の 4 バイトには VSAM RBA が入ります。

POS 呼び出しの目標である順次従属セグメントが同じ同期インターバルで挿入された場合には、位置情報は戻されません。 11 バイトから 18 バイトまでは X'FF' が入ります。その他のフィールドには、通常のデータが入ります。

(フィールド **3)**

順次従属部分の未使用 **CI**
順次従属部にある未使用制御インターバルの数が入っている、4 バイト・フィールド。

(フィールド **4)**

独立オーバーフロー部分の未使用 **CI**
独立オーバーフロー部分にある未使用制御インターバルの数が入っている、4 バイト・フィールド。

(フィールド **5)**

コミット済み順次従属セグメントのタイム・スタンプ
修飾 POS 呼び出しで位置指定された SDEP セグメントに該当するタイム・スタンプが入っている、8 バイト・フィールド。

(フィールド **6)**

IMS ID
SDEP セグメントが配置されていた CI を所有する IMS を識別します。

(フィールド **7)**

埋め込み
入出力域をダブルワード境界に桁合わせするための 8 バイトの埋め込み域。このフィールドにはデータは戻されません。

フリー・スペースの識別

すべてのオンライン域から、区域名と、順次従属部の中の次の使用可能位置をリトリーブするために、非修飾 POS 呼び出しを出すことができます。このタイプの呼び出しでは、独立オーバーフロー部分および順次従属部の中の未使用スペースもリトリーブされます。

非修飾 POS 呼び出しが不成功に終わったとき、入出力域には長さ (LL) が入り、その後ろにデータベース内の既存区域と同じ数の項目が続きます。どの項目にも、次のフィールドが含まれています。

区域名
AREA からの DD 名が入った 8 バイト・フィールド。

位置　2 進ゼロからなる 8 バイト・フィールド。

未使用の **SDEP CI**
2 進ゼロからなる 4 バイト・フィールド。

未使用の **IOV CI**
2 進ゼロとそれに続く状況コードからなる 4 バイト・フィールド。

DEDB におけるコミット・ポイント処理

IMS は、プログラムがコミット・ポイントに達するまで、データベースの更新をプロセッサー・ストレージ内に保存します。IMS は、DEDB の更新を高速機能のバッファーに保管します。プログラムがコミット・ポイント処理を正常に完了させるまで、データベース更新は DEDB に適用されません。

プログラムがコミット・ポイント処理を正常に完了させるまで、データベース更新は DEDB に適用されません。ただし、MSDB に対する Get 呼び出しの場合と異なり、DEDB 内の更新済みセグメントに対して Get 呼び出しを出すと、コミット・ポイントが発生していない場合にも更新済みの値が戻されます。

BMP は、DEDB を処理するときには、終了する前に CHKP または SYNC 呼び出しを出してコミット・ポイント処理を行う必要があります。これを行わないと、BMP は異常終了コード U1008 を出して異常終了します。

関連概念:

379 ページの『MSDB および DEDB におけるコミット・ポイント処理』

P 処理オプション

ユーザー・プログラムの PCB 内で P 処理オプションを指定しておくと、セグメントのリトリーブまたは挿入を行う呼び出しによって作業単位 (UOW) 境界が越えられたときに、GC 状況コードがユーザーのプログラムに戻されるようになります。

関連資料: DEDB の UOW の詳細については、「*IMS V15* データベース管理」を参照してください。

UOW 境界を越えても、ユーザーのプログラムにとって特に重要ではないと思われますが、GC 状況コードが戻されることは、そのときが SYNC または CHKP のいずれかの呼び出しを発行するのに適した時点であることを示しています。ユーザーのプログラムが GC 状況コードを受け取った後で SYNC または CHKP 呼び出しを出すことには、次のような利点があります。

- データベース内のユーザーの位置が保持されます。通常は、SYNC または CHKP 呼び出しを出すとデータベース内の位置が失われ、アプリケーション・プログラムは処理を再開する前に位置を再設定する必要があります。
- コミット点が一定の間隔で発生します。

GC 状況コードが戻されると、データはリトリーブも挿入もされません。プログラムでは、次のいずれかを行うことができます。

- SYNC または CHKP 呼び出しを発行し、GC 状況コードが戻される原因となった呼び出しを発行し直してデータベース処理を再開する。
- GC 状況コードを無視し、この状況コードが戻される原因となった呼び出しを出し直してデータベース処理を再開する。

関連概念:

402 ページの『DEDB に対する従属セグメントを使用した呼び出し』

H 処理オプション

ユーザーの呼び出し側プログラムの PCB 内で H 処理オプションが指定されていると、セグメントのリトリーブまたは挿入を行う呼び出しによって作業単位 (UOW) 境界または区域境界を越えたときに、GC 状況コードがユーザーのプログラムに戻されます。このプログラムは、その PCB に対する別の呼び出しを出す前にコミット処理を行う必要があります。

コミット処理が行われないと、FR 状況コード (合計バッファー割り振りの超過) が戻され、この同期インターバルにおけるすべてのデータベース変更は廃棄されます (同期点エラー)。

区域境界を越えると GC 状況コードが戻されるため、アプリケーション・プログラムが SYNC または CHKP 呼び出しを発行し、直前の区域の処理で獲得されたリソース (バッファーなど) を強制的に終結処理できるようになります。直前の GC 状況

コードに対して SYNC または CHKP 呼び出しが適切に発行された場合でも、この終結処理が、GN または GHN 呼び出しに対して GC 状況コードを連続して戻す場合があります。

アプリケーションが HSSP を実行し、DEDB AREA を準じに処理しているとき、IOV チェーンが大きくて、バッファー不足状態が発生することがあります。そのときは、FW 状況コードがアプリケーションに返されます。普通、アプリケーションはコミット要求を出し、位置を次の UOW に設定しますが、これでは直前の UOW が処理を終えることができません。これに対し、FW 状況コードを受け取ってからコミット要求を出し、位置を同じ UOW にとどめておくようにすれば、直前の UOW の処理を終えることができます。また、アプリケーションの位置を、その FW 状況コードを発生させた位置に戻すことも必要です。次に、コマンド・シーケンスとそれに対応するアプリケーション応答を示します。

```
GN                        root1
GN                        root2
GN                        root3
GN                        root4            /*FW status code received*/
CHKP
GN    SSA=(root4)        root4            /*User reposition prior to CHKP*/
GN                        root5
```

DEDB に対する従属セグメントを使用した呼び出し

DEDB の直接従属セグメントおよび順次従属セグメントに対し、DL/I 呼び出しを発行できます。

ルート・セグメントに対して発行することのできる DL/I 呼び出しは、GU、GN (GNP は、ルート・セグメントに対しては意味を持ちません)、DLET、ISRT、および REPL です。直接従属セグメントに対してはすべての DL/I 呼び出しを発行することができ、また、順次従属セグメントに対しては Get 呼び出しと ISRT 呼び出しを発行できます。

直接従属セグメント

直接従属セグメントに対する DL/I 呼び出しには、既存の階層内のレベル (最高 15) と同数の SSA が含まれます。また、コマンド・コードと複数の修飾ステートメントを含めることもできます。 DEDB に対する DL/I 呼び出しでコマンド・コードを使用する場合には、全機能データベースの場合と同じ規則が適用されます。

DEDB に対する呼び出しで D コマンド・コードを使用する場合には、プログラムの PCB で P 処理オプションを指定する必要はありません。 DEDB の場合の P 処理オプションは全機能データベースのときとは異なる意味をもちます。

サブセット・ポインターを使用する DEDB だけで使用することのできる、特殊なコマンド・コードがいくつかあります。ユーザーのプログラムは、これらのコマンド・コードを使用して、サブセット・ポインターの読み取りと更新を行います。

順次従属セグメント

順次従属セグメントは発生順に保管されるため、ジャーナル、データ収集、および監査アプリケーション・プログラムに使用すると便利です。順次従属セグメントに

は、直接アクセスすることもできます。ただし通常は、順次従属セグメントは、データベース・スキャン・ユーティリティーを使用して順次にリトリーブされます。

制約事項**:** 順次従属セグメントを処理する場合の制約事項は次のとおりです。

- 順次従属セグメントには、F コマンド・コードしか使用できません。IMS は、他のすべてのコマンド・コードを無視します。
- 順次従属セグメントに対する呼び出しでは、ブール演算子を使用することはできません。

関連資料**:** ユーティリティーの詳細については、「*IMS V15* データベース・ユーティリティー」を参照してください。

関連概念:

380 ページの『サブセット・ポインター・コマンド・コードを使用した高速機能 DEDB の処理』

関連資料:

401 ページの『P 処理オプション』

DEDB 情報を取り出すための DEDB DL/I 呼び出し

DL/I 呼び出しを発行して、高速処理データベース (DEDB) に関する構造情報を取得できます。 DL/I 呼び出しを発行できるすべてのアプリケーションは、ここに記載されている DL/I 呼び出しを利用できます。

呼び出しには、次の 2 つの基本的なタイプがあります。

- 最初のタイプは、特定のタイプ '2' DL/I 呼び出しに必要となる最小入出力域長を返します。
- 2 番目のタイプは、指定された DEDB に関する具体的な情報を返します。

これらの DL/I 呼び出しはそれぞれ、Input Output Input Area (IOAI)と呼ばれる呼び出しインターフェース・ブロック、通信プログラム PCB (TP PCB)、および入出力域を必要とする特定の呼び出しを使用します。IOAI の初期化には、すべての呼び出しに共通する必須の初期化と、個々の呼び出しに固有の初期化があります。IOAI および入出力域は、キー 8 ストレージ内に取得する必要があります。

次の表は、DEDB 情報を抽出するための DL/I 呼び出しについての説明です。

表 *56. DEDB DL/I* 呼び出し

DL/I 呼び出し	説明
AL_LEN	AREALIST 呼び出しに必要となる入出力域の最小の長さを返します。
DI_LEN	DEDBINFO 呼び出しに必要となる入出力域の最小の長さを返します。
DS_LEN	DEDBSTR 呼び出しに必要となる入出力域の最小の長さを返します。
AREALIST	指定された DEDB に含まれる領域のリストを返します。各領域は、DBFCDAL1 によってマップされています。

表 56. DEDB DL/I 呼び出し (続き)

DL/I 呼び出し	説明
DEDBINFO	DBFCDDI1 によってマップされた、DMCB からの DEDB 情報を返します。
DEDBSTR	DEDB のセグメントのリストとセグメント・データを返します。各セグメントは、DBFCDDS1 によってマップされています。

標準インターフェースとレジスター 1 を使用する DL/I 呼び出しは、IOAI_CA を指す必要があります。

IOAI の構造を次に示します。

```
                              starting
                               offset    note
IOAI_START     DS   0F
IOAI_NAME      DC   CL4'IOAI'      0      *1
IOAI_#FPU      DC   CL4'#FPU'      4      *1
IOAI_#FPI      DC   CL8'#FPUCDPI'  8      *1
IOAI_SUBC      DC   CL8' '        10      *1
*
IOAI_BLEN      DC   A(0)          18      *1
IOAI_ILEN      DC   A(0)          1C      *1
IOAI_IOAREA    DC   A(0)          20      *1
*
IOAI_CALL      DC   A(0)          24      *1
IOAI_PCBI      DC   A(0)          28      *1
IOAI_IOAI      DC   A(0)          2C      *1
*
IOAI_DLEN      DC   A(0)          30      *2
IOAI_STATUS    DC   CL2' '        34      *2
IOAI_B_LEVEL   DC   XL2'0'        36      *2
IOAI_STATUS_RC DC   A(0)          38      *2
IOAI_USERVER   DC   A(0)          3C      *1
IOAI_IMSVER    DC   A(0)          40      *2
*
IOAI_IMSLEVEL  DC   A(0)          44      *2
*
IOAI_APPL_NAME DC  CL8' '         48      *1
IOAI_USERDATA  DC  CL8' '         50      *1
IOAI_TIMESTAMP DC  CL8' '         58      *2
*                       input words.
IOAI_IN0       DC   A(0)          60      *3
IOAI_IN1       DC   A(0)          64      *3
IOAI_IN2       DC   A(0)          68      *3
IOAI_IN3       DC   A(0)          6C      *3
IOAI_IN4       DC   A(0)          70      *3
*                       feedback words
IOAI_FDBK0     DC   A(0)          74      *2
IOAI_FDBK1     DC   A(0)          78      *2
IOAI_FDBK2     DC   A(0)          7C      *2
IOAI_FDBK3     DC   A(0)          80      *2
IOAI_FDBK4     DC   A(0)          84      *2
*                       workareas.
IOAI_WA0       DC   A(0)          88      *4
IOAI_WA1       DC   A(0)          8C      *4
IOAI_WA2       DC   A(0)          90      *4
IOAI_WA3       DC   A(0)          94      *4
IOAI_WA4       DC   A(0)          98      *4
*
```

```
              DS 20F'0'          9C      for future expansion
IOAI_END_CHAR DC   CL4'IEND'     EC      *1
IOAI_LEN         len(DBFIOAI) = x'F0' bytes
```

注**:**

1. ユーザーは、これらのフィールドの初期化に責任を負います。
2. IMS は、これらのフィールドを使用して、呼び出し元にデータを返します。返されたデータがどのフィールドにあるかは DL/I 呼び出しに依存し、その情報は特定の呼び出しタイプのセクションで文書化されます。
3. 各 DL/I 呼び出しタイプの説明に示すとおり、DL/I 呼び出しに関する追加データを渡すために使用できます。
4. これらのフィールドは変更されず、アプリケーションで作業域として使用できます。

次の表の各フィールドは、以下のすべての DL/I 呼び出しで初期化する必要があります。

表 *57. DEDB DL/I* 呼び出しのフィールドの初期化

フィールド	説明
IOAI_NAME	このブロックを識別する文字「IOAI」。
IOAI_#FPU	これが #FPU 呼び出しであることを示す文字「#FPU」。
IOAI_#FPI	これがサブセット呼び出しであることを示す文字「#FPUCDPI」。
IOAI_SUBC	DL/I 呼び出し: AL_LEN、AREALIST、DS_LEN、 DEDBSTR、DI_LEN、または DEDBINFO。
IOAI_BLEN	IOAI の合計長 (x'F0')。
IOAI_CALL	IOAI_#FPU のアドレス。
IOAI_PCBI	TPCB のアドレス。
IOAI_IOAI	このブロックのアドレス。ユーザーは、DL/I リストの終わりを示すために、上位ビットをオンに設定する必要があります。
IOAI_USERVER	呼び出しバージョン番号。デフォルトは 1 です。これは、特定の呼び出しのバージョン番号です。このフィールドは、アプリケーションが変更に対して敏感になるように、特定の呼び出しが変更されると将来更新されます。
IOAI_END_CHAR	ブロックの終わりを識別する文字「IEND」。

次の各フィールドは、特定の DL/I 呼び出しで初期化されます。特定の呼び出しが入出力域を必要としない場合、これらのフィールドは無視されます。

表 *58.* 特定の *DEDB DL/I* 呼び出しで初期化されるフィールド

フィールド	説明
IOAI_ILEN	接頭部と接尾部を含む入出力域の合計長。

表 58. 特定の DEDB DL/I 呼び出しで初期化されるフィールド (続き)

フィールド	説明
IOAI_IOAREA	入出力域のアドレス。
入出力域	最初のワード: 入出力域の長さ (IOAI_ILEN と同じ)。最後のワード: X'FFFFFFFF'。これは「入出力域の終わり」マーカーです。
IOAI_IN0 から IOAI_IN4	必須の場合もある 5 つの入力ワード。

次の各フィールドは、すべての DEDB DL/I 呼び出しタイプで、IMS によって更新されます。

表 59. すべての DL/I 呼び出しタイプで IMS によって更新されるフィールド

フィールド	説明
IOAI_DLEN	IMS が返す出力データの長さ。このフィールドは情報提供のみのフィールドです。
IOAI_STATUS	2 バイトの状況コード。
IOAI_STATUS_RC	戻りコード (必要な場合)。
IOAI_IMSVER	この呼び出しの最大バージョン。
IOAI_IMSLEVEL	IMS レベル。

次の各フィールドは、特定の DL/I 呼び出しによって更新される場合があります。

表 60. 特定の DL/I 呼び出しによって更新されるフィールド

フィールド	説明
入出力域	最初のワード: 無変更。データ: 特定の呼び出しタイプを参照。最後のワード: 変更される可能性があります。
IOAI_FDBK0 から IOAI_FDBK4	5 つの出力ワード。特定の呼び出しによって説明されているとおり、データを返す場合があります。

```
DBFCDAL1 mapping:        offset
CDAL_START   DS   0F
CDAL_ARNM    DS    CL8     00      Area name
CDAL_FLGS    DS    0XL4    08      Flag Bytes
CDAL_FLG1    DS    XL1     08      Flags for area status:
CDAL_F1OP     EQU   X'01'         - Area is opened
CDAL_F1BK     EQU   X'02'         - Temporary bit for backout
CDAL_F1UT     EQU   X'04'         - Utility active on this area
CDAL_F1ER     EQU   X'08'         - Error recovery needed
CDAL_F1AF     EQU   X'80'         - Sequential dep. part full
CDAL_F1EP     EQU   X'40'         - I/O error
CDAL_F1ST     EQU   X'20'         - Area stop request
CDAL_F1RE     EQU   X'10'         - Area restart request
CDAL_FLG2    DS    XL1     09       Reserved for Flag Byte #2
CDAL_FLG3    DS    XL1     0A       Reserved for Flag Byte #3
CDAL_FLG4    DS    XL1     0B       Reserved for Flag Byte #4
             DS    1F      0C       for growth
CDAL_LEND    DS    0F             End of area list entry
CDAL_LEN     EQU   *-&AA._START;   Len of area list entry
```

```
DBFCDDI0 mapping:        offset
CDDI_START   DS   0D
CDDI_DBNM    DS    CL8     00    Database name
CDDI_ANR     DS    H       08    Number of areas defined
CDDI_HSLV    DS    H       0A    Max SEGM level in the DB
CDDI_SGNR    DS    H       0C    Highest valid SEGM code
CDDI_SEGL    DS    H       0E    Maximum IOA length
CDDI_HBLK    DS    F       10    Number of anchor blocks
CDDI_RMNM    DS    CL8     14    Randomizing module name
CDDI_RMEP    DS    F       1C    Randomizing module entry point
             DS    8F      20    Reserved
             DS   0D             Align on double word boundary
CDDI_LEN     EQU   *-&AA._START;   Length of this area (x'40')

DBFCDDS1 mapping:         offset
CDDS_START   DS   0F
CDDS_GNAM    DS    CL8      00  SEGMENT NAME
CDDS_GDOF    DS    H        08  OFFSET FROM START SEQ TO DATA
CDDS_MAX     DS    H        0A  MAX SEG LEN
CDDS_MIN     DS    H        0C  MIN SEG LEN
CDDS_DBOF    DS    H        0E  OFFSET TO SEG ENTRIES
CDDS_NRFLD   DS    FL1      10  NUMBER OF FIELDS IN SEG
CDDS_SC      DS    FL1      11  SEGMENT CODE
CDDS_PREF    DS    H        12  POINTER OFFSET IN PARENT PREF
CDDS_FLG1    DS    X        14  FLAG BYTE
CDDS_FL1K     EQU   X'80'        KEY SEGMENT
CDDS_FL1S     EQU   X'40'        SEQUENTIAL DEP SEGMENT
CDDS_FL1P     EQU   X'20'        PCL POINTER TO PARENT
CDDS_FISRT   DS    X        15  INSERT RULES
CDDS_PARA    DS    H        16  OFFSET TO PARENT SEGMENT
CDDS_SBLP    DS    F        18  SIBLING POINTER
CDDS_LEVL    DS    XL1      1C  SEGMENT LEVEL
CDDS_KEYL    DS    XL1      1D  KEY LENGTH - 1
CDDS_KDOF    DS    H        1E  OFFSET TO KEY FIELD IN SEGMENT
CDDS_RSRVE   DS    XL4      20  FOR USE IN UMDR0 | RESERVED
CDDS_CMPC    DS    A        24  A(CMPC)
CDDS_FLG2    DS    XL1      28  FLAG BYTE 2 (fixed length)
             DS    XL3      29  FOR GROWTH
             DS   5F        2C  for growth
CDDS_END     DS   0F            END
CDDS_LEN     EQU    *-&AA._START;  len of SDB entry
```

以下の状況コードは、これらの新規 DL/I 呼び出しに固有のものです。

表 *61.* 特定の *DEDB DL/I* 呼び出しに関する状況コード

状況コード	説明
AA	無効な #FPU/#FPUCDPI 呼び出し。
AB	getmain エラー。
AC	DEDB 名が見つかりません。
AD	入出力域がデータを格納するのに十分な長さではありませんでした。
AE	IOAI_LEN がゼロで埋められていました。これは、呼び出し元が埋める必要があります。
AF	入出力域アドレスが IOAI_IOAREA によって渡されませんでした。
AG	IOAI は、それ自体である IOAI_IOAI を指していません。
AH	IOAI に「IOAI」が入っていませんでした。

表 61. 特定の DEDB DL/I 呼び出しに関する状況コード (続き)

状況コード	説明
AI	入出力域内の入出力域長が IOAI に一致しません。
AJ	入出力域にリストの終わりマーカーが入っていませんでした。
AK	IOAI にブロックの終わりマーカー「IEND」がありませんでした。
AL	IOAI_BLEN が正しくありません。
AM	#FPUCDPI 呼び出しで IOAI を介して DEDB が渡されませんでした。

AL_LEN 呼び出し

AL_LEN 呼び出しは、AREALIST 呼び出しに必要な入出力域の最小の長さを戻します。

入力

IOAI
上記の説明のように、形式が設定されて入力されます。

IOAI_IN0
DEB 名が入っているストレージを指します。

出力

IOAI_STATUS
呼び出し状況で、ブランクは成功を意味します。

IOAI_FDBK0
入出力域の最小の長さ。

IOAI_FDBK1
この DEDB 内の領域の数。

DI_LEN 呼び出し

DEDBINFO 呼び出しに必要となる入出力域の最小の長さを返します。

入力

IOAI
上記の説明のように、形式が設定されて入力されます。

IOAI_IN0
DEB 名が入っているストレージを指します。

出力

IOAI_STATUS
呼び出し状況で、ブランクは成功を意味します。

IOAI_FDBK0
入出力域の最小の長さ。

DS_LEN 呼び出し

DEDBSTR 呼び出しに必要となる入出力域の最小の長さを返します。

入力

IOAI
上記の説明のように、形式が設定されて入力されます。

IOAI_IN0
DEB 名が入っているストレージを指します。

出力

IOAI_STATUS
呼び出し状況で、ブランクは成功を意味します。

IOAI_FDBK0
入出力域の最小の長さ。

IOAI_FDBK1
この DEDB 内のセグメントの数。

AREALIST 呼び出し

AREALIST 呼び出しは、指定された DEDB の一部のエリア (各エリアは DBFCDAL1 によってマップされる) のリストを戻します。

入力

IOAI
上記の説明のように、形式が設定されて入力されます。

IOAI_IN0
DEB 名が入っているストレージを指します。

入出力域
上記の説明のように形式が設定されます。

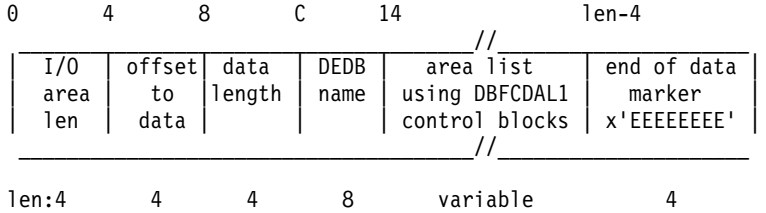
出力

IOAI_STATUS
呼び出し状況で、ブランクは成功を意味します。

IOAI_FDBK0
入出力域の最小の長さ。

IOAI_FDBK1
この DEDB 内の領域の数。

入出力域



DEDBINFO 呼び出し

DBFCDDI1 によってマップされた、DMCB からの DEDB 情報を返します。

入力

IOAI
上記の説明のように、形式が設定されて入力されます。

IOAI_IN0
DEB 名が入っているストレージを指します。

入出力域
最初のワード内の長さと、入出力域の終わりマーカーとしての「FFFFFFFF」によって形式が設定されます。

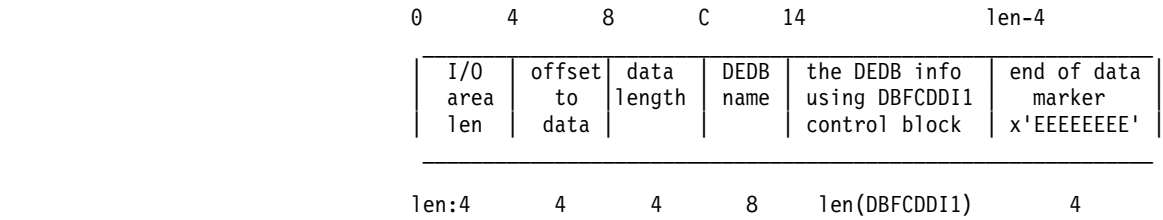
出力

IOAI_FDBK0
入出力域の最小の長さ。

IOAI_FDBK1
DEDBSTR 呼び出し用の最小入出力域。

IOAI_FDBK2
AREALIST 呼び出し用の最小入出力域。

入出力域



DEDSTR 呼び出し

DEDB のセグメントのリストおよびセグメント・データを返します。各セグメントは、DBFCDDS1 によってマップされています。

入力

IOAI
上記の説明のように、形式が設定されて入力されます。

IOAI_IN0
DEB 名が入っているストレージを指します。

入出力域
最初のワード内の長さと、入出力域の終わりマーカーとしての「FFFFFFFF」によって形式が設定されます。

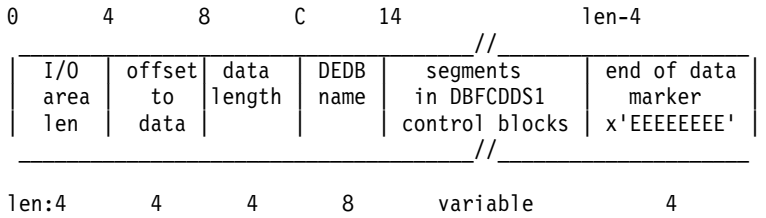
出力

IOAI_STATUS
入出力域の最小の長さ。

IOAI_FDBK0
DEDBSTR 呼び出し用の最小入出力域。

IOAI_FDBK1
セグメント呼び出し用の最小入出力域。

入出力域



高速機能のコーディングに関する考慮事項

DL/I 呼び出しを使用すれば高速機能データベースにアクセスすることができます。その他に、FLD と POS の 2 つの呼び出しも使用できます。これらのいずれの呼び出しを使用することができるかは、処理したい高速機能データベースのタイプによって異なります。

MSDB を処理するために、以下の呼び出しを使用できます。

- 非端末関連 MSDB の場合

  FLD

  GU および GHU

  GN および GHN

  REPL
- 端末関連、固定 MSDB の場合

  FLD

  GU および GHU

  GN および GHN

  REPL
- 端末関連、動的 MSDB の場合

  DLET

  FLD

  GU および GHU

  GN および GHN

  ISRT

  REPL

DEDB の処理には、以下の呼び出しを使用できます。

- DEQ
- DLET
- FLD
- GU および GHU

- GN および GHN
- GNP および GHNP
- ISRT
- POS
- REPL
- RLSE

第 22 章 ODBA アプリケーション・プログラムの作成

ODBA インターフェースを使用すると、Db2 for z/OS ストアード・プロシージャーなど、IMS の制御範囲外の環境から IMS DB データベースにアクセスできます。

MPR、BMP、IFP など IMS 制御領域内でローカルに制御されたデータベースへアクセスする場合、ODBA インターフェースは必要ありません。

z/OS アプリケーション・プログラム (以降は ODBA アプリケーション・プログラムと呼びます) は、IMS では制御領域と異なる領域と見なされる、別の z/OS アドレス・スペースで稼働します。この別個の z/OS アドレス・スペースは、これ以降は、z/OS アプリケーション領域と呼びます。

ODBA インターフェースは、データベース・リソース・アダプター (DRA) を介して IMS DB にアクセスします。ODBA アプリケーション・プログラム (このプログラムが実行されている z/OS 内のすべてのアドレス・スペースにアクセス可能) は、ODBA インターフェースを介して IMS DB データベースにアクセスします。次の図はこの概念を表したもので、この環境におけるコンポーネント間の関係を示しています。

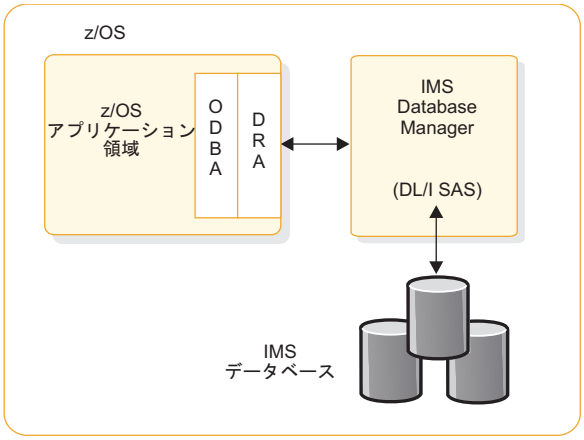


図 *69. z/OS* アプリケーション領域による *IMS DB* への接続

1 つの z/OS アプリケーション領域は複数の IMS DB と接続可能で、複数の z/OS アプリケーション領域は単一の IMS DB と接続可能です。この接続は、CICS から DBCTL への接続と似ています。

ODBA アプリケーション・プログラムの一般的なアプリケーション・プログラム・フロー

ODBA アプリケーション・プログラムは、アプリケーション・インターフェース・ブロック (AIB) の AERTDLI インターフェースを使用して、呼び出し (DL/I 呼び出しなど) を発行します。

AIB 呼び出しを正常に行うには、以下の条件を満たさなければなりません。

1. 呼び出しで AIB が渡されない場合、異常終了コード U261 が出されます。
2. 渡された AIB が有効でない場合、異常終了コード U476 が出されます。
3. 渡された AIB のサイズが十分 (264 バイト) でない場合、AIB の戻りコードおよび理由コードが、X'104' および X'228' に設定されます。
4. 渡された AIB がフルワード境界にない場合、z/OS システムは異常終了コード S201 を戻します。
5. 呼び出しにその他内部的な問題がある場合、その他の戻りコードおよび理由コードが z/OS アプリケーション・プログラムに返されます。これらの戻りコードおよび理由コードの完全なリストについては、「*IMS V15* メッセージおよびコード 第 *4* 巻*: IMS* コンポーネント・コード」を参照してください。

ODBA アプリケーション・プログラムは、言語モジュール (DFSCDLI0) とリンク・エディットさせる必要があります。そうしないと、このモジュールは、z/OS アプリケーション領域にロードされる可能性があります。 DFSCDLI0 のエントリー・ポイントは AERTDLI です。

制約事項**:** ODBA インターフェースは、バッチ DL/I 領域への呼び出しをサポートしません。

ODBA アプリケーション・プログラムの基本的なプログラムの流れは、次のようになります。

1. アプリケーション実行環境を確立する。 アプリケーション実行環境は、z/OS アプリケーション領域で初期設定する必要があります。

   この環境を初期化するには、CIMS INIT 呼び出しを使用するか、複数の IMS システムと接続を確立する必要がある場合は、CIMS CONNECT 呼び出しを使用します。

   CIMS の INIT 副次機能を使用すると、開始テーブル ID を AIB のオプション・フィールドである AIBRSNM2 に組み込み、開始テーブルにリストされている IMS DB システムに接続できます。 AIBRSNM2 フィールドがブランクの場合は、PSB を割り振る際に IMS DB に接続します。

   接続呼び出しの形式は、次のようになります。

   ```
   CALL AERTDLI parmcount, CIMS, AIB
   ```

   ここで:

   CIMS 必須の呼び出し関数です。

   AIB 次のフィールドから構成されます。

AIBSFUNC
ここに指定する副次機能は、INIT または CONNECT のいずれかです。このフィールドは必須です。

AIBRSNM1
AIB に関連付けられたアプリケーション・サーバーの目印 ID を提供する、オプションのフィールドです。このフィールドは 8 バイトです。

AIBRSNM2
INIT 副次機能の場合のオプション・フィールドであり、ここにオプションの 4 バイトの開始テーブル ID を指定できます。呼び出しを前提条件としてのみ発行する場合、この ID はオプションです。ID を指定した場合、z/OS アプリケーション領域は、選択した開始テーブルの DBCTLID パラメーターで指定した IMS DB に接続します。AIBRSNM2 フィールドは、CONNECT 副次機能ではサポートされません。

接続の特性は、DRA 開始テーブルによって判別されます。開始テーブル名は DFSxxxx0 で、xxxx は CIMS 呼び出しと APSB 呼び出しで使用される開始テーブル ID を示します。各開始テーブルによって接続属性の組み合わせが定義され、属性のいずれかが IMS DB のサブシステム ID です。

CONNECT 副次機能を使用する場合、呼び出し元のアプリケーション・プログラムは、オプションでその固有の接続プロパティー・テーブルを提供できます。これは、そのテーブルのアドレスを、CONNECT 副次機能が使用する ODBA データ・ストア接続テーブル内の項目で指定することで可能になります。接続プロパティー・テーブルは、DFSPRP マクロによりマップされます。

関連資料: DRA 開始テーブルの作成について詳しくは、「*IMS V15* システム定義」を参照してください。

2. PSB を割り振る。 APSB 呼び出しは、CPIC ドリブン・プログラム用に導入されたもので、これを ODBA インターフェースで使用して、PSB を z/OS アプリケーション領域に割り振ります。

   セキュリティーが検査されると、呼び出しは正常に行われます。詳しくは、「*IMS V15* コミュニケーションおよびコネクション」の『Accessing IMS databases through the ODBA interface』を参照してください。

   APSB 呼び出しは、次のような形式になります。

   ```
   CALL AERTDLI parmcount, APSB, AIB
   ```

   ここで:

   APSB 必須の呼び出し関数です。

   AIB アプリケーション・インターフェース・ブロックの名前です。AIB の以下のフィールドに入力する必要があります。

   AIBRSNM1
   8 文字の PSB 名です。

AIBRSNM2
開始テーブル ID として使用される、IMS の 4 バイトの別名です。

割り振り要求を正常に行うには、以下の条件を満たさなければなりません。

- PSB が存在し、RACF を介したセキュリティー検査が正常に終了する必要があります。
- CIMS INIT 呼び出しまたは CIMS CONNECT 呼び出しのいずれかによって、ODBA 環境が確立されている必要があります。
- APSB 呼び出しを行うときには、z/OS リソース・リカバリー・サービス (RRS) がアクティブでなければなりません。

複数の PSB を同時にアクティブにすることができます。これはサーバー環境で一般的です。特定の IMS DB への特定の呼び出しを行う際に、どの PSB を使用するかを識別するトークンは特に示されません。そのため、同じ PSB インスタンス (APSB、DB 呼び出し、および DPSB) への呼び出しには、すべて同じ AIB を使用する必要があります。こうすることにより、同じ IMS DB に対して、同一の PSB の複数のインスタンスを同時に使用することができます。この並列処理は、開始テーブルで指定したスレッド数によって制御されます。IMS DB インスタンスでサポートされるスレッドおよび従属領域の最大数は、999 です。

3. DB 呼び出しを実行する。 ごく一部の例外を除いて、DL/I 呼び出しはすべて AIB を介してサポートされます。非サポートの呼び出しを実行すると、メッセージ処理 (IOPCB はシステム呼び出しでのみ使用可能)、 CKPT、 ROLL、 ROLB、および INQY PROGRAM が発生します。代替宛先の PCB は使用できません。全機能データベースと DEDB の両方が使用可能です。
4. 変更をコミットする。 同期は、分散コミット呼び出しの SRRCMIT または ATRCMIT を発行して、あるいはそのロールバック形式の SRRBACK や ATRBACK を発行して行われます。IMS 同期点呼び出しは使用できません。コミットは、z/OS タスク内のすべての RRS 制御のリソースに対して有効です。
5. PSB を割り振り解除する。

   DPSB 呼び出しは、作業単位が完了したら使用します。デフォルトでは、DPSB 呼び出しを発行する前に必ずコミット呼び出しを発行する必要があります。コミットから DPSB 呼び出しまでの間、システム・サービス呼び出しも含めて DL/I 呼び出しを一切発行できません。

   DPSB 呼び出しは、次のような形式になります。

```
CALL AERTDLI parmcount, DPSB, AIB
```

   ここで:

   DPSB 必須の呼び出し関数です。

   AIB アプリケーション・インターフェース・ブロックの名前です。AIB の以下のフィールドに入力する必要があります。

   AIBRSNM1
   8 文字の PSB 名です。

AIBSFUNC
オプションのフィールドです。コミット処理を初期設定する前に PSB の割り振りを解除する場合、およびアプリケーションの外部からコミット処理が行われる場合は、「PREPbbbb」に設定します。

IMS は、フェーズ 1 のコミット処理を実行してリクエスターに制御を戻しますが、RRS (コミット・マネージャー) が完全コミット処理を要求するまで、未確定作業を保持しています。その例が DB2 UDB for z/OS ストアード・プロシージャーにあります。この場合、Db2 for z/OS がプロシージャーの代わりにコミット処理を初期設定します。このシナリオについては、Db2 for z/OSを参照してください。

6. 接続を終了する。

終了呼び出しは、次のような形式になります。

```
CALL AERTDLI parmcount, CIMS, AIB
```

ここで:

CIMS 必須の呼び出し関数です。

AIB アプリケーション・インターフェース・ブロックの名前です。AIB の以下のフィールドに入力する必要があります。

AIBSFUNC
必須フィールドで、値は TERM または TALL です。単一の IMS DB 接続を切断するには、TERM を使用します。この z/OS アプリケーション領域のためのすべての接続を切断してアドレス・スペースから DRA を除去するには、TALL を使用します。

AIBRSNM1
AIB に関連付けられたアプリケーション・サーバーの目印 ID を提供する、オプションのフィールドです。このフィールドは 8 バイト長です。

AIBRSNM2
副次機能が TERM の場合、前の APSB 呼び出しで使用された 4 バイトの開始テーブル ID を提供します。副次機能が TALL の場合、このフィールドは不要です。

サーバー・プログラム構造

z/OS アプリケーション環境内でのコミットの有効範囲は、z/OS リソース・リカバリー・サービス (RRS) へのコミット要求が出される TCB の下で行われるすべての処理を含みます。そのためサーバー環境には、個々のクライアント要求を管理するための別の TCB が必要です。各 TCB は、スレッド処理のために PST へマッピングされます。

次の図に、サーバー環境の TCB 構造の例を示します。

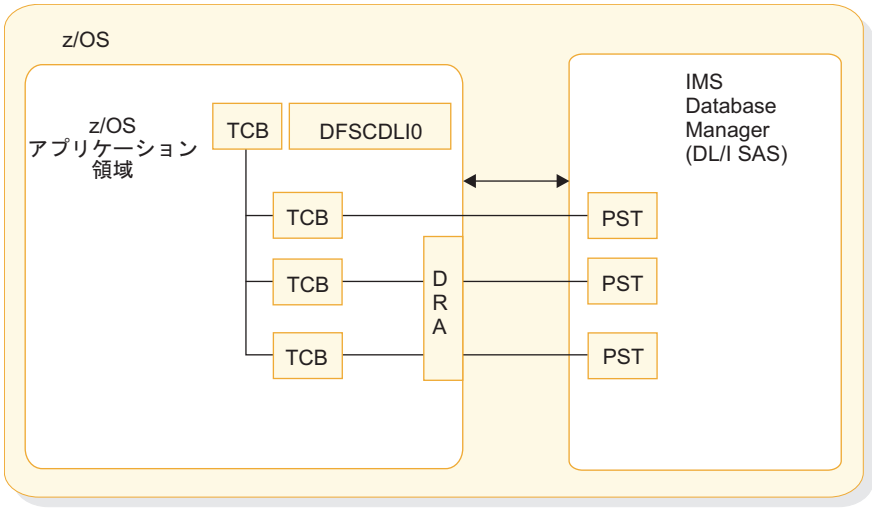


図 *70. 1* つのスレッドに対して *1* つの *TCB* を使用する *DRA*

IMS DB への各接続は、TCB 下のスレッドを使用します。APSB 呼び出しを処理するとコンテキストが確立され、TCB に結合されます。この TCB のすべてのコンテキストは、コミットの際に RRS でコミットされるか異常終了します。

z/OS アプリケーション領域が、IMS DB に接続するスレッドのインスタンスを数多く持つ多数のクライアントをサポートするサーバーである場合、リンク・エディットよりも DFSCDLI0 をロードすることを推奨します。

Db2 for z/OS ストアード・プロシージャーでの ODBA の使用

ODBA に接続する Db2 for z/OS ストアード・プロシージャーは、z/OS のワークロード・マネージャー管理対象 (WLM 管理対象) ストアード・プロシージャーのアドレス・スペースで実行する必要があります。

Db2 for z/OS は、ストアード・プロシージャーのアドレス・スペースに対して CIMS 呼び出しの INIT 副次機能または CONNECT 副次機能のどちらかを指定して、ODBA 環境を確立します。 CIMS INIT 呼び出しを発行する場合は、APSB 呼び出しを発行すると、特定の IMS DB に接続されます。 CIMS CONNECT 呼び出しを使用する場合、CIMS CONNECT 呼び出しまたは APSB 呼び出しを発行すると、オプションで 1 つ以上の IMS DB システムに接続できます。

ストアード・プロシージャーのアドレス・スペース内で実行中のストアード・プロシージャーは、それぞれストアード・プロシージャーが初期設定された際に Db2 for z/OS で確立した独自の TCB の下で実行します。制御が Db2 for z/OS に戻ると、Db2 for z/OS はストアード・プロシージャーに代わってコミット呼び出しを発行します。ストアード・プロシージャーによって発行されるのは、DPSB 呼び出しの PREP 副次機能のみです。

制約事項: ストアード・プロシージャーを単一の WLM ストアード・プロシージャー・アドレス・スペースにネストして IMS ODBA を呼び出すと、ODBA スレッドがハングします。

次の図では、Db2 for z/OS ストアード・プロシージャーのアドレス・スペースから IMS DB サブシステムへの接続を示しています。この接続により、DL/I データ

が SQL インターフェースを介して表示されます。ローカルでこの Db2 for z/OS に表示されるか、または Db2 for z/OS データベースに接続した DRDA で表示されます。

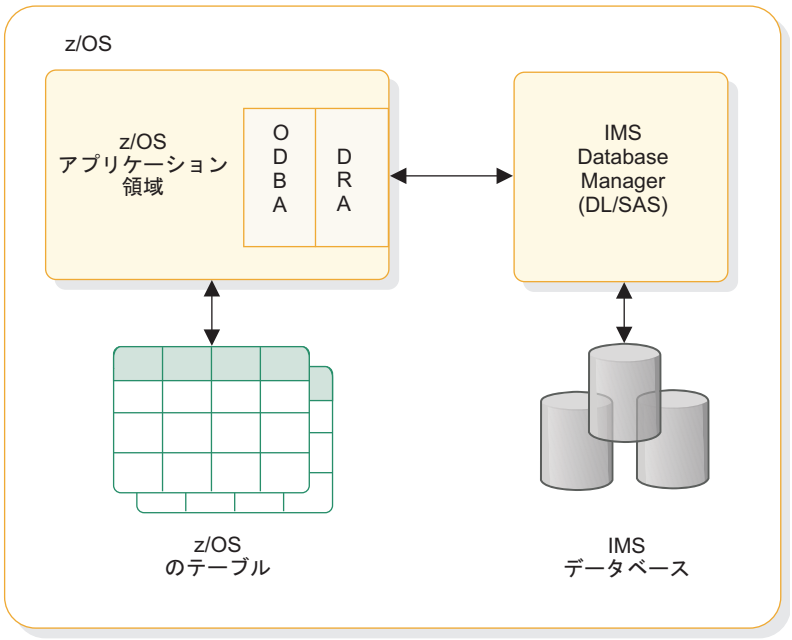


図 71. *Db2 for z/OS* ストアード・プロシージャーの *IMS DB* への接続

次の図では、Db2 for z/OS ストアード・プロシージャーと IMS DB を同時に使用した場合の一般的な関係を示しています。

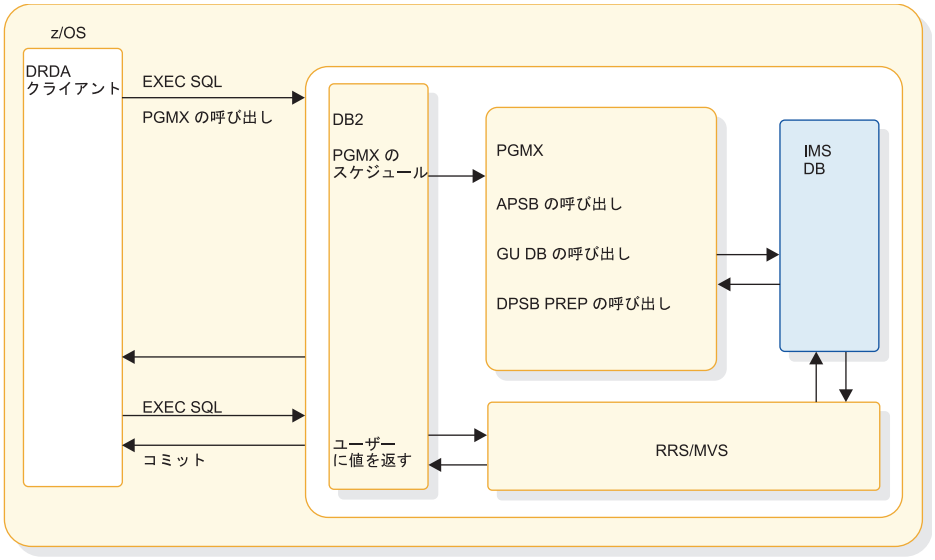


図 72. *Db2 for z/OS* ストアード・プロシージャーの関係

ODBA を使用する Db2 for z/OS ストアード・プロシージャーのベスト・プラクティス

ODBA サブシステムを使用する Db2 for z/OS ストアード・プロシージャーの z/OS アプリケーション・ランタイム環境では、特定のキー変更を行うことで、パフォーマンスの向上、ハング・スレッドの削減、アドレス指定スペースのリサイクルによって発生するオーバーヘッドの削減、および安定性の向上が可能になります。

ストアード・プロシージャーの設計に関する既存ガイダンスに従うほかに、以下の推奨事項について考慮してください。

- IMS Common Service Layer (CSL) の Open Database Manager (ODBM) コンポーネントを使用して、IMS で Db2 for z/OS ストアード・プロシージャーが予期せずに終了しないようにしてください。
- ODBA の問題の原因となり得るパラメーター (例えば、Db2 for z/OS ASUTIME 値) を使用するストアード・プロシージャーを設計しないでください。
- 高速実行ストアード・プロシージャー (不要なデータベース・ロックを保持したり、1 つの DB2 スレッド内で一度に複数の APSB 呼び出しを発行したりしないもの) では、パフォーマンスが向上し、ハング・スレッドの頻度が削減されます。
- IMS DFSPRP マクロ内の IMS システム・パラメーター (例えば、MINTHRD= や MAXTHRD=) の値を調整してください。
- 保守容易性を向上し、主要な診断データ収集機能を活動化するようにシステムを構成することで、すべての関連診断情報を迅速に収集し、ODBA の問題のトラブルシューティングに要する時間を短縮してください。

ODBA Db2 for z/OS ストアード・プロシージャーの設計に関するベスト・プラクティス

ODBA を使用する Db2 for z/OS ストアード・プロシージャーを設計する際に特定の推奨事項に従うと、それらのストアード・プロシージャーでは問題が発生する可能性が低くなります。

Open Database Manager (ODBM) を使用して、IMS で Db2 for z/OS ストアード・プロシージャーが予期せずに終了しないようにする

ODBM では IMS との通信に ODBA インターフェースを使用します。CSLDCxxx ODBM 構成 PROCLIB メンバー内のパラメーターの多くは、DFSPRP マクロ・パラメーターから作成される DFSxxxx0 始動テーブルで ODBA が使用するパラメーターと同じです。

ODBM は ODBA を使用するため、Db2 for z/OS や WebSphere Application Server for z/OS などの ODBA アプリケーション・サーバーは、ODBA ではなく ODBM を介して IMS に接続するように構成できます。ODBM を介して接続すると、DL/I 処理中に DB2 ストアード・プロシージャーまたは WebSphere

Application Server アプリケーション・プログラムが予期せずに終了した場合に、U0113 異常終了が発生するのを防ぐことができます。

ODBM を使用するように ODBA アプリケーション・サーバーを構成する場合、その ODBA アプリケーション・サーバーのもとで実行される既存のアプリケーション・プログラムを変更する必要はありません。

ODBM を使用するには、CSL の Structured Call Interface (SCI) コンポーネントおよび Operations Manager (OM) コンポーネントも有効にする必要があります。詳しくは、ODBM を使用する ODBA アプリケーション・サーバーの構成 (システム管理)を参照してください。

短時間実行ストアード・プロシージャーを設計する

ODBA のストアード・プロシージャーを作成する際は、それぞれの APSB 呼び出しから DPSB 呼び出しまでの間の合計実行時間が比較的短い (およそ 1 秒以下) になるようにストアード・プロシージャーを設計してください。この推奨事項は MPP ストアード・プロシージャーの推奨事項と同様です。

ODBA ストアード・プロシージャーは、その実行中はデータベース・ロックを保持しています。このストアード・プロシージャーが終了し、ロックを解放するまでは、データベースの IMS /DBR コマンドは完了できず、結果的に他の IMS 機能も失われる可能性があります。 APSB 呼び出しから DPSB 呼び出しまでの実行時間を最小限にすることで、/DBR コマンドの遅延と IMS 機能の喪失が起こる可能性も最小限に抑えられます。

この問題を防ぐもう 1 つの方法は、/DBR コマンドの発行前にすべての ODBA ストアード・プロシージャーを停止することです。

ODBA と対話する Db2 for z/OS ストアード・プロシージャーでは ASUTIME ステートメントを使用しない

Db2 for z/OS ASUTIME パラメーターは、Db2 for z/OS が Db2 for z/OS ストアード・プロシージャーを取り消すようになるまでの間の、このプロシージャーが実行していられるプロセッサー時間の制限を定義します。

推奨事項: ODBA と対話する Db2 for z/OS ストアード・プロシージャーでは ASUTIME ステートメントを使用しないでください。

ASUTIME の制限に達すると、Db2 for z/OS は ODBA に CIMS FTHD呼び出しを発行し、Db2 for z/OS が TCB を終了するまでの猶予時間として 6 秒を ODBA に与えます。環境によっては、TCB が終了する前に ODBA が準備を完了できない場合、スレッドのハングやその他の問題が発生する可能性があります。

Db2 for z/OS スレッドごとに一度に 1 つの APSB 呼び出しのみを使用する

Db2 for z/OS スレッドごとに一度に 1 つの APSB のみを使用してください。一度に複数の APSB をアクティブにすると、アプリケーションが異常終了した場合にさまざまなタイミングの問題が発生するおそれがあります。

タイミングの問題を回避するには、APSB 呼び出しを行ったら必ず、次の APSB 呼び出しの前に DPSB 呼び出しを行います。以下に例を示します。

```
APSB
(Other application calls)
DPSB
APSB
```

DRA 始動テーブルで IMS MINTHRD= パラメーターと MAXTHRD= パラメーターに同じ値を使用する

アプリケーション・プログラムの実行中に頻繁にスレッドを作成したり破棄したりすると、タイミングに関連するエラーが発生するおそれがあります。

MINTHRD= パラメーターと MAXTHRD= パラメーターを同じ値にすると、ODBA は初期設定中に要求された数のスレッドを作成し、終了までそれらを破棄しません。途中でスレッドが作成されたり破棄されたりすることはありません。

推奨事項: DRA 始動パラメーターでは MINTHRD= を MAXTHRD= と同じ値に設定してください。これらのパラメーターは Db2 for z/OS NUMTCB パラメーターとは異なります。NUMTCB は単一の Db2 for z/OS アドレス・スペースのもとで同時に実行できるスレッドの数を制御しますが、複数の Db2 for z/OS アドレス・スペースが存在できます。したがって、システム内の Db2 for z/OS タスクの総数が NUMTCB 設定によって制限されると想定しないでください。

関連タスク:

- ODBM を使用する ODBA アプリケーション・サーバーの構成 (システム管理)

関連資料:

- IMS PROCLIB データ・セットの CSLDCxxx メンバー (システム定義)
- DRA 始動テーブル (システム・プログラミング API)

ODBA を使用する Db2 for z/OS ストアード・プロシージャーの作成

ODBA 環境で実行するアプリケーション・プログラムを作成する場合、ODBA 呼び出しの一般的パターンに従うことでエラーを回避できます。

推奨事項: Db2 for z/OS スレッドごとに一度に 1 つの APSB 呼び出しのみを使用してください。一度に複数の APSB がアクティブになっていて、アプリケーションが異常終了すると、さまざまなタイミングの問題が発生するおそれがあります。

ODBA アプリケーションに複数の APBS 呼び出しが含まれる場合は、以下のいずれかの方法を使用して、検出不能なデッドロックを PST レベルで回避してください。

- 一連の更新を行った後、後続の各 APSB 呼び出しを発行する前に DPSB 呼び出しと SRRCMIT 呼び出しを発行します。
- ODBA 領域で単一スレッドを使用します。
- すべての ODBA アプリケーションに対してデータベース・アクセスとデータベース更新に関する標準手順を適用します。

従う呼び出しパターンは、RRS 同期呼び出しの開始時点によって異なります。

- RRS 同期呼び出しが ODBA アプリケーション内にから開始された場合は、次の呼び出しパターンを使用します。
  1. まず PSB 割り振り (APSB) 呼び出しを行って、ODBA アプリケーションに PSB を割り振ります。
  2. APSB 呼び出しの後に、アプリケーション・プログラムの本文 (DLI 呼び出しなどのステートメントが含まれます) を作成します。
  3. RRS 同期をコーディングします。
  4. PSB 割り振り解除 (DPSB) 呼び出しをコーディングします。これにより、APSB 呼び出しによって割り振られた PSB が割り振り解除されます。
- RRS 同期呼び出しが ODBA アプリケーションの外部から開始された場合は、次の呼び出しパターンを使用します。
  1. まず PSB 割り振り (APSB) 呼び出しを行って、ODBA アプリケーションに PSB を割り振ります。
  2. APSB 呼び出しの後に、アプリケーション・プログラムの本文 (DLI 呼び出しなどのステートメントが含まれます) を作成します。
  3. DPSB PREP 呼び出しをコーディングします。このアクションにより、アプリケーションが ODBA スレッドにアクセスできなくなります。 IMS リソース (DL/I 呼び出しの一環で取得されたデータベース・バッファーやデータベース・ロックなど) は、RRS 同期が実行されるまで解放されません。
  4. アプリケーション・デプロイヤーに戻ります。
  5. DB2 アプリケーションをデプロイして DB2 リソースを更新します。
  6. RRS 同期を行います。

Db2 for z/OS ストアード・プロシージャー・スレッドの停止

ODBA ストアード・プロシージャー・スレッドを停止するためのオプションはいくつかあります。ハング・スレッドやその他の可用性に関する問題が発生する可能性があるため、推奨されるオプションとそうでないものがあります。

次にリストする方法は、優先的に試す必要がある順序で示されています。

1. 『Db2 for z/OS -STOP PROCEDURE コマンド』
2. 424 ページの『Db2 for z/OS でのスレッドの取り消し』
3. 424 ページの『IMS STOP REGION コマンド』
4. 424 ページの『z/OS MVS コマンド VARY WLM の REFRESH オプション』
5. 424 ページの『IMS のリサイクル』

Db2 for z/OS -STOP PROCEDURE コマンド

Db2 for z/OS -STOP PROCEDURE コマンドは、考えられるマイナス面の副次作用が最も少ないため、最適なスレッド停止方法です。 -STOP PROCEDURE コマンドを使用すると、Db2 for z/OS は 1 つ以上のストアード・プロシージャーの SQL CALL ステートメントを受け入れなくなります。

例えば、次のコマンドはストアード・プロシージャー USERPRC1 および USERPRC3 を停止します。-STOP PROCEDURE コマンドが有効である間、これらのストアード・プロシージャーを実行しようとしても、その試行はキューに入れられます。

```
-STOP PROCEDURE(USERPRC1,USERPRC3)
```

Db2 for z/OS でのスレッドの取り消し

Db2 for z/OS の -DISPLAY THREAD コマンドと -CANCEL THREAD コマンドを使用して、スレッドと保持されているロックを表示したうえで、Db2 for z/OS からスレッドを取り消します。

```
-DISPLAY THREAD(*) TYPE(PROC)
-CANCEL THREAD(token)
```

IMS STOP REGION コマンド

IMS コマンドを使用してハング・スレッドを表示してから、以下のいずれかのコマンドを使用してそれらのスレッドを停止します。

- /STOP REGION *reg#* | *job*
- /STOP REGION *reg#* | *job* ABDUMP *tran*

z/OS MVS コマンド VARY WLM の REFRESH オプション

z/OS MVS VARY WLM コマンドの REFRESH オプションを使用して、WLM 環境をリフレッシュ (すなわち、リサイクル) します。

WLM 環境のリフレッシュは ODBA では問題があります。このリフレッシュでは、ODBA スレッドを終了するための IMS AIB DPSB または TALL 呼び出しを使用せずにストアード・プロシージャーが終了されるためです。通常、ODBA が TERM 要求または TALL 要求を受け取ると、IMS への接続は、アクティブ・スレッドが DRA に戻るのを待ってから終了します。 VARY REFRESH ではこれが許容されないため、ハング・スレッドが起こる可能性があります。

次の例は VARY コマンド構文を示しています。

```
VARY WLM,APPLENV=xxxx,REFRESH
```

IMS のリサイクル

前述の方法でうまくいかない場合は、最後の手段として IMS アドレス・スペースをリサイクルして ODBA スレッドを停止できます。

ODBA アプリケーション・プログラムのテスト

ODBA アプリケーション・プログラムに対してプログラム単体テストを実行し、入力データ、処理、出力データをプログラムが正しく処理することを確かめる必要があります。実行するテストの量およびタイプは、個々のプログラムによって異なります。

プログラムのテストを開始する前に、確立されたテスト手順について確認してください。テストを開始するには、次の項目が必要です。

- テスト JCL ステートメント
- テスト・データベース

  プログラムのテストは、必ずテスト専用データベースで行います。実動データベースではプログラムのテストを行わないでください。プログラムに障害のあるがあると、重要なデータが損傷したり削除されたりする恐れがあります。
- テスト入力データ

  使用する入力データは現行のものである必要はありませんが、有効なデータでなければなりません。有効な入力データを使用しない限り、出力データが有効であることが確認できません。

プログラム・テストの目的は、起こる可能性のあるあらゆる状態を、プログラムが正しく処理できるかどうか確認することです。徹底的にプログラムをテストするには、プログラムがとりうるできるだけたくさんの経路をテストしてください。以下に例を示します。

プログラムに各分岐を実行させる入力データを使用して、プログラムの各経路をテストしてください。プログラムがエラー・ルーチンをテストしていることを確かめてください。 プログラムにできるだけ多くのエラー条件をテストさせる入力データを使用してください。プログラムで使用している編集ルーチンをテストしてください。 プログラムにできるだけたくさんの異なるデータの組み合わせを与えて、入力データが正しく編集されていることを確認してください。次の表には、オンライン (IMSDB)、バッチ、および BMP プログラムをテストするために使用可能なツールをリストします。

表 *62.* プログラムのテストに使用できるツール

ツール	オンライン **(IMS DB)**	バッチ	**BMP**
DFSDDLT0	なし	あり[1]	あり
DL/I イメージ・キャプチャー・プログラム	あり	あり	あり

注**:** 1. 呼び出しレベル・プログラムのみです。(コマンド・レベル・バッチ・プログラムでは、最初に DL/I イメージ・キャプチャー・プログラムを使用して、DFSDDLT0 に関する呼び出しを作成できます。)

ODBA プログラムをテストするためにイメージ・キャプチャーを使用する DL/I 呼び出しトレース

DL/I イメージ・キャプチャー・プログラム (DFSDLTR0) は、バッチ、BMP、およびオンライン (IMS DB 環境) プログラムが出す DL/I 呼び出しをトレースし、記録できるトレース・プログラムです。 DFSDDLT0 に対する入力として使用できる呼び出しを作成することができます。

イメージ・キャプチャー・プログラムを使用して、以下のことを行うことができます。

- プログラムをテストする

イメージ・キャプチャー・プログラムは、トレースしている呼び出しでエラーを検出すると、できるだけ多くの呼び出しを複製します。 ただし、エラーが発生した場所を文書化することはできず、また常にすべての SSA を複製できるとは限りません。

- DFSDDLT0 (DL/I テスト・プログラム) 用入力を作成する

  イメージ・キャプチャー・プログラムが作成した出力を、 DFSDDLT0 への入力として使用することができます。イメージ・キャプチャー・プログラムは、状況ステートメント、コメント・ステートメント、呼び出しステートメント、および比較ステートメントを、DFSDDLT0 用に作成します。例えば、ODBA アプリケーションにイメージ・キャプチャー・プログラムを使用して、 DFSDDLT0 に対する呼び出しを作成することができます。
- プログラムをデバッグする

  プログラムが異常終了する場合、イメージ・キャプチャー・プログラムを使用してプログラムを再実行できます。イメージ・キャプチャー・プログラムは、プログラム障害を引き起こす条件を複製し、文書化します。イメージ・キャプチャー・プログラムが作成した報告書の情報を使用して、問題を検出して修正することができます。

ODBA プログラムをテストするためのイメージ・キャプチャーと DFSDDLT0 との併用

イメージ・キャプチャー・プログラムは、DFSDDLT0 への入力として使用できる次の制御ステートメントを作成します。

- 状況ステートメント

  イメージ・キャプチャー・プログラムを呼び出すと、状況ステートメントが作成されます。状況ステートメントは、以下のことを行います。
  - 印刷オプションを設定します。これにより DFSDDLT0 は、すべての呼び出しトレース・コメント、すべての DL/I 呼び出し、およびすべての比較の結果を印刷します。
  - アプリケーション・プログラムの実行中に PCB に変更が発生するたびに、新規の相対 PCB 番号を判別します。
- コメント・ステートメント

  イメージ・キャプチャー・プログラムを実行すると、コメント・ステートメントも作成されます。 コメント・ステートメントから以下のものが得られます。

  IMS がトレースを開始した時刻および日付

  トレースされている PSB の名前

  またイメージ・キャプチャー・プログラムは、IMS によるエラー検出の対象となるどの呼び出しよりも前に、コメント・ステートメントを作成します。
- 呼び出しステートメント

  イメージ・キャプチャー・プログラムは、 DL/I 呼び出しのそれぞれについて、呼び出しステートメントを作成します。
- 比較ステートメント

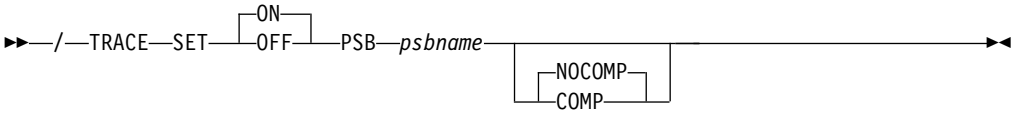
DLITRACE 制御ステートメントに COMP を指定する場合、イメージ・キャプチャー・プログラムはデータおよび PCB 比較ステートメントを作成します。

オンラインでのイメージ・キャプチャーの実行

オンラインでイメージ・キャプチャー・プログラムを実行する場合、トレース出力は IMS ログ・データ・セットに送られます。オンラインでイメージ・キャプチャー・プログラムを実行するには、z/OS コンソールから IMS TRACE コマンドを出してください。

BMP をトレースしている場合、および DFSDDLT0 にトレース結果を使用しようとしている場合、BMP には処理を行っているデータベースへの排他的書き込みアクセスがなければなりません。 アプリケーション・プログラムに排他的アクセスがない場合、 DFSDDLT0 の結果はアプリケーション・プログラムの結果と異なってしまう可能性があります。

次の図は TRACE コマンドの形式を示します。



SET ON|OFF
トレースのオン、オフを切り替えます。

PSB *psbname*
トレースしたい PSB 名を指定します。各 PSB ごとに別々の TRACE コマンドを出すことにより、複数の PSB を同時にトレースすることができます。

COMP|NOCOMP
DFSDDLT0 で使用されるデータおよび PCB 比較ステートメントを、イメージ・キャプチャー・プログラムに作成させるかどうかを指定します。

ログ・データ・セットからのイメージ・キャプチャー・データのリトリーブ

トレース出力が IMS ログ・データ・セットに送信される場合は、ユーティリティー DFSERA10 および DL/I 呼び出しトレース出口ルーチン DFSERA50 を使用してリトリーブできます。

DFSERA50 は、リトリーブされるイメージ・キャプチャー・プログラム・レコードの非ブロック化、形式設定、および番号付けを行います。 DFSERA50 を使用するには、DFSERA10 入力ストリームに、順次出力データ・セットを定義する DD ステートメントを挿入しなければなりません。 この DD ステートメントのデフォルト DD 名は、TRCPUNCH です。カードは、BLKSIZE=80 を指定してください。

例: SYSIN データ・セットに以下の DFSERA10 入力制御ステートメントの例を使用して、ログ・データ・セットからイメージ・キャプチャー・プログラム・データをリトリーブすることができます。

- すべてのイメージ・キャプチャー・プログラム・レコードを印刷する。

```
Column 1       Column 10
OPTION         PRINT OFFSET=5,VALUE=5F,FLDTYP=X
```

- PSB 名ごとに選択したイメージ・キャプチャー・プログラム・レコードを印刷する。

```
Column 1       Column 10
OPTION         PRINT OFFSET=5,VALUE=5F,COND=M
OPTION         PRINT OFFSET=25,VLDTYP=C,FLDLEN=8,
                   VALUE=psbname, COND=E
```

- イメージ・キャプチャー・プログラム・レコードを形式設定する (DFSDDLT0 への入力として使用可能な形式に)。

```
Column 1       Column 10
OPTION         PRINT OFFSET=5,VALUE=5F,COND=M
OPTION         PRINT EXITR=DFSERA50,OFFSET=25,FLDTYP=C
               VALUE=psbname,FLDLEN=8,DDNAME=OUTDDN,COND=E
```

DFSERA50 が使用する DD ステートメントの名前を指定するのに、DDNAME= パラメーターが使用されます。 OUTDDN DD ステートメントに定義されたデータ・セットは、デフォルト TRCPUNCH DD ステートメントの代わりに使用されます。この例では、DD は以下のようになります。

```
//OUTDDN DD ...,DCB=(BLKSIZE=80),...
```

ODBA プログラムのモニターおよびデバッグの要求

ODBA プログラムをデバッグするため、統計 (STAT) またはログ (LOG) 要求を発行できます。

次の要求を使用すると、プログラムをデバッグする際に役立ちます。

- 統計 (STAT) 要求は、データベース統計をリトリーブします。 STAT は、呼び出しレベル・プログラムとコマンド・レベル・プログラムの両方から出すことができます。
- LOG (ログ) 要求により、アプリケーション・プログラムはシステム・ログにレコードを書き込むことができます。 バッチ・プログラムのコマンドまたは呼び出しとして、LOG を出すことができます。この場合、レコードは IMS ログに書き込まれます。 IMS DB 環境のオンライン・プログラムの呼び出しまたはコマンドとして、 LOG を出すことができます。この場合、レコードは IMS DB ログに書き込まれます。

ODBA プログラムの異常終了時の処置

プログラムが異常終了したときに、必ずいくつかの処理をとることができれば、問題の検出および修正の作業を単純化することができます。 ODBA は、いずれの戻りコードまたは理由コードも出しません。ODBA アプリケーション・プログラムでは、終了を伴わないエラーの多くは、AIB 戻りコードおよび理由コードで通知されます。異常終了したプログラムが置かれている状況については、できるだけ多くの情報を記録できます。さらに、ある種の初期設定および実行エラーについても検査できます。

ODBA プログラムの異常終了後の推奨処置

ここでは、プログラムが異常終了した場合の、共通に使用できるガイドラインをいくつか提案します。

- エラー状態を文書化して、調査および訂正に役立ててください。役立つ情報のいくつかを、以下に示します。
  - プログラムの PSB 名
  - 呼び出し機能
  - 端末装置 ID (オンライン・プログラムのみ)
  - AIB あるいは PCB の内容
  - 問題が発生した際の入出力域の内容
  - データベース要求が実行されていた場合、(もしあれば) その要求が使用する SSA あるいは SEGMENT および WHERE オプション
  - 日時
- プログラムでエラーが発生した場合、すべての必要なエラー情報を標準エラー・ルーチンに渡すことができます。
- LOG 要求を出してシステム・ログにメッセージを送信することができます。

ODBA プログラムの異常終了の診断

テストしているプログラムまたは実行しているプログラムが正しく稼働しない場合は、問題を分離する必要があります。 問題はプログラミング・エラー (例えば、要求したうちの一つをコーディングした方法のエラー) から、システム問題までのものである可能性があります。

プログラムの実行の失敗時、異常終了時、または誤った結果の出力時には、以下のエラーを検査することができます。

ODBA 初期設定エラー

プログラムが制御を受け取る前に、IMS は、アプリケーション・プログラムが使用している PSB および DBD を正しくロードし、初期設定しなければなりません。この領域に問題があると、システム・プログラマーまたは DBA (あるいは同等の資格の専門家) に修正を依頼することがしばしば必要になります。アプリケーション・プログラマーが行えるのは、DBD、PSB、および生成された制御ブロックに最近変更があったかどうかを見つけ出すことです。

ODBA 実行エラー

初期設定エラーがない場合には、プログラムの以下の点を検査してください。

1. コンパイラーからの出力。すべてのエラー・メッセージが解決されているかどうかを確かめてください。
2. バインダーからの出力。
   - すべての外部参照が解決されていますか?
   - すべての必要なモジュールが組み込まれていましたか?
   - 言語インターフェース・モジュールが正しく組み込まれていましたか?
3. 使用中の JCL。 データベースを含むファイルを説明した情報は正確ですか? 正しくない場合、DBA に相談してください。

第 23 章 DRDA のための IMS サポートを使用したプログラミング

IMS では、Distributed Relational Database Architecture™ (DRDA) プロトコルを実装しており、これを使用して、独自の IMS Connect TCP/IP クライアント・アプリケーションを作成できます。

DRDA は、多様なプラットフォーム上にあるアプリケーションとデータベース・システムとの間の通信を可能にする、オープン・アーキテクチャーです。DRDA プロトコルを使用したデータベース・アクセス操作の実行の詳細については、DRDA の公開仕様に記載されています。続く説明では、DRDA のための IMS サポートが提供する IMS 固有の拡張機能についてのみ記しています。

DRDA のための IMS サポートを使用するには、DRDA クライアント・ドライバー (DRDA ソース・サーバー) を作成する必要があります。クライアント・システムで追加のソフトウェアをインストールまたは構成する必要はありません。 DRDA ターゲット・サーバーは、z/OS の IMS とともに稼働する、IMS Connect および Open Database Manager (ODBM) で構成されます。

DRDA のための IMS サポートには、アプリケーション指向トランザクション区分 (local) および XA 対応 (グローバル) トランザクションの両方のサポートがあります。

IMS は、以下の DRDA 機能はサポートしません。

- 複数行入力
- クライアント・リルート
- セキュリティー・プラグイン

DRDA のための IMS サポートは、DRDA バージョン 4 技術標準を基本としています。 DRDA の仕様は、The Open Group Consortium (http://www.opengroup.org/) によって文書化されています。

サーバー互換性検査

ソース DRDA サーバーとターゲット DRDA サーバー間のすべての通信は、初期化とセキュリティー検査から開始されます。初期化フローでは、DRDA クライアントが EXCSAT コマンドを発行し、DRDA ターゲット・サーバーから EXCSATRD データ・オブジェクトが返信されます。

DRDA のための IMS サポートの実装では、EXCSATRD 応答データ・オブジェクトの中には、サーバー・リリース・レベル (SRVRLSLV) パラメーターが含まれます。 SRVRLSLV パラメーターは、分散データベース管理 (DDM) 言語のバージョン番号を指定するストリングであり、IMS Connect および ODBM サーバー・コンポーネントにより認識されます。このストリングは、クライアントが送信するすべてのコード・ポイントを IMS Connect と ODBM の両方が認識することを確認するための、クライアントによるサーバー互換性検査の実行に使用されます。 DDM

バージョン番号は、DRDA のための IMS サポートに固有のものです。 DRDA のための IMS サポートの場合、すべての互換性検査は、SRVRLSLVL パラメーターに基づいて実施されます。

重要: EXCSAT コマンドに対する応答としてターゲット・サーバーから返される SRVRLSLV パラメーターの値は、OD-ICON 1 OD-ODBM 1 です。

IMS の最新のメンテナンス・リリースでソース・サーバーを更新した場合、ご使用のすべての IMS Connect または ODBM インストール済み環境にその同じメンテナンス・リリースを適用しないと、ソース・サーバーとターゲット・サーバーとの非同期の原因となる可能性があります。この可能性を回避するため、サーバー互換性検査では、ソース・サーバーが使用する DDM バージョン・レベルを、DRDA のための IMS サポート・ターゲット・サーバーが認識できる場合に限り、接続を許可します。

IMS データと DRDA プロトコルの対応

DRDA のための IMS サポートでのデータベース照会操作において、行 とは、aibdbpcbStream データ構造のインスタンスに、IMS 階層パス内の要求されたすべてのフィールドを加えた連結と定義できます。 aibdbpcbStream インスタンスは、aibStream データ構造のインスタンスと、それに続く dbpcbStream データ構造のインスタンスとを連結したものです。要求されるフィールドは、OPNQRY コマンドで送信される RTRVFLD オブジェクトで表されます。 aibdbpcbStream インスタンスとデータ・フィールドとの連結は、照会行セット内の単一行として表されます。

DRDA のための IMS サポートでは、行または戻される結果セットのサイズに応じて各照会ブロックのサイズが異なっていても構わない、柔軟なブロッキングのみをサポートします。指定された照会ブロック・サイズは初期サイズとして使用され、取り出し処理を実行するために必要であれば、照会ブロックはこのサイズより大きくできます。

DRDA のための IMS サポートの実装では、データは DRDA ターゲット・サーバーからバイト・ストリーム形式で戻され、データ・タイプの処理はクライアントが受け持ちます。

関連概念:

CSL オープン・データベース・マネージャーの概要 (システム管理)

IMS DB へのアクセスに対する IMS Connect のサポート (コミュニケーションおよびコネクション)

関連資料:

DRDA DDM コマンド・アーキテクチャーの参照情報 (アプリケーション・プログラミング API)

DRDA のための IMS サポートを使用したデータ操作用の DDM コマンド

DRDA のための IMS サポートが提供する分散データベース管理 (DDM) コマンドは、singleton およびバッチでのデータ操作用に使用します。

データベースにアクセスする前に、まず DRDA クライアント・アプリケーションから ACCRDB コマンドを実行してデータベース接続を確立し、DRDA ターゲット・サーバーから戻される ACCRDBRM データ・オブジェクトを正常に受信できるようにする必要があります。

接続を確立した後は、DDM コマンドを発行することで、DRDA クライアント・アプリケーションからデータにアクセスできます。

- データをリトリーブするには、OPNQRY コマンドを発行します。
- データを挿入、更新、または削除するには、EXCSQLIMM コマンドを発行します。

データ操作は、singleton またはバッチ操作で実行できます。データ操作のタイプは、DDM コマンドにチェーニングされた DLIFUNC コマンド・オブジェクトに、バイト・ストリング・データ表記 (BYTSTRDR) パラメーターを設定することで指定します。

次の表では、DRDA のための IMS サポートでのデータ操作のために DRDA クライアントが発行する DDM コマンドを示しています。

表 *63. DRDA* のための *IMS* サポートを使用したデータ操作用の *DDM* コマンド

データ操作	**DDM** コマンド	DLIFUNC コマンド・オブジェクト用の **BYTSTRDR** パラメーター値
データ挿入	EXCSQLIMM	ISRT
データ・リトリーブ - DL/I Get Hold Unique	OPNQRY	GHU
データ・リトリーブ - DL/I Get Unique	OPNQRY	GU
データ・リトリーブ - DL/I Get Hold Next	OPNQRY	GHN
データ・リトリーブ - DL/I Get Next	OPNQRY	GN
データ・リトリーブ - DL/I Get Hold Next Within Parent	OPNQRY	GHNP
データ・リトリーブ - DL/I Get Next Within Parent	OPNQRY	GNP
データ更新	EXCSQLIMM	REPL
データ削除	EXCSQLIMM	DLET

次の表では、DRDA のための IMS サポートでのバッチ・データ操作のために DRDA クライアントが発行する DDM コマンドを示しています。

表 *64. DRDA* のための *IMS* サポートを使用したバッチ・データ操作用の *DDM* コマンド

バッチ・データ操作	**DDM** コマンド	DLIFUNC コマンド・オブジェクト用の **BYTSTRDR** パラメーター値
データ・リトリーブ	OPNQRY	RETRIEVE

表 *64. DRDA* のための *IMS* サポートを使用したバッチ・データ操作用の *DDM* コマンド (続き)

バッチ・データ操作	**DDM** コマンド	DLIFUNC コマンド・オブジェクト用の **BYTSTRDR** パラメーター値
データ更新	EXCSQLIMM	UPDATE
データ削除	EXCSQLIMM	DELETE

関連資料:

DRDA DDM コマンド・アーキテクチャーの参照情報 (アプリケーション・プログラミング API)

第 3 部 IMS TM 用のアプリケーション・プログラミング

IMS では、IMS トランザクションにアクセスするアプリケーション・プログラムの作成がサポートされています。

第 24 章 IMS TM のアプリケーション・プログラム・エレメントの定義

アセンブラー言語、C、COBOL、Java、Pascal、または PL/I で DL/I 呼び出しを使用して、IMS Transaction Manager と通信するアプリケーション・プログラムを作成できます。

言語インターフェースのための DL/I 呼び出しのフォーマット設定

アセンブラー言語、C 言語、COBOL、Pascal、または PL/I おける DL/I 呼び出しを使用する場合には、DL/I 言語インターフェースを呼び出して、DL/I 呼び出しで指定された機能を開始する必要があります。

IMS では、DL/I 呼び出しへのインターフェースには以下のものがあります。

- 言語環境プログラム準拠の任意のプログラムのための言語非依存インターフェース (CEETDLI)
- サポートされるすべての言語に対応する言語固有の各インターフェース (xxxTDLI)
- サポートされるすべての言語に対応する言語固有でないインターフェース (AIBTDLI)

Java では、3 つすべての DL/I 言語インターフェースを使用しますが、その使用は内部的なものであるため、DL/I 呼び出しで指定した機能を開始する際に、呼び出しは必要ありません。

関連概念:

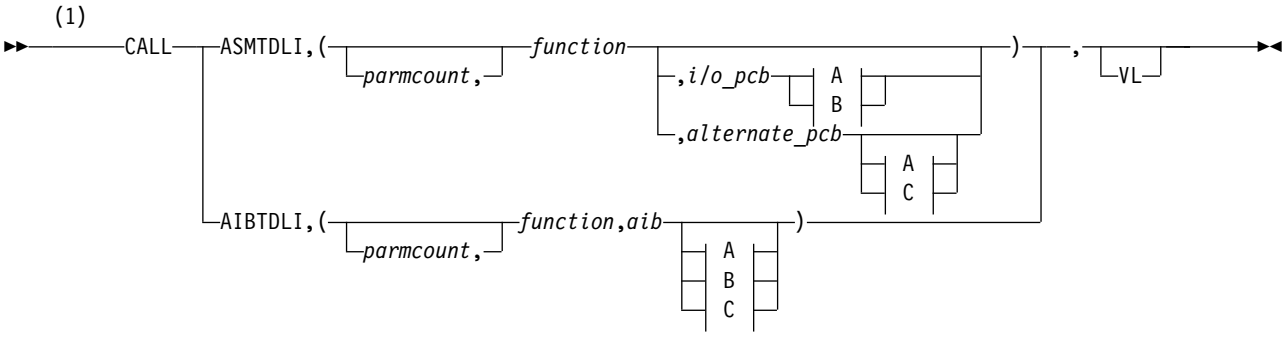
693 ページの『第 38 章 Java 開発用の IMS ソリューションの概要』

アセンブラー言語によるアプリケーション・プログラミング

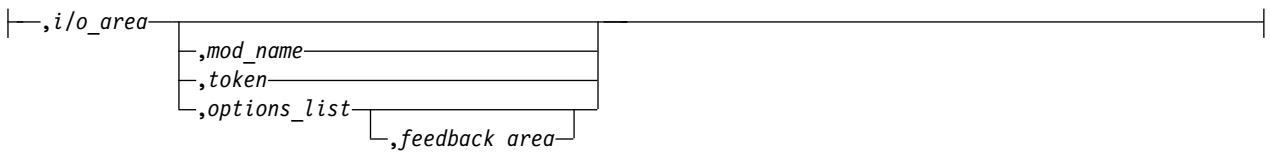
アセンブリー言語によるアプリケーション・プログラムは、以下の形式、パラメーター、および DL/I 呼び出しを使用して、IMS Transaction Manager と通信します。

アセンブラー言語プログラムでは、アドレスとして渡されるすべての DL/I 呼び出しパラメーターはレジスターで渡すことができます。ただし、使用する際には括弧で囲まなければなりません。

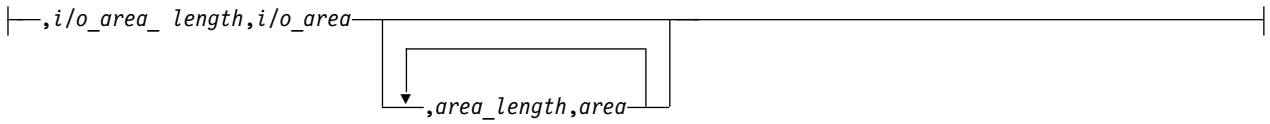
フォーマット



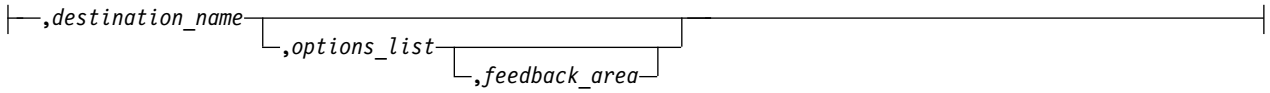
A:



B:



C:



注**:**

1 アセンブラー言語のプログラムでは、必ず *parmcount* か VL のいずれかを使用してください。

パラメーター

parmcount
ユーザー定義ストレージの、4 バイト・フィールドのアドレスを指定します。ここには、*parmcount* の後に続くパラメーター・リスト内のパラメーターの数が入ります。アセンブラー言語のアプリケーション・プログラムでは、必ず *parmcount* か VL のいずれかを使用してください。

function
ユーザー定義ストレージの、4 バイトのフィールドのアドレスを指定します。ここには、使用する呼び出し機能が入ります。呼び出し機能は必ず左寄せし、ブランクを埋め込む必要があります。例えば、(GUbb) は呼び出し機能です。

i/o pcb
入出力プログラム連絡ブロック (PCB) のアドレスを指定します。 I/O PCB アドレスは、以下の環境で、アプリケーション・プログラムへの入り口で PCB リストで渡される最初のアドレスです。

- DLI 領域またはデータベース管理バッチ (DBB) 領域 (PSB で CMPAT=YES がコーディングされている) で実行中のプログラム
- CMPAT= 値にかかわらず、バッチ・メッセージ処理プログラム (BMP) 領域、メッセージ処理プログラム (MPP) 領域または IMS 高速機能 (IFP) 領域で実行中の任意のプログラム

alternate pcb
呼び出しで使用する代替 PCB のアドレスを指定します。PCB アドレスは、アプリケーション・プログラムの入り口で PCB リストに入れて渡される PCB アドレスの 1 つでなければなりません。

aib
ユーザー定義ストレージのアプリケーション・インターフェース・ブロック (AIB) のアドレスを指定します。

i/o area
呼び出しで使用するユーザー定義ストレージの、入出力域のアドレスを指定します。入出力域は、返されるデータが入るだけの大きさがなければなりません。

i/o area length
ユーザー定義ストレージの、4 バイトのフィールドのアドレスを指定します。ここには、入出力域の長さが入ります (2 進数で指定)。

area length
ユーザー定義ストレージの、4 バイトのフィールドのアドレスを指定します。ここには、パラメーター・リスト内のすぐあとの区域の長さが入ります (2 進数で指定)。区域の長さと区域は最大 7 組指定できます。

area
チェックポイントをとるユーザー定義ストレージの区域のアドレスを指定します。区域の長さと区域は最大 7 組指定できます。

token
ユーザー定義ストレージの、4 バイトのフィールドのアドレスを指定します。ここには、ユーザー・トークンが入ります。

options list
ユーザー定義ストレージの、*options list* のアドレスを指定します。ここには、その呼び出しで使用する処理オプションが入ります。

feedback area
オプション・リストの処理エラーに関する情報を受信する、ユーザー定義ストレージのフィードバック域のアドレスを指定します。

mod name
ユーザー定義ストレージの、8 バイトの区域のアドレスを指定します。ここには、その呼び出しで使用するユーザー定義の MOD 名が入ります。*mod name* パラメーターは、MFS でのみ使用します。

destination name
ユーザー定義ストレージの、8 バイトのフィールドのアドレスを指定します。ここには、呼び出しの結果メッセージが送られる論理端末の名前またはトランザクション・コードが入ります。

VL パラメーター・リストの終わりを示します。アセンブラー言語のプログラムでは、必ず *parmcount* か VL のいずれかを使用してください。

DL/I 呼び出しの形式の例

DL/I AIBTDLI インターフェース:

```
CALL  AIBTDLI,(function,aib,i/o area),VL
```

DL/I 言語固有のインターフェース:

```
CALL  ASMTDLI,(function,i/o pcb,i/o area),VL
```

関連概念:

279 ページの『AIBTDLI インターフェース』

関連資料:

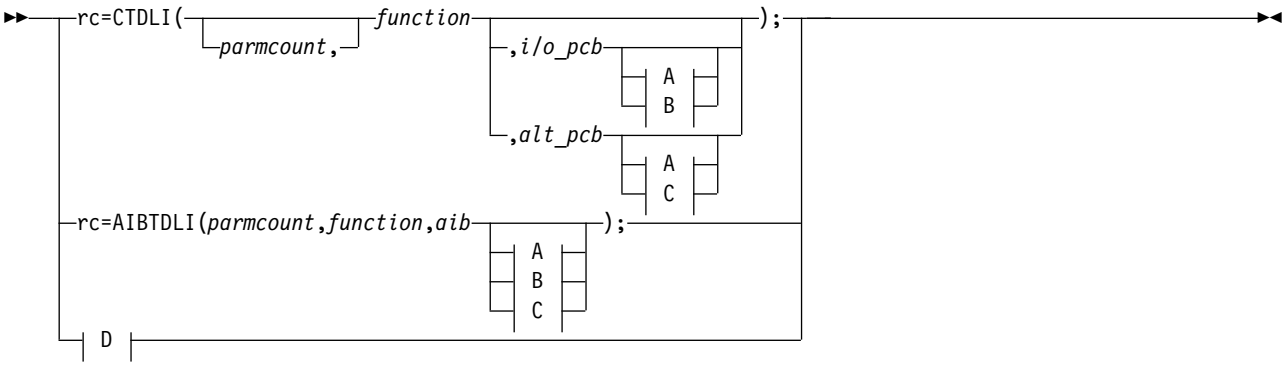
トランザクション管理のための DL/I 呼び出し (アプリケーション・プログラミング API)

IMS TM システム・サービスのための DL/I 呼び出し (アプリケーション・プログラミング API)

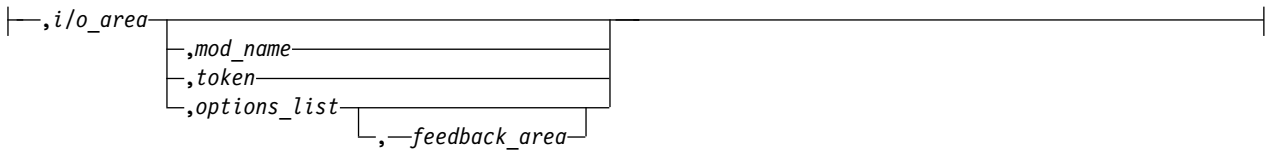
C 言語によるアプリケーション・プログラミング

C 言語によるアプリケーション・プログラムは、以下の形式、パラメーター、および DL/I 呼び出しを使用して、IMS Transaction Manager と通信します。

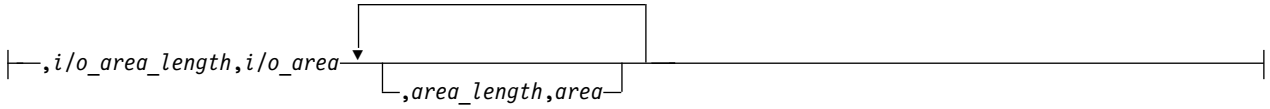
フォーマット



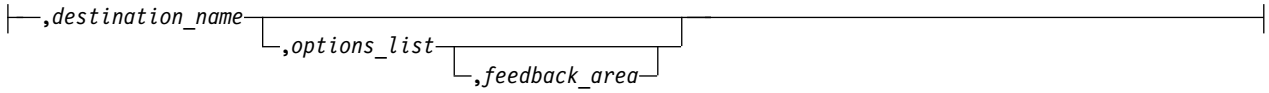
A:



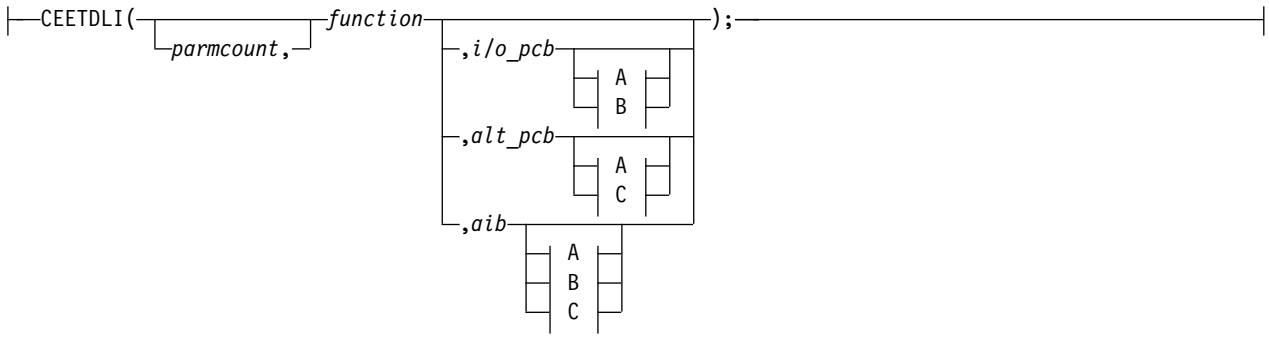
B:



C:



D:



パラメーター

rr DL/I 状況コードまたは戻りコードを受け取ります。これは 2 文字のフィールドで、整変数 (*int*) の低位の 2 バイトに入ります。状況コードまたは戻りコードがブランク 2 個であれば、このフィールドには 0 が入ります。rc パラメーターは、if ステートメントでテストすることができます。例えば、`if (rc == 'IX')` のようになります。また、switch ステートメントで rc を使用することもできます。 rc に入っている値を無視し、その代わりに、プログラム連絡ブロック (PCB) に返された状況コードを使用することを選択することもできます。

parmcount
ユーザー定義ストレージの固定 2 進 (31) 変数の名前を指定します。これは、*parmcount* の後に続くパラメーター・リスト内のパラメーターの数が入るポインターです。*parmcount* フィールドは長さを指すポインターです。

function
ユーザー定義ストレージの文字 (4) 変数の名前を左寄せして指定します。ここには、使用される呼び出し機能が入ります。呼び出し機能にはブランクを埋め込みます。例えば、(GUbb) は呼び出し機能です。

i/o pcb
I/O PCB のアドレスを指定します。I/O PCB アドレスは、以下の環境で、アプリケーション・プログラムへの入り口で PCB リストで渡される最初のアドレスです。

- DLI 領域またはデータベース管理バッチ (DBB) 領域 (PSB で `CMPAT=YES` がコーディングされている) で実行中のプログラム
- `CMPAT=` 値にかかわらず、バッチ・メッセージ処理プログラム (BMP) 領域、メッセージ処理プログラム (MPP) 領域または IMS 高速機能 (IFP) 領域で実行中の任意のプログラム

alternate pcb
ポインター変数の名前を指定します。ここには、呼び出しで使用される I/O PCB アドレスまたは代替 PCB アドレスが含まれます。 PCB アドレスは、常

に、アプリケーション・プログラムへの入り口で PCB リストで渡される PCB アドレスの 1 つでなければなりません。

aib
ポインター変数の名前を指定します。ここには、ユーザー定義ストレージのアプリケーション・インターフェース・ブロック (AIB) を定義する構造のアドレスが入ります。

i/o area
呼び出しで使用するユーザー定義ストレージの入出力域を定義する、大構造、配列、または文字ストリングのポインター変数の名前を指定します。入出力域は、返されるデータが入るだけの大きさがなければなりません。

i/o area length
ユーザー定義ストレージの固定 2 進 (31) 変数の名前を指定します。ここには、入出力域の長さが入ります。

area length
ユーザー定義ストレージの固定 2 進 (31) 変数の名前を指定します。ここには、パラメーター・リスト内のすぐあとの区域の長さが入ります。区域の長さと区域は最大 7 組指定できます。

area
ポインター変数の名前を指定します。ここには、チェックポイントをとるユーザー定義ストレージを定義する構造のアドレスが入ります。区域の長さと区域は最大 7 組指定できます。

token
ユーザー定義ストレージの文字 (4) 変数の名前を指定します。ここには、ユーザー・トークンが入ります。

options list
ポインター変数の名前を指定します。ここには、呼び出しで使用される処理オプションが入るユーザー定義ストレージを定義する構造のアドレスが入ります。

feedback area
ポインター変数の名前を指定します。ここには、オプション・リストの処理エラーに関する情報を受け取るユーザー定義ストレージを定義する構造のアドレスが入ります。

mod name
ユーザー定義ストレージの文字 (8) 変数の名前を指定します。ここには、呼び出しで使用するユーザー定義 MOD 名が入ります。*mod name* パラメーターは、MFS でのみ使用します。

destination name
ユーザー定義ストレージの文字 (8) 変数の名前を指定します。ここには、呼び出しの結果のメッセージが送られる論理端末の名前またはトランザクション・コードが入ります。

入出力域

C 言語では、入出力域は、構造または配列も含めてどのタイプでもかまいません。leawi.h の中の ceetdli 宣言および ims.h の中の ctdli 宣言ではプロトタイプ情報が指定されないため、パラメーターのタイプ検査は行われません。入出力域は、

auto、static、または allocated (malloc または calloc を伴う) とすることができます。 C 言語の規則であるヌルによるストリングの終端 ('¥0') を DL/I は認識しないため、C 言語のストリングについては特に注意を払ってください。strcpy および strcmp 関数の代わりに、memcpy および memcmp 関数を使用する場合があります。

DL/I 呼び出しの形式の例

DL/I CEEDTLI インターフェース:

```
#include <leawi.h>
ceetdli(function,aib,i/o_area)
```

DL/I AIBTDLI インターフェース:

```
int rc;
.
.
.
rc = aibtdli(parmcount,function,aib,i/o_area)
```

DL/I 言語固有のインターフェース:

```
#include <ims.h>
int rc;
.
.
.
rc = ctdli(function,i/o_pcb,i/o_area)
```

関連概念:

279 ページの『AIBTDLI インターフェース』

関連資料:

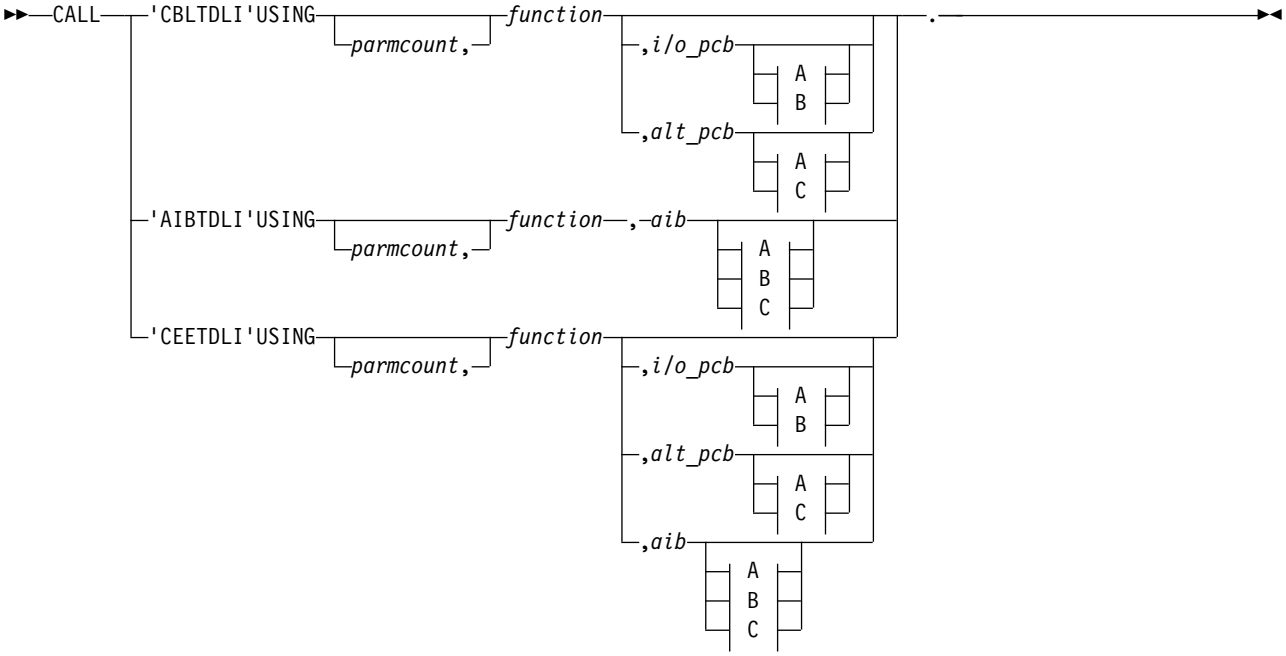
トランザクション管理のための DL/I 呼び出し (アプリケーション・プログラミング API)

IMS TM システム・サービスのための DL/I 呼び出し (アプリケーション・プログラミング API)

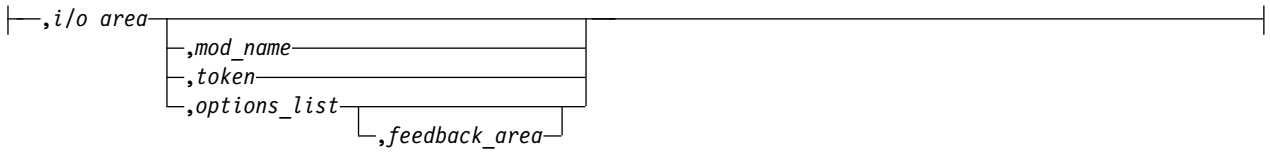
COBOL によるアプリケーション・プログラミング

COBOL によるアプリケーション・プログラムは、以下の形式、パラメーター、および DL/I 呼び出しを使用して、IMS Transaction Manager と通信します。

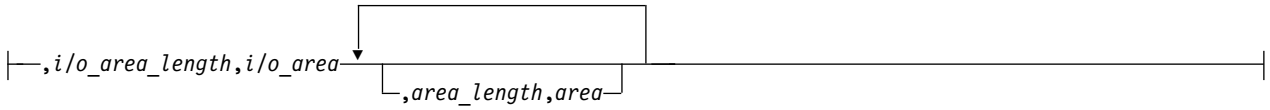
フォーマット



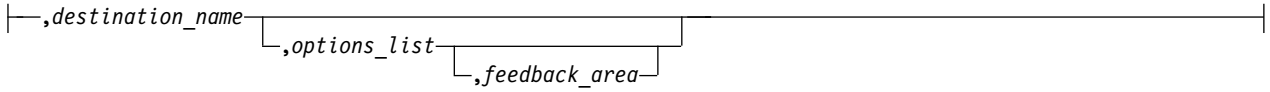
A:



B:



C:



パラメーター

parmcount
パラメーター・リストにおける *parmcount* の後のパラメーターの数が入る、ユーザー定義ストレージ内の使用法 2 進 (4) バイト・データ項目の ID を指定します。

function
ユーザー定義ストレージの Usage Display (4) バイト・データ項目の ID を、

左寄せで指定します。ここには、使用する呼び出し機能が入ります。呼び出し機能にはブランクを埋め込みます。例えば、(GUbb) は呼び出し機能です。

i/o pcb
入出力プログラム連絡ブロック (PCB) のアドレスを指定します。 I/O PCB アドレスは、以下の環境で、アプリケーション・プログラムへの入り口で PCB リストで渡される最初のアドレスです。

- DLI 領域またはデータベース管理バッチ (DBB) 領域 (PSB で `CMPAT=YES` がコーディングされている) で実行中のプログラム
- `CMPAT=` 値にかかわらず、バッチ・メッセージ処理プログラム (BMP) 領域、メッセージ処理プログラム (MPP) 領域または IMS 高速機能 (IFP) 領域で実行中の任意のプログラム

alternate pcb
エントリーにあるアプリケーション・プログラムへ渡された PCB リストから、I/O PCB または代替 PCB グループ項目の ID を指定します。この ID は呼び出しで使用されます。

aib
ユーザー定義ストレージ内のアプリケーション・インターフェース・ブロック (AIB) を定義するグループ項目の ID を、指定します。

i/o area
呼び出しに使用する入出力域を定義する、グループ項目、テーブル、または Usage Display データ項目の ID を指定します。入出力域は、返されるデータが入るだけの大きさがなければなりません。

i/o area length
ユーザー定義ストレージの Usage Binary (4) バイト・データ項目の ID を指定します。ここには、入出力域の長さが入ります。

area length
ユーザー定義ストレージの Usage Binary (4) バイト・データ項目の ID を指定します。ここには、パラメーター・リスト内のすぐあとの区域の長さが入ります。区域の長さと区域は最大 7 組指定できます。

area
チェックポイントをとる区域を定義するグループ項目の ID を指定します。区域の長さと区域は最大 7 組指定できます。

token
Usage Display (4) バイト・データ項目の ID を指定します。ここには、ユーザー・トークンが入ります。

options list
ユーザー定義ストレージを定義するグループ項目の ID を指定します。ここには、呼び出しで使用する処理オプションが入ります。

feedback area
オプション・リストの処理エラーに関する情報を受け取るユーザー定義ストレージを定義するグループ項目の ID を指定します。

mod name
ユーザー定義ストレージの Usage Display (8) バイト・データ項目の ID を指定します。ここには、呼び出しで使用するユーザー定義 MOD 名が入ります。

destination name
Usage Display (8) バイト・データ項目の ID を指定します。ここには、呼び出しの結果、メッセージが送られる論理端末の名前またはトランザクション・コードが入ります。

DL/I 呼び出しの形式の例

DL/I CEETDLI インターフェース:

```
CALL 'CEETDLI' USING function, aib,i/o area.
```

DL/I AIBTDLI インターフェース:

```
CALL 'AIBTDLI' USING function, aib,i/o area.
```

DL/I 言語固有のインターフェース:

```
CALL 'CBLTDLI' USING function, i/o pcb, i/o area.
```

関連概念:

279 ページの『AIBTDLI インターフェース』

関連資料:

トランザクション管理のための DL/I 呼び出し (アプリケーション・プログラミング API)

IMS TM システム・サービスのための DL/I 呼び出し (アプリケーション・プログラミング API)

IMS 用の Java アプリケーション・プログラミング

IMS は Java プログラミング言語を使用したアプリケーション開発にサポートを提供します。

Java 開発用の IMS ソリューションのドライバーおよびリソース・アダプターを使用して、IMS データベースへのアクセスや IMS トランザクションの処理を実行する Java アプリケーションを作成できます。

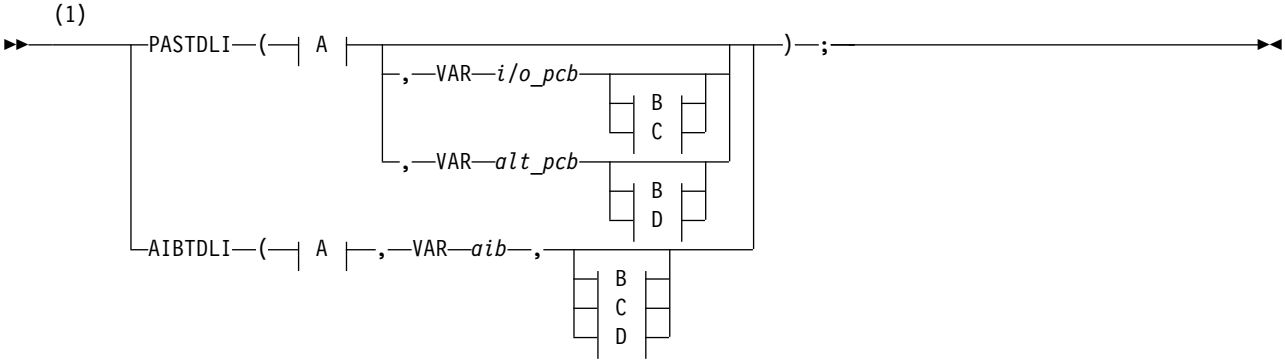
関連概念:

693 ページの『第 38 章 Java 開発用の IMS ソリューションの概要』

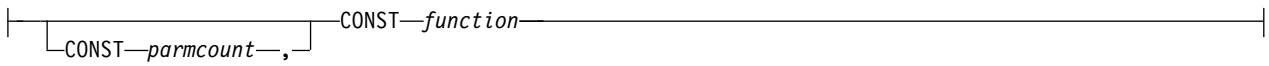
Pascal のアプリケーション・プログラミング

Pascal によるアプリケーション・プログラムは、以下の形式、パラメーター、および DL/I 呼び出しを使用して、IMS Transaction Manager と通信します。

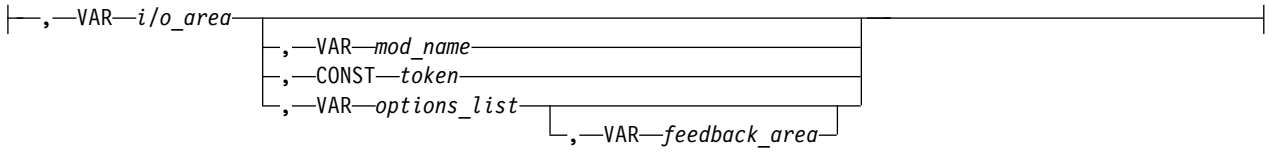
フォーマット



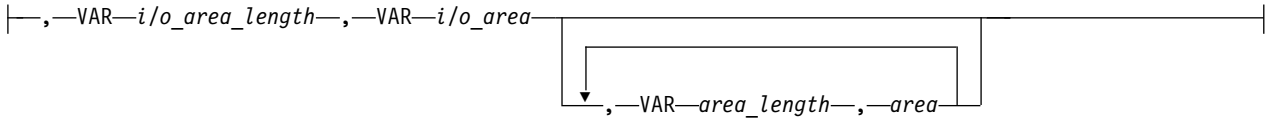
A:



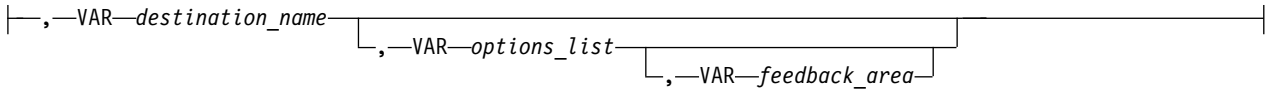
B:



C:



D:



注:

1 AIBTDLI の場合、アプリケーションには *parmcount* が必須です。

パラメーター

parmcount
ユーザー定義ストレージの固定 2 進 (31) 変数のアドレスを指定します。ここには、*parmcount* の後に続くパラメーター・リスト内のパラメーターの数が入ります。

function
ユーザー定義ストレージの文字 (4) 変数の名前を左寄せして指定します。ここには、使用される呼び出し機能が入ります。呼び出し機能にはブランクを埋め込みます。例えば、(GUbb) は呼び出し機能です。

i/o pcb
プログラム連絡ブロック (I/O PCB) のアドレスを指定します。入出力 PCB アドレスは、以下の環境で、アプリケーション・プログラムへの入り口で PCB リストで渡される最初のアドレスです。

- DLI 領域またはデータベース管理バッチ (DBB) 領域 (PSB で CMPAT=YES がコーディングされている) で実行中のプログラム
- CMPAT= 値にかかわらず、バッチ・メッセージ処理プログラム (BMP) 領域、メッセージ処理プログラム (MPP) 領域または IMS 高速機能 (IFP) 領域で実行中の任意のプログラム

alternate pcb
ポインター変数の名前を指定します。ここには、呼び出しプロシージャー・ステートメントで定義された I/O PCB アドレスが含まれます。

aib
ポインター変数の名前を指定します。ここには、ユーザー定義ストレージのアプリケーション・インターフェース・ブロック (AIB) を定義する構造のアドレスが入ります。

i/o area
呼び出しで使用するユーザー定義ストレージの入出力域を定義する、大構造、配列、または文字ストリングのポインター変数の名前を指定します。入出力域は、返されるデータが入るだけの大きさがなければなりません。

i/o area length
ユーザー定義ストレージの固定 2 進 (31) 変数の名前を指定します。ここには、入出力域の長さが入ります。

area length
ユーザー定義ストレージの固定 2 進 (31) 変数の名前を指定します。ここには、パラメーター・リスト内のすぐあとの区域の長さ (2 進数で指定) が入ります。区域の長さと区域は最大 7 組指定できます。

area
ポインター変数の名前を指定します。ここには、チェックポイントをとるユーザー定義ストレージの区域を定義する構造のアドレスが入ります。区域の長さと区域は最大 7 組指定できます。

token
ユーザー定義ストレージの文字 (4) 変数の名前を指定します。ここには、ユーザー・トークンが入ります。

options list
ポインター変数の名前を指定します。ここには、呼び出しで使用される処理オプションが入るユーザー定義ストレージを定義する構造のアドレスが入ります。

feedback area
ポインター変数の名前を指定します。ここには、オプション・リストの処理エラーに関する情報を受け取るユーザー定義ストレージを定義する構造のアドレスが入ります。

mod name
ユーザー定義ストレージの文字 (8) 変数の名前を指定します。ここには、呼び出しで使用するユーザー定義 MOD 名が入ります。

destination name
ユーザー定義ストレージの文字 (8) 変数の名前を指定します。ここには、呼び出しの結果、メッセージが送られる論理端末名またはトランザクション・コードが入ります。

DL/I 呼び出しの形式の例

DL/I AIBTDLI インターフェース:

```
AIBTDLI(CONST function,
        VAR aib,
        VAR I/O area);
```

DL/I 言語固有のインターフェース:

```
PASTDLI(CONST function,
area    VAR I/O PCB
        VAR I/O area);
```

関連概念:

279 ページの『AIBTDLI インターフェース』

関連資料:

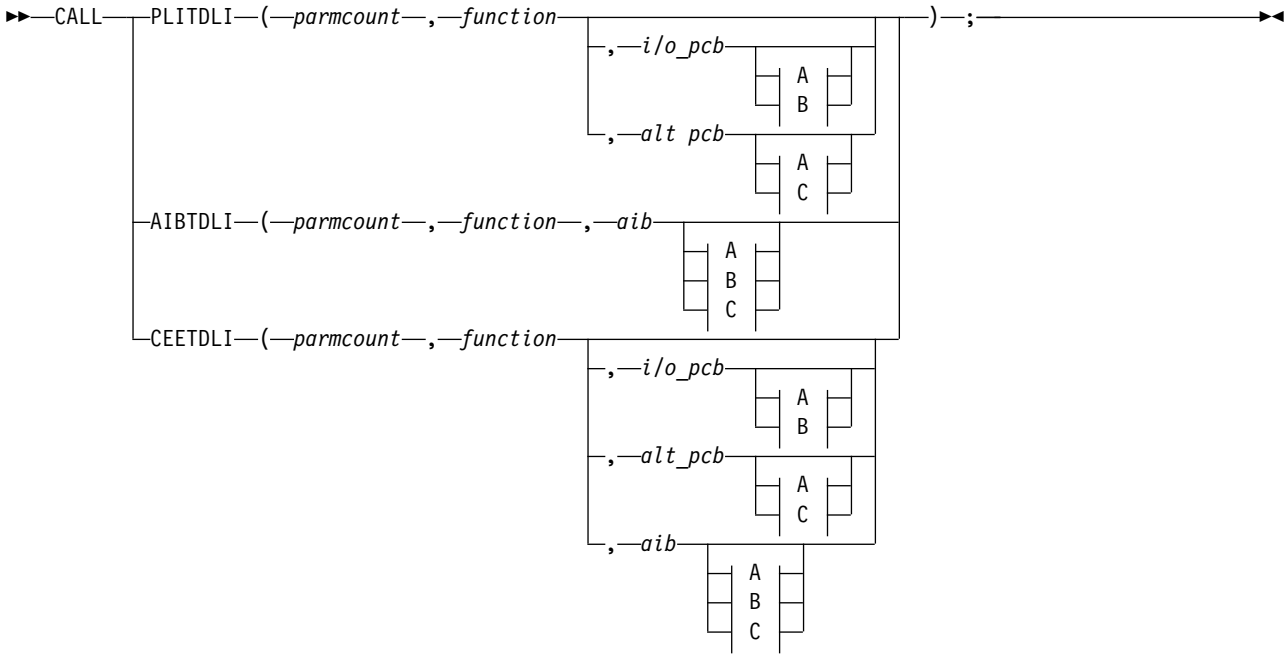
トランザクション管理のための DL/I 呼び出し (アプリケーション・プログラミング API)

IMS TM システム・サービスのための DL/I 呼び出し (アプリケーション・プログラミング API)

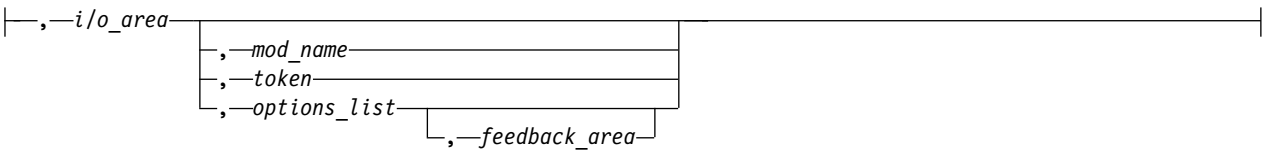
PL/I のアプリケーション・プログラミング

PL/I によるアプリケーション・プログラムは、以下の形式、パラメーター、および DL/I 呼び出しを使用して、IMS Transaction Manager と通信します。

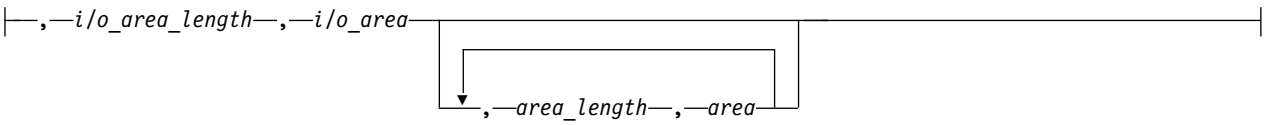
フォーマット



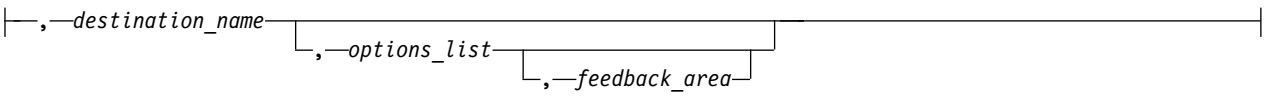
A:



B:



C:



パラメーター

parmcount

固定 2 進 (31 バイト) 変数の名前を指定します。ここには、*parmcount* の後に続く引数の数が入ります。

function
文字 (4 バイト) 変数の名前を、左寄せで、ブランクを埋め込んだ文字ストリングで指定します。ここには、使用される呼び出し機能が入ります。例えば、(GUbb) は呼び出し機能です。

i/o pcb
プログラム連絡ブロック (I/O PCB) のアドレスを指定します。入出力 PCB アドレスは、以下の環境で、アプリケーション・プログラムへの入り口で PCB リストで渡される最初のアドレスです。

- DLI 領域または DBB 領域 (PSB で `CMPAT=YES` がコーディングされている) で実行中のプログラム
- `CMPAT=` 値にかかわらず、バッチ・メッセージ処理プログラム (BMP) 領域、メッセージ処理プログラム (MPP) 領域または IMS 高速機能 (IFP) 領域で実行中の任意のプログラム

alternate pcb
呼び出しで使用される I/O PCB または代替 PCB に関連した構造を指定します。この構造は、アプリケーション・プログラムへの入り口で渡される PCB アドレスの 1 つでなければならない PCB アドレスに基づいています。

aib
アプリケーション・インターフェース・ブロック (AIB) を定義する構造の名前を指定します。

i/o area
呼び出しで使用する入出力域の名前を指定します。入出力域は、返されるデータが入るだけの大きさがなければなりません。

i/o area length
ユーザー定義ストレージの固定 2 進 (31) 変数の名前を指定します。ここには、入出力域の長さが入ります (2 進数で指定)。

area length
固定 2 進 (31) 変数の名前を指定します。ここには、パラメーター・リスト内のすぐあとの区域の長さ (2 進数で指定) が入ります。区域の長さと区域は最大 7 組指定できます。

area
チェックポイントをとる区域の名前を指定します。区域の長さと区域は最大 7 組指定できます。

token
文字 (4) 変数の名前を指定します。ここには、ユーザー・トークンが入ります。

options list
構造の名前を指定します。ここには、呼び出しで使用する処理オプションが入ります。

feedback area
オプション・リストの処理エラーに関する情報を受け取る構造の名前を指定します。

mod name
文字 (8) 変数文字ストリングの名前を指定します。ここには、呼び出しで使用するユーザー定義の MOD 名が入ります。

destination name
文字 (8) 変数文字ストリングの名前を指定します。ここには、呼び出しの結果を示すメッセージが送られる論理端末の名前またはトランザクション・コードが入ります。

DL/I 呼び出しの形式の例

DL/I CEETDLI インターフェース:

```
%INCLUDE CEEIBMAW;
CALL CEETDLI (function, i/o pcb, i/o area);
```

DL/I AIBTDLI インターフェース:

```
CALL AIBTDLI (parmcount, function, aib, i/o area);
```

DL/I 言語固有のインターフェース:

```
CALL PLITDLI (parmcount, function, i/o pcb, i/o area);
```

呼び出しと PCB タイプとの関係

次の表に、DL/I 呼び出しと、入出力プログラム連絡ブロック (PCB) および代替プログラム連絡ブロック (PCB) との関係を示します。

PCB は、使用する xxxTDLI インターフェースに応じて、呼び出しリストまたは AIB のパラメーターとして指定できます。

表 *65.* 呼び出しと *PCB* との関係 :

呼び出し	**I/O PCB**	代替 **PCB**
APSB [1]		
AUTH	X	
CHKP (基本)	X	
CHKP (シンボリック)	X	
CHNG [2]		X
CMD	X	
DPSB [1]		
GCMD	X	
GN	X	
GSCD	X	
GU	X	
INIT	X	
INQY	X	X
ISRT	X	X
LOG	X	
PURG	X	X
ROLB	X	

表 *65.* 呼び出しと *PCB* との関係 *(*続き*)*:

呼び出し	**I/O PCB**	代替 **PCB**
ROLS	X	
ROLL [1]		
SETO	X	X
SETS	X	
SETU	X	
SYNC	X	
XRST	X	

注**:**

1. この呼び出しは PCB とは関係ありません。
2. この呼び出しで使用する代替 PCB は変更可能でなければなりません。

I/O PCB マスクの指定

プログラムが入出力プログラム連絡ブロック (PCB) を使って呼び出しを発行すると、IMS は呼び出しの結果に関する情報を I/O PCB に返します。呼び出しの結果を判別するには、IMS が返す情報をプログラムでチェックする必要があります。

システム・サービス呼び出しを発行するには、I/O PCB が必要です。 I/O PCB はユーザーのプログラムの外部にあるため、ユーザーは自分のプログラム内に PCB のマスクを定義し、IMS 呼び出しの結果をチェックする必要があります。マスクには、I/O PCB と同じフィールドが同じ順序で入らなければなりません。このようにすると、プログラムで、PCB マスクを介して PCB 内のフィールドを参照することができます。

I/O PCB には、次の表にリストされているフィールドがあります。この表には、これらのフィールド、フィールドの長さ、および各フィールドの適用環境が示されています。

表 *66. I/O PCB* マスク

記述子	長さ **(**バイト数**)**	**DB/DC**	**DBCTL**	**DCCTL**	**DB** バッチ	**TM** バッチ
論理端末名 [1]	8	X		X		
IMS に予約済み [2]	2	X		X		
状況コード [3]	2	X	X	X	X	X
4 バイトの ローカル日付および 時間 [4]						
日付	4	X		X		
時間	4	X		X		
入力メッセージの順序番号 [5]	4	X		X		
メッセージ出力記述子名 [6]	8	X		X		
ユーザー ID [7]	8	X		X		

表 66. I/O PCB マスク (続き)

記述子	長さ (バイト数)	DB/DC	DBCTL	DCCTL	DB バッチ	TM バッチ
グループ名 [8]	8	X		X		
12 バイトのタイム・スタンプ [9]						
日付	4	X		X		
時間	6	X		X		
UTC オフセット	2	X		X		
ユーザー ID 標識[10]	1	X		X		
IMS に予約済み [2]	3					

注:

1. 論理端末名

   このフィールドには、メッセージを送信した端末の名前が入ります。ユーザーのプログラムが入力メッセージを受け取ると、IMS は、メッセージを送信した論理端末の名前をこのフィールドに入れます。この端末にメッセージを送り返すときは、ISRT 呼び出しを発行するときに I/O PCB を参照してください。IMS は、I/O PCB からの論理端末名を宛先とします。

2. **IMS** に予約済み

   これらのフィールドは予約済みです。

3. 状況コード

   IMS は、DL/I 呼び出しの結果を記述する状況コードを、このフィールドに入れます。プログラムが出す各 DL/I 呼び出しの後で、IMS は状況コードを更新します。DL/I 呼び出しを発行した後は、必ずプログラムで状況コードをテストしてください。

   状況コードは、次の 3 つのカテゴリーに分類されます。

   - 成功した状況コード、または例外だが有効な条件を伴う状況コード。このカテゴリーには、エラーは入りません。呼び出しが完全に成功した場合は、このフィールドにはブランクが入ります。このカテゴリーに属するコードのうちの多くは、情報のみを表すものです。例えば、QC 状況コードは、プログラムのメッセージ・キューにこれ以上メッセージが存在しないことを意味します。この状況コードを受け取ったときは、プログラムを終了してください。
   - プログラミング・エラー。このカテゴリーに属するエラーは、通常は訂正可能です。例えば、AD 状況コードは、無効な機能コードを示します。
   - 入出力エラーまたはシステム・エラー

   2 つ目と 3 つ目のカテゴリーでは、ユーザー・プログラムに、プログラムの終了前に出された最後の呼び出しに関する情報を印刷するエラー・ルーチンを指定する必要があります。ほとんどのインストール先では、そのインストール先のすべてのアプリケーション・プログラムで使用される標準のエラー・ルーチンが用意されています。

4. ローカル日時

   ローカルの現在日時は、非メッセージ・ドリブン BMP から発信されたものを除き、すべての入力メッセージの接頭部に置かれます。ローカル日付は、右寄せしたパック 10 進数で、yyddd の形式です。ローカル時刻はパック 10 進数の時刻であり、形式は hhmmsst です。ローカルの現在日時は、いつ IMS がメッセージ全体を受け取り、メッセージをプログラムに対する入力データとしてキューに入れたかを示すもので、アプリケーション・プログラムがメッセージを受け取った時刻ではありません。アプリケーションの処理時刻を入手するには、使用するプログラミング言語の時刻機構を使用する必要があります。

   会話の場合、プログラムから発信される入力メッセージの場合、または複数システム結合機能 (MSC) を使用して受け取られたメッセージの場合には、時刻および日付は、元のメッセージがいつ端末から受け取られたかを示します。

   注: I/O PCB のローカル日時と、オペレーティング・システムから戻された現在時刻を比較する際は注意してください。I/O PCB 日時は現在時刻と整合していない場合があります。この時刻は、以下の理由により現在時刻より進んでいることもあります。

   - I/O PCB のタイム・スタンプは IMS がメッセージを受信した地方時です。メッセージの受信後に地方時が変更された場合、現在時刻が I/O PCB 時刻より見かけ上早くなる可能性があります。これは、秋の時刻変更でクロックを 1 時間遅らせて設定した直後の 1 時間に起こる場合があります。
   - I/O PCB のタイム・スタンプはメッセージと共に保管された内部の IMS タイム・スタンプから派生しています。この内部タイム・スタンプは協定世界時 (UTC) にあり、メッセージがエンキューされた際に有効であった時間帯オフセットを含みます。この時間帯オフセットは、UTC 時間に追加され、I/O PCB に配置されている地方時が入手されます。ただし、保管されている時間帯オフセットは 15 分のみです。実際の時間帯オフセットが 15 分の整数倍でない場合、I/O PCB で戻された地方時は実際の時間と 7.5 分前後異なります。これにより I/O PCB 時刻が現在時刻より遅れることになり場合があります。詳細については、「*IMS V15* オペレーションおよびオートメーション」を参照してください。

   I/O PCB にあるローカルのタイム・スタンプの値に関する不安事項は、 IMS V6 で導入された拡張タイム・スタンプの使用により削減されます。システム管理者は拡張タイム・スタンプの形式を地方時にも UTC にもなるように選択できます。状態によっては、アプリケーションがオペレーティング・システムから UTC での時刻を要求し、その時刻と拡張タイム・スタンプの UTC フォームを比較すると有効な場合があります。これは、地方時の変更に際して、IMS UTC オフセットと z/OS UTC オフセットとの同期を保つ ETR がないインストール・システムで選択可能なオプションです。

5. 入力メッセージの順序番号

   入力メッセージ・シーケンス番号は、非メッセージ・ドリブンの BMP から発信されたものを除き、すべての入力メッセージの接頭部に置かれます。このフィールドには、IMS が入力メッセージに割り当てた順序番号が入ります。数値は 2 進数です。 IMS は、物理端末ごとに順序番号を割り当てます。この番号は、IMS の最後の始動時から連続しています。

6. メッセージ出力記述子名

   このフィールドを使用するのは、MFS を使用するときだけです。メッセージ出力記述子 (MOD) を伴う GU 呼び出しが出されると、IMS はこの区域にその名前を入れます。プログラムがエラーを検出した場合には、画面の形式を変更し、このフィールドを使用して端末にエラー・メッセージを送ることができます。これを行うためには、ISRT 呼び出しまたは PURG 呼び出しで MOD 名のパラメーターを指定して、プログラムで MOD 名を変更する必要があります。

   MFS は APPC をサポートしませんが、LU 6.2 プログラムでインターフェースを使用すると、MFS をエミュレートすることができます。例えば、アプリケーション・プログラムでエラー・メッセージの形式設定の方法を指定する場合、MOD 名を使用して IMS と通信することができます。

   関連資料: MOD 名と LTERM インターフェースの詳細については、「*IMS V15* コミュニケーションおよびコネクション」を参照してください。

7. ユーザー **ID**

   このフィールドは、RACF サインオン・セキュリティーとともに使用されます。システム内でサインオンがアクティブでない場合には、このフィールドにはブランクが入ります。

   サインオンがシステムでアクティブになっている場合、このフィールドには次のいずれかが入ります。

   - 発信元端末からのユーザーの識別名。
   - 発信元端末の LTERM 名 (その端末についてサインオンがアクティブでない場合)。
   - 許可 ID。バッチ指向 BMP の場合の許可 ID は、PROCLIB メンバー DFSDCxxx の中の BMPUSID= キーワードに指定されている値によって、次のように異なります。
     - BMPUSID=USERID が指定されている場合、JOB ステートメントの USER= キーワードの値が使用されます。
     - JOB ステートメントで USER= が指定されていない場合は、プログラムの PSB 名が使用されます。
     - BMPUSID=PSBNAME が指定されている場合、または BMPUSID= がまったく指定されていない場合は、プログラムの PSB 名が使用されます。 PSBNAME が RACF に対して定義されていない場合は、現行アドレス・スペースのユーザー ID が使用されます。これは、LSO=Y または PARDLI=1 が BMP に対して指定されている場合、ホーム従属領域のものか、制御領域のものになります。DFSBSEX0 が RC08 を返した場合も、現行アドレス・スペースのユーザー ID が使用されます。

       関連資料: 従属領域のリソース使用の許可の詳細については、「*IMS V15* システム管理」を参照してください。

8. グループ名

   SQL 呼び出しのセキュリティーのため DB2 により使用されるグループ名は、IMS トランザクションによって作成されます。

グループ名に適用される 3 つのインスタンスは次のとおりです。

- IMS システムで RACF およびサインオンを使用している場合、RACROUTE SAF (抽出) 呼び出しは 8 文字のグループ名を返します。
- IMS システムでセキュリティー・パッケージを使用する場合、RACROUTE SAF 呼び出しはそのパッケージから任意の 8 文字の名前を返し、それをグループ名として扱います。 RACROUTE SAF 呼び出しが 4 または 8 の戻りコードを返すと、グループ名は返されず、IMS はグループ名フィールドをすべてブランクにします。
- LU 6.2 を使用する場合、トランザクション・ヘッダーにはグループ名を含めることができます。

  関連資料**:** LU 6.2 の詳細については、「*IMS V15* コミュニケーションおよびコネクション」を参照してください。

9. **12** バイト・タイム・スタンプ

   このフィールドに現在日付と時刻が収められますが、IMS 内部パック 10 進数形式になっています。タイム・スタンプは、以下の各部分から構成されています。

   日付 `yyyydddf`

   このパック 10 進数日付は、年 (`yyyy`)、年間通算日 (`ddd`)、および有効なパック 10 進数の + 記号、例えば (`f`) から成ります。

   時間 `hhmmssthmiju`

   このパック 10 進数時刻は、時間、分、秒 (`hhmmss`)、およびマイクロ秒までの、秒の小数部分 (`thmiju`) から成ります。タイム・スタンプのこの部分には、パック 10 進数の符号は加えられていません。

   UTC オフセット `aqq$`

   パック 10 進数 UTC オフセットの接頭部は、4 ビットから成る属性 (`a`) です。(`a`) の 4 番目のビットが 0 であれば、タイム・スタンプは UTC、そうでなければタイム・スタンプは、地方時です。DFSPBxxx PROCLIB メンバーで指定されている制御領域パラメーター `TSR=(U/L)` が、地方時または UTC 時刻のいずれかでのタイム・スタンプの表現を制御します。

   オフセット値 (`qq$`) は、地方時または UTC 時刻に変換するためにそれぞれ UTC 時刻または地方時に加えるべき、4 分の 1 時間単位のオフセットです。

   オフセットの符号 (`$`) は、パック 10 進数の正負の符号規則に従います。

   I/O PCB マスクのフィールド 4 は、常にローカルの日時を持っています。フィールド 4 の説明については、前の表の注を参照してください。

   関連資料**:** 内部パック 10 進数時刻形式の詳細については、「*IMS V15* システム・ユーティリティー」を参照してください。

10. ユーザー **ID** 標識

ユーザー ID 標識は、I/O PCB 内および INQY 呼び出しへの応答に入れられます。ユーザー ID 標識には、以下のいずれかが入ります。

- U - サインオン時にソース端末から得られたユーザー ID
- L - 送信元端末の LTERM 名 (サインオンがアクティブでない場合)
- P - 送信元 BMP またはトランザクションの PSBNAME
- O - その他の名前

ユーザー ID 標識フィールド内の値は、ユーザー ID フィールドの内容を示します。

関連概念:

476 ページの『メッセージの結果: I/O PCB』

代替 PCB マスクの指定

代替プログラム連絡ブロック (PCB) のマスクには、3 つのフィールドが入っています。

次の表は、これらのフィールド、フィールド長、およびそのフィールドが適用される環境を説明しています。

表 *67.* 代替 *PCB* マスク

記述子	長さ(バイト数)	DB/DC	DBCTL	DCCTL	DB バッチ	TM バッチ
論理端末名 [1]	8 バイト	X		X		
IMS に予約済み [2]	2 バイト	X		X		
状況コード [3]	2 バイト	X		X		

注:

1. 論理端末名

   このフィールドには、メッセージを送る先の論理端末の名前、LU 6.2 記述子、またはトランザクション・コードが入ります。

   関連資料: LU 6.2 の詳細については、「*IMS V15* コミュニケーションおよびコネクション」を参照してください。

2. **IMS** に予約済み

   この 2 バイトのフィールドは予約済みです。

3. 状況コード

   このフィールドには、この PCB を最近使用した呼び出しの結果を記述している 2 バイトの状況コードが入ります。

関連概念:

488 ページの『他の端末およびプログラムへのメッセージの送信』

AIB マスクの指定

アプリケーションでプログラム連絡ブロック (PCB) アドレスを指定していない場合、または呼び出し機能で PCB を使用しない場合に、プログラムは IMS との通信に AIB を使用します。

使用可能であれば、アプリケーション・プログラムは、返された PCB アドレスを使用して PCB 内の状況コードを検査したり、そのアプリケーション・プログラムに必要なその他の情報を入手したりできます。 AIB マスクを使用すると、プログラムで制御ブロック定義を解釈することができます。 AIB の構造は、次の表に示すとおり、フィールドの順序およびバイト長に従って、作業用ストレージにフルワード境界で定義され、初期設定される必要があります。表の注は、各フィールドの内容を説明しています。

表 *68. AIB* フィールド

記述子	長さ (バイト数)	DB/DC	DBCTL	DCCTL	DB バッチ	TM バッチ
AIB ID [1]	8	X	X	X	X	X
DFSAIB 割り振り長さ [2]	4	X	X	X	X	X
副次機能コード [3]	8	X	X	X	X	X
リソース名 1[4]	8	X	X	X	X	X
予約済み 1 [5]	8					
リソース名 2 [6]	8					
出力域の最大長 [7]	4	X	X	X	X	X
使用される出力域の長さ [8]	4	X	X	X	X	X
AIBRSFLD [9]	4					
予約済み 2 [10]	8					
戻りコード [11]	4	X	X	X	X	X
理由コード [12]	4	X	X	X	X	X
エラー・コード拡張子 [13]	4	X		X		
リソース・アドレス [14]	4	X	X	X	X	X
予約済み 3 [15]	40					

注:

1. **AIB ID (AIBID)**

   この 8 バイトのフィールドには、AIB ID が入ります。DL/I 呼び出しを出す前に、アプリケーション・プログラムで AIBID を DFSAIBbb に初期設定しておく必要があります。このフィールドは必須です。呼び出しが完了したとき、このフィールドに返された情報は変更されていません。

2. **DFSAIB** 割り振り長さ **(AIBLEN)**

   このフィールドには、プログラムで定義された AIB の実際の長さが 4 バイトで入ります。DL/I 呼び出しを発行する前に、アプリケーション・プログラム

で AIBLEN を初期設定しなければなりません。最小でも 128 バイトの長さが必要です。呼び出しが完了したとき、このフィールドに返された情報は変更されていません。このフィールドは必須です。

3. 副次機能コード **(AIBSFUNC)**

   この 8 バイトのフィールドには、副次機能を使用する呼び出しのための副次機能コードが入ります。 DL/I 呼び出しを発行する前に、アプリケーション・プログラムで AIBSFUNC を初期設定しなければなりません。呼び出しが完了したとき、このフィールドに返された情報は変更されていません。

4. リソース名 **(AIBRSNM1)**

   この 8 バイトのフィールドには、リソースの名前が入ります。リソースは呼び出しによって変わります。DL/I 呼び出しを発行する前に、アプリケーション・プログラムで AIBRSNM1 を初期設定しなければなりません。呼び出しが完了したとき、このフィールドに返された情報は変更されていません。このフィールドは必須です。

   呼び出しリストで PCB アドレスを渡す代りに PCB 名を渡すために AIB が使用される PCB 関連の呼び出しの場合、このフィールドには PCB 名が入ります。I/O PCB の PCB 名は IOPCBbb です。その他のタイプの PCB の PCB 名は、PSBGEN 内の PCBNAME= パラメーターで定義されています。

5. 予約済み **1**

   この 16 バイト・フィールドは予約済みです。

6. リソース名 **2**

   この 8 バイト・フィールドは予約されています。

7. 出力域の最大長 **(AIBOALEN)**

   この 4 バイトのフィールドには、呼び出しリストに指定された出力域の長さがバイト単位で入ります。出力域にデータを返すすべての呼び出しについて、アプリケーション・プログラムで AIBOALEN を初期設定しなければなりません。呼び出しが完了したとき、ここに返された情報は変更されていません。

8. 使用された出力域の長さ **(AIBOAUSE)**

   この 4 バイト・フィールドには、出力域にデータを返すすべての呼び出しで、IMS により返されるデータの長さが入ります。呼び出しが完了したとき、このフィールドには、この呼び出しに使用された入出力域の長さが入ります。

9. 予約済み **2**

   この 8 バイト・フィールドは予約されています。

   最初の 4 バイトは ICAL 呼び出しによって使用され、同期呼び出し処理が完了するまで待機する時間を指定します (AIBRSFLD)。

10. 戻りコード **(AIBRETRN)**

    呼び出しが完了すると、この 4 バイト・フィールドに戻りコードが入ります。

11. **AIBRSFLD**

この 4 バイト・フィールドには、リソース情報が含まれます。このフィールドの使用法は、呼び出しによって異なります。

12. 理由コード **(AIBREASN)**

    呼び出しが完了すると、この 4 バイト・フィールドには、理由コードが入ります。

13. エラー・コード拡張 **(AIBERRXT)**

    この 4 バイト・フィールドには、AIBRETRN の戻りコードおよび AIBREASN の理由コードに基づく、追加のエラー情報が入ります。

14. リソース・アドレス **(AIBRSA1)**

    呼び出しが完了すると、この 4 バイト・フィールドには呼び出し固有の情報が入ります。呼び出しリストで PCB アドレスを渡す代りに、AIB を使用して PCB 名を渡す PCB 関連の呼び出しの場合、このフィールドは PCB アドレスを返します。

15. 予約済み **3**

    この 40 バイト・フィールドは予約されています。

入出力域の指定

入出力域は、アプリケーション・プログラムと IMS との間でセグメントの受け渡しを行うために使用します。

入出力域には、呼び出しのタイプによって以下のものが入ります。

- セグメントをリトリーブすると、IMS は要求されたセグメントを入出力域に入れます。
- 新しいセグメントを追加するときには、ユーザーは最初に、新しいセグメントを入出力域内で作成します。
- セグメントを修正するときには、ユーザーのプログラムは、まず最初にそのセグメントをリトリーブする必要があります。セグメントをリトリーブすると、IMS はセグメントを入出力域に入れます。

プログラムと IMS との間で渡されるレコード・セグメントの形式は、固定長でも可変長でもかまいません。その際にアプリケーション・プログラムにとって重要な相違点は 1 つだけです。すなわち、セグメントのデータ域の先頭にあるメッセージ・セグメントには 2 バイト (PLITDLI インターフェースの場合は 4 バイト) の長さのフィールドが含まれています。

IMS 呼び出しのための入出力域は、ユーザーのプログラムが IMS からリトリーブしたり、IMS に送信したりする最大のセグメントを収められる大きさでなければなりません。

ユーザーのプログラムが、D コマンド・コードを使用する Get または ISRT 呼び出しを出す場合、入出力域は、プログラムがリトリーブまたは挿入するセグメントの最大パスを収められる大きさでなければなりません。

AIBTDLI インターフェース

AIBTDLI は、アプリケーション・プログラムと IMS との間のインターフェースとして使用します。

制約事項**:** AIB 内のフィールドは、IMS で定義されている場合を除き、アプリケーション・プログラムで使用することはできません。

AIBTDLI インターフェースを使用するときには、AIB のリソース名フィールドに (PSBGEN で定義した) PCB 名を入れることにより、呼び出しに必要なプログラム連絡ブロック (PCB) を指定します。PCB アドレスは指定しません。 AIB には PCB 名が入るので、アプリケーション・プログラムは、PCB アドレスではなく PCB 名を参照します。アプリケーション・プログラムは、PCB リスト内の PCB の相対的な位置を知る必要はありません。呼び出しが完了すると、AIB は、アプリケーション・プログラムが渡した PCB 名に対応する PCB アドレスを返します。

DB PCB および代替 PCB の名前は、PSBGEN の間にユーザーが定義します。入出力 PCB はすべて、PCB 名 bbb で生成されます。生成済みプログラム仕様ブロック (GPSB) の場合は、IOPCBbbb という PCB 名の入出力 PCB が生成され、また TPPCB1bb という PCB 名の修正可能な代替 PCB が生成されます。

PCB 名を渡すことができるということは、PCB リスト内の相対的な PCB 番号を知る必要がないことを意味します。さらに、AIBTDLI インターフェースにより、アプリケーション・プログラムは、PCB リストにない PCB で呼び出しを行うことができます。LIST= キーワードは、PSBGEN の間に PCB マクロで定義され、その PCB が PCB リストに含まれるかどうかを制御します。

AIB は、AIBTDLI インターフェースを使用する DL/I 呼び出しのために IMS に渡されるユーザー定義のストレージ内にあります。呼び出しが完了したとき、AIB は IMS により更新されます。AIB に割り振るストレージは、128 バイト以上にする必要があります。

関連概念:

352 ページの『GSAM データベース用の PCB マスク』

関連資料:

449 ページの『PL/I のアプリケーション・プログラミング』

446 ページの『Pascal のアプリケーション・プログラミング』

440 ページの『C 言語によるアプリケーション・プログラミング』

437 ページの『アセンブラー言語によるアプリケーション・プログラミング』

241 ページの『アセンブラー言語によるアプリケーション・プログラミング』

言語固有のエントリー・ポイントの指定

IMS は、エントリー・ポイントを介してアプリケーション・プログラムに制御権を渡します。アセンブラー言語、C 言語、COBOL、Pascal、および PL/I で入り口ステートメントをコーディングするための適切な形式を使用します。

ユーザーのエントリー・ポイントは、PSB に定義されている順番で、プログラム連絡ブロック (PCB) を参照する必要があります。

IMS が PCB ポインターを PL/I のプログラムに渡す形式は、アセンブラー言語、C 言語、COBOL、Java、または Pascal のプログラムに渡す形式とは異なります。さらに、Pascal の場合は、PCB ポインターを渡す前に、IMS が整数を渡す必要があります。 IMS は、PSBGEN の LANG キーワードまたは PSBGEN ステートメントを使用して、制御権を渡すプログラムのタイプを判別します。したがって、PSBGEN の間に指定した言語は、プログラムの言語と矛盾しないようにしなければなりません。

AIB 構造 (AIBTDLI または CEETDLI) を使用するアプリケーション・インターフェースは、PCB 構造ではなく PCB 名を使用するので、入り口でアプリケーション・プログラムに PCB リストを渡す必要はありません。

各 DL/I 呼び出しをコーディングするときには、呼び出しに使用する PCB を用意しなければなりません。すべての IMS TM アプリケーション・プログラムの場合、プログラムがアクセスできる PCB のリストは、エントリー・ポイントでプログラムに渡されます。

アセンブラー言語

アセンブラー言語の DL/I プログラムへの入り口ステートメントには、任意の名前を使用できます。 IMS がアプリケーション・プログラムに制御を渡すときには、レジスター 1 には、可変長フルワード・パラメーター・リストのアドレスが入っています。リスト内の各ワードには、PCB のアドレスが入っています。レジスター 1 の内容を上書きする前に、パラメーター・リスト・アドレスを保管しておきます。IMS は、リスト内の最後のフルワードの高位バイトを X'80' に設定し、リストの終わりを示します。順方向および逆方向のチェーニングには、標準の z/OS リンケージ規則を使用してください。

C 言語

IMS がプログラムに制御権を渡すとき、ポインターの形式で、プログラムが使用するそれぞれの PCB のアドレスを渡します。通常、argc 引数および argv 引数は、IMS で呼び出されるプログラムには使用できません。IMS パラメーター・リストは、__pcblist マクロを使用してアクセスできます。PCB は、__pcblist[0]、__pcblist[1] により直接参照することができます。または、より分かりやすい名前を付与するマクロを定義することもできます。以下のように適切なタイプを得るために、I/O PCB をキャストしなければなりません。

```
(IO_PCB_TYPE *)(__pcblist[0])
```

C 言語プログラムの入り口ステートメントは main ステートメントです。

```
#pragma runopts(env(IMS),plist(IMS))
#include <ims.h>

main()
{
.
.
.
}
```

env オプションは、C 言語プログラムが実行される操作環境を指定します。例えば、C 言語プログラムが IMS のもとで呼び出され、IMS 機能を使用する場合には、env(IMS) を指定します。 plist オプションは、C 言語プログラムが呼び出されたときに C 言語プログラムが受け取る呼び出しパラメーターの形式を指定します。

IMS などのシステム・サポート・サービス・プログラムでプログラムを呼び出すときには、主プログラムに渡されるパラメーターの形式は、C 言語の形式である argv、argc、および envp に変換しなければなりません。この変換を行うには、C 言語プログラムが受け取るパラメーター・リストの形式を指定する必要があります。 ims.h インクルード・ファイルには、PCB マスクの宣言が入ります。

以下の 3 つの方法で、プログラム実行を終了することができます。

- 明示的な return ステートメントなしでメイン・プロシージャーを終了する。
- main から return ステートメントを実行する。
- どこからでも exit または abort 呼び出しを実行するか、あるいは longjmp を main に出して、正常な戻りを行う。

C 言語プログラムでは、system 関数を使用して別のプログラムに制御権を渡すことができます。パラメーターの受け渡しについては通常の規則が適用されます。例えば、system 関数を使用するときには、argc 引数および argv 引数を使用して情報を渡すことができます。呼び出されたプログラムは、初期 __pcblist を使用できるようになります。

COBOL

プロシージャー・ステートメントは、必ず I/O PCB を最初に、使用する代替 PCB を次に、使用する DB PCB を最後に参照することになります。代替 PCB および DB PCB は、PSB に定義されている順番でリストされていなければなりません。

```
Procedure division using the PCB-NAME-1 [,...,PCB-NAME-N]
```

IMS の旧バージョンでは、入り口ステートメントに `using` キーワードをコーディングして PCB を参照できました。しかし、IMS では入り口ステートメント上のそのようなコーディングを引き続き受け入れます。

推奨事項: PCB を参照するには、入り口ステートメントではなくプロシージャー・ステートメントを使用してください。

Pascal

エントリー・ポイントは、REENTRANT プロシージャーとして宣言しなければなりません。IMSが Pascal プロシージャーに制御を渡すと、パラメーター・リスト内の最初のアドレスは Pascal で使用するために予約され、その他のアドレスはプログラムで使用する PCB となります。 PCB タイプは、この入り口ステートメントの前に定義しなければなりません。IMS インターフェース・ルーチン PASTDLI は、GENERIC 指令で宣言しなければなりません。

```
procedure ANYNAME(var SAVE: INTEGER;
                  var pcb1-name: pcb1-name-type[;
                  ...
                  var pcbn-name: pcbn-name-type]); REENTRANT;
procedure ANYNAME;
(* Any local declarations *)
  procedure PASTDLI; GENERIC;
begin
  (* Code for ANYNAME *)
end;
```

PL/I

入り口ステートメントには任意の有効な PL/I 名を付けることができ、プログラム内の最初の実行可能ステートメントでなければなりません。IMS がユーザーのプログラムに制御を渡すと、ユーザー・プログラムが使用する各 PCB のアドレスがポインターの形で渡されます。入り口ステートメントをコーディングするときには、必ず PCB 名ではなく、PCB を指すポインターとしてこのステートメントのパラメーターをコーディングします。

```
anyname: PROCEDURE (pcb1_ptr [,..., pcbn_ptr]) OPTIONS (MAIN);
:
RETURN;
```

CCETDLI および AIBTDLI インターフェースの考慮事項

CCETDLI の考慮事項は以下のとおりです。

- PL/I プログラムでは、CEETDLI のエントリー・ポイントは、CEEIBMAW 組み込みファイルに定義されます。あるいは、ユーザーが自分で宣言することもできます。宣言する場合は、アセンブラー言語の入り口 (DCL CEETDLI OPTIONS(ASM);) として宣言する必要があります。
- C 言語のアプリケーションの場合には、env (IMS) および plist (IMS) を指定して、アプリケーションが引数の PCB リストを受け入れることができるようにします。CEETDLI 関数は、 <leawi.h> の中で定義されます。また、CTDLI 関数は、<ims.h> の中で定義されます。

AIBTDLI の考慮事項は以下のとおりです。

- C/MVS、COBOL、または PL/I のいずれかの言語のアプリケーションで AIBTDLI インターフェースを使用するときは、異常終了の代行受信を抑制する言語の実行時オプション (NOSPIE および NOSTAE) を指定する必要があります。ただし、言語環境プログラム準拠のアプリケーションの場合、NOSPIE および NOSTAE 制限は除去されます。
- PL/I プログラムの場合には、AIBTDLI エントリー・ポイントは、アセンブラー言語の入り口 (DCL AIBTDLI OPTIONS (ASM) ;) として宣言しなければなりません。
- C 言語のアプリケーションの場合には、env (IMS) および plist (IMS) を指定して、アプリケーションが引数の PCB リストを受け入れることができるようにします。

プログラム連絡ブロック (PCB) リスト

アプリケーション・プログラムでは、正しい形式のプログラム連絡ブロック (PCB) リストおよび生成されたプログラム仕様ブロック (GPSB) PCB リストを使用します。

PCB リストの形式

以下は、PCB の形式です。

```
[IOPCB]
[Alternate PCB ... Alternate PCB]
[DB PCB ... DB PCB]
[GSAM PCB ... GSAM PCB]
```

各 PSB には、少なくとも 1 つの PCB がなければなりません。トランザクション管理呼び出しには、I/O PCB または代替 PCB が必要で、ほとんどのシステム・サービス呼び出しには I/O PCB が必要です。DL/I データベースの DB PCB は、IMS Database Manager (IMS DB) でしか使用されませんが、プログラムが DCCTL または TM バッチのもとで実行される場合でも指定することはできます。DB PCB は、全機能 PCB、DEDB PCB、または MSDB PCB のいずれであってもかまいません。 GSAM PCB は、DCCTL または TM バッチで使用することができます。

GPSB PCB リストの形式

生成されたプログラム仕様ブロック (GPSB) は、以下の形式になります。

```
[IOPCB]
[Alternate PCB]
```

GPSB には、I/O PCB と変更可能代替 PCB がそれぞれ 1 つだけ入ります。これは、すべてのトランザクション管理アプリケーション・プログラムで使用可能で、これにより、指定した PCB へのアクセスが可能になります。その際、PSBGEN は必要ありません。

GPSB 内の PCB には、PCB 名が事前定義されています。I/O PCB の PCB 名は、IOPCBbb です。代替 PCB の 名前は TPPCB1bb です。

PCB の要約

I/O PCB と代替 PCB は、さまざまなタイプのアプリケーション・プログラムで使用できます。

TM バッチ・プログラム
: 代替 PCB は、IMS TM によりプログラムに提供される PCB のリストに常に含まれています。I/O PCB は、CMPAT オプションが PSBGEN に指定されているかどうかにかかわらず、PCB リスト内に常に存在します。

BMP、**MPP**、および **IFP**
: I/O PCB は、CMPAT オプションが PSB に指定されているかどうかにかかわらず、常に、PCB リスト内に存在し、リスト内の最初のアドレスになります。 PCB リストには、まず I/O PCB のアドレス、続いて任意の代替 PCB のアドレス、そして最後に DB PCB のアドレスが入ります。

言語環境プログラム

IBM 言語環境プログラムは、1 つ以上の高水準言語で書かれたユーザーのアプリケーション・プログラムを実行するための戦略的な実行環境を提供します。

この言語環境プログラムは、言語固有の実行時サポートだけでなく、初期設定、終了、メッセージ処理、条件処理、ストレージ管理のサポートおよび各国語サポートなどのアプリケーションに対する実行時サービスを言語間にわたって提供します。言語環境プログラムのサービスの多くは、各種プログラミング言語間で共通の一連の言語環境プログラム・インターフェースにより、明示的にアクセスできます。つまり、これらのサービスは、言語環境プログラムに準拠したすべてのプログラムからアクセスできます。

言語環境プログラムに準拠したプログラムは、次のコンパイラーを使ってコンパイルすることができます。

- IBM C++/MVS
- IBM COBOL
- IBM PL/I

デフォルトでは、言語環境プログラム・インフラストラクチャーは 31 ビット・アドレッシング・モードを使用します。JVM=64 を指定すると、言語環境プログラムは 64 ビット・アドレッシング・モードを使用するよう変更されます。

言語環境プログラムは、C、C++、およびアセンブリー言語のインターオペラビリティーを 64 ビット・アドレッシング・モードでサポートしますが、COBOL および PL/I のインターオペラビリティーを 64 ビット・アドレッシング・モードではサポートしません。Java アプリケーションが COBOL または PL/I のいずれかを呼び出す場合は、JVM=64 に切り替えないでください。JVM=64 を使用するようリージョンを不注意に切り替えてしまった場合、互換性のない相互運用可能なアプリケーションが実行されていると、アプリケーションはシステム異常終了やユーザー異常終了を受け取る可能性があります。

IMS への CEETDLI インターフェース

IMS への言語非依存型の CEETDLI インターフェースは、言語環境プログラムによって提供されます。これは、言語環境プログラムが提供する拡張エラー処理機能をサポートする唯一の IMS インターフェースです。CEETDLI インターフェースは、他の IMS アプリケーション・インターフェースと同じ機能をサポートします。特性は次のとおりです。

- parmcount 変数は任意指定。
- 長さフィールドは 2 バイト長。
- 直接ポインターを使用。

関連資料: 言語環境プログラムの詳細については、「*z/OS* 言語環境プログラム プログラミング・ガイド」を参照してください。

PL/I の互換性のための PSBGEN の LANG= オプション

PLICALLA エントリー・ポイントを使用する互換モードで実行する IMS PL/I アプリケーションの場合には、PSBGEN に LANG=PLI を指定しなければなりません。その他のオプションでは、エントリー・ポイントを変更し、SYSTEM (IMS) をコンパイル段階の EXEC PARM に追加することができるので、PSBGEN で LANG=ブランクまたは LANG=PLI を指定することができます。以下の表では、LANG=ブランクおよび LANG=PLI が使用可能である状況を要約しています。

表 *69.* 言語環境プログラムにおける *PL/I* との互換性のための *LANG=* オプションの使用

コンパイル **EXEC** ステートメントが **PARM=(...,SYSTEM(IMS)...**	エントリー・ポイント名が **PLICALLA**	有効な **LANG=** 値
あり	あり	LANG=PLI
あり	なし	LANG=ブランクまたは LANG=PLI

表 69. 言語環境プログラムにおける *PL/I* との互換性のための *LANG=* オプションの使用 (続き)

コンパイル **EXEC** ステートメントが **PARM=(...,SYSTEM(IMS)...**	エントリー・ポイント名が **PLICALLA**	有効な **LANG=** 値
なし	なし	注: IMS PL/I アプリケーションには無効
なし	あり	LANG=PLI

PLICALLA は、言語環境プログラムと PL/I との互換性を確保するためだけに有効なエントリー・ポイントです。バインド時に PLICALLA 入り口を使用する PL/I アプリケーションを、PLICALLA 入り口を持つ言語環境プログラムを用いてバインドすると、そのバインドは成功します。ただし、PSB で LANG=PLI を指定する必要があります。アプリケーションを、PL/I for z/OS & VM バージョン 1 リリース 1 またはそれ以降を使用して再コンパイルした後、言語環境プログラムのバージョン 1 リリース 2 またはそれ以降でバインドすると、バインドは失敗します。PLICALLA 入り口ステートメントをバインドから外さなければなりません。

IMS TM プログラミングでの DL/I の特殊な考慮事項

IMS Transaction Manager のアプリケーション・プログラミングを行う際の特殊な考慮事項としては、混合言語プログラミング、z/OS の拡張アドレッシング機能の使用法、プリロードされたプログラムのための COBOL コンパイラー・オプション、および DCCTL 環境に関する考慮事項があります。

混合言語プログラミング

アプリケーション・プログラムが言語環境プログラムの言語非依存インターフェース CEETDLI を使用する場合、IMS は呼び出し側プログラムの言語を意識する必要はありません。

アプリケーション・プログラムが言語依存のインターフェースで IMS を呼び出すとき、IMS は呼び出し側プログラムの言語を CALL ステートメントに指定された入り口名によって判断します。

- CALL CBLTDLI は、プログラムが COBOL であることを示す。
- CALL PLITDLI は、プログラムが PL/I であることを示す。
- CALL PASTDLI は、プログラムが Pascal であることを示す。
- ctdli (...) は、プログラムが C 言語であることを示す。
- CALL ASMTDLI は、プログラムがアセンブラー言語であることを示す。

PL/I プログラムでアセンブラー言語サブルーチンを呼び出し、そのアセンブラー言語サブルーチンで CALL ASMTDLI を使用して DL/I 呼び出しを行う場合、アセンブラー言語サブルーチンでは PL/I の規則ではなく、アセンブラー言語呼び出し規則に従っていなければなりません。

入出力域に LLZZ という形式を使用するこの状態では、LL はハーフワードであり、PLITDLI に使用されるフルワードではありません。

言語環境プログラムのルーチン保存の使用

言語環境プログラムを必要とする IMS TM の従属領域 (IMS メッセージ処理領域など) でプログラムを実行する場合、のライブラリー・ルーチンの保存と IMS TM の既存の PREINIT 機能を同時に使用すると、パフォーマンスを向上させることができます。

関連資料: 言語環境プログラムの詳細については、「*z/OS* 言語環境プログラム プログラミング・ガイド」を参照してください。

z/OS の拡張アドレッシング機能の使用

z/OS の拡張アドレッシング機能の 2 つのモードは、アドレッシング・モード (AMODE) と常駐モード (RMODE) です。

IMS では、アプリケーション・プログラムの RMODE および AMODE には制約は課されません。プログラムは、拡張仮想記憶域内に常駐させることができます。呼び出しの際に参照されるパラメーターも、拡張仮想記憶域内に入れることができます。

関連資料: 言語環境プログラムの詳細については、「*z/OS MVS* プログラミング: アセンブラー・サービス ガイド」を参照してください。

プリロードされたプログラムのための COBOL コンパイラー・オプション

COBOL プログラムを COBOL for z/OS & VM コンパイラーでコンパイルし、プリロードする場合には、COBOL コンパイラー・オプション RENT を使用しなければなりません。COBOL プログラムを VS COBOL II コンパイラーでコンパイルしてプリロードする場合は、COBOL コンパイラー・オプション RES および RENT を使用する必要があります。

DCCTL

DCCTL 環境では、アプリケーションが参照できるのは、I/O PCB、代替 PCB、または GSAM PCB だけです。アプリケーション・プログラムではデータベースを参照する PCB を含む PSB を使用できますが、これらの PCB は処理中に使用することはできません。 COBOL、PL/I、C、および Pascal の入り口ステートメントは、処理できない可能性のある PCB も含め、PSB 内のすべての PCB を参照しなければなりません。PCB は PSB 内にリストされている順序で組み込まれていなければならないためです。これには最後に参照された PCB より前のすべての PCB が含まれ、DB PCB を含むことがあります。GSAM PCB を使用した場合には、その前のすべての PCB を参照しなければなりません。

第 25 章 IMS TM によるメッセージ処理

IMS Transaction Manager アプリケーション・プログラムは、アセンブラー言語、C 言語、COBOL、Pascal、および PL/I で作成して、メッセージを処理することができます。

ユーザーのプログラムによるメッセージの処理

メッセージのリトリーブおよび送信を行うには、IMS TM アプリケーション・プログラムで IMS TM に呼び出しを発行します。 プログラムで呼び出しを発行してメッセージをリトリーブする際、IMS TM はその呼び出しの中で指定された入出力域に入力メッセージを入れます。呼び出しを発行してメッセージを送信する前に、プログラムの入出力域内に出力メッセージを作成しなければなりません。

メッセージのタイプ

端末オペレーターが IMS TM に対して送信できるメッセージは 4 種類あります。

IMS TM メッセージの宛先から、送信されるメッセージの種類が識別されます。

- 別の端末。先頭の 8 バイトの論理端末名は、これが別の端末を宛先とするメッセージ通信であることを意味します。別の論理端末にメッセージを送信する論理端末ユーザーの場合は、受信する論理端末の名前を入力し、その後にメッセージを続けます。IMS TM 制御領域は、指定された論理端末へメッセージを経路指定します。この種のメッセージの結果、メッセージ処理プログラム (MPP) でのアクティビティーが何かスケジュールされるということはありません。
- アプリケーション・プログラム。先頭からの 8 バイトのトランザクション・コードは、メッセージの宛先がアプリケーション・プログラムであることを意味します。IMS TM は、トランザクション・コードを使用して、MPP およびトランザクション指向バッチ・メッセージ処理プログラム (BMP) を識別します。特定のアプリケーション・プログラムを使用して要求を処理する場合は、そのアプリケーション・プログラムのトランザクション・コードを入力します。
- **IMS TM**。先頭のバイトの「/」(スラッシュ) は、そのメッセージが IMS TM 宛てのコマンドであることを意味します。
- メッセージ通信サービス。システム・サービスの DFSAPPC 要求は、メッセージ通信サービスに送られます。

アプリケーション・プログラムが送信できるのは、次の 3 種類のメッセージです。

- コマンド。メッセージ・テキストの先頭のバイトの「/」(スラッシュ) は、そのメッセージが IMS TM 宛てのコマンドであることを意味します。プログラマーは、通常は端末オペレーターが実行するタスクをプログラムで実行したいときにコマンドを出すようにアプリケーションを設計します。これは自動化操作プログラム・インターフェース (AOI) と呼ばれるもので、「*IMS V15* コミュニケーションおよびコネクション」および「*IMS V15* オペレーションおよびオートメーション」で説明されています。

コマンドは、CMD 呼び出しを使用して出します。ISRT で作成したメッセージには、コマンドとして指定しないものでも先頭バイトにスラッシュが入るため、ISRT 呼び出しはコマンドを発行する目的では使用しないでください。

- 論理端末名を指定することによる論理端末へのメッセージ。
- トランザクション・コードを使用するプログラム間通信。

プログラムが送受信するメッセージは、複数のセグメントで構成されています。 GU 呼び出しを使用して新しいメッセージの最初のセグメントをリトリーブし、GN 呼び出しを使用してそのメッセージの残りのセグメントをリトリーブします。以下の図は、3 つのメッセージを示しています。メッセージ A には 1 つのセグメントが、メッセージ B には 2 つのセグメントが、メッセージ C には 3 つのセグメントがそれぞれあります。

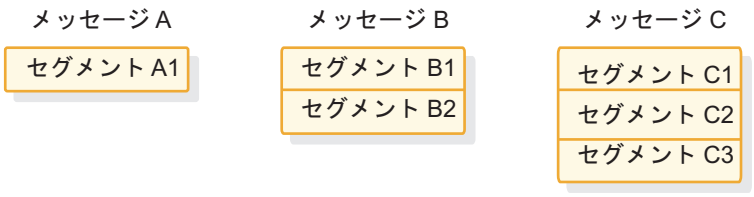


図 73. メッセージ・セグメント

メッセージ A をリトリーブするには、GU 呼び出しを発行するだけです。メッセージ B および C をリトリーブするには、GU 呼び出しを 1 回発行して最初のセグメントをリトリーブし、次いで残りの各セグメントごとに GN 呼び出しを発行してリトリーブします。これは、各メッセージにいくつセグメントが入っているかをユーザーが知っていることを前提としています。メッセージの数が分からない場合は、IMS TM がそのメッセージのすべてのセグメントがリトリーブされたことを示す QD 状況コードを返すまで、GN 呼び出しを発行してください。

複数セグメント・メッセージの最初のセグメントをリトリーブした後で、不注意から GU 呼び出しを発行した場合、IMS TM は QC 状況コードを返します。この状況はこれ以上メッセージが存在しないことを示すもので、プログラムはメッセージに関連した追加のセグメントをリトリーブすることはありません。データは失われることになり、脱落が起こったことは何も示されません。

入力メッセージの形式と内容

アプリケーション・プログラムが端末または別のプログラムから受信する入力メッセージには、ほとんどの場合に長さフィールド、ZZ フィールド、トランザクション・コード・フィールド、およびテキスト・フィールドが含まれています。

ただし、入力メッセージが変更 (CHNG) 呼び出しの結果であり、その呼び出しが、変更可能代替 PCB の宛先をユーザー指定の論理端末、LU 6.2 記述子、またはトランザクション・コードに設定する場合、トランザクション・コード・フィールドは存在しなくてもかまいません。

次に示す各表は、メッセージ入力のレイアウトを示しています。入力メッセージのフィールド名は、それぞれの表の第 1 行に示されています。各フィールド名の下の数値は、そのフィールドに定義された長さをバイト数で表しています。次の表は、AIBTDLI、ASMTDLI、CBLTDLI、CEETDLI、CTDLI、および PASTDLI の各イン

ターフェースで使用する入力メッセージの形式を示しています。 PLITDLI インターフェースのメッセージは、いくらか異なっています。

表 *70.* 入力メッセージ形式

フィールド名	フィールド長
LL	2
ZZ	2
TRANCODE	8
テキスト	可変

表 *71. PLITDLI* インターフェースの場合の入力メッセージの形式

フィールド名	フィールド長
LLLL	4
ZZ	2
TRANCODE	8
テキスト	可変

入力メッセージ・フィールドの内容は以下のとおりです。

LL または LLLL
長さフィールドには、LL (または LLLL) および ZZ を含む、入力メッセージ・セグメントの長さが 2 進数で入ります。入力メッセージのリトリーブ時、長さフィールドに、IMS TM がこの数値を入れます。

AIBTDLI、ASMTDLI、CBLTDLI、CEETDLI、CTDLI、および PASTDLI の各インターフェースの場合は、LL フィールドを 2 バイト長として定義します。

PLITDLI インターフェースの場合は、LLLL フィールド 4 バイト長として定義します。LLLL フィールドの値は、入力メッセージの長さから 2 バイトを引いたものです。例えば、テキストの長さが 12 バイトであれば、フルワード LLLL には値 24 バイトが入ります。この値は、LLLL (4 バイト) + ZZ (2 バイト) + TRANCODE (8 バイト) + テキスト (12 バイト) - 2 バイトの合計です。

ZZ ZZ フィールドは 2 バイト・フィールドで、IMS TM 用に予約されています。ユーザー・プログラムでこのフィールドを修正することはありません。

TRANCODE
TRANCODE は、着信メッセージのトランザクション・コードです。

テキスト
このフィールドには、端末からアプリケーション・プログラムに送信されるメッセージ・テキストが入ります。メッセージの最初のセグメントには、メッセージのテキスト部分の先頭部分プログラムに関連するトランザクション・コードが入ります。入力メッセージにトランザクション・コードを含める必要はありませんが、整合性を保つために含めることもできます。

入力メッセージ内のテキスト・フィールドの内容およびプログラムがメッセージを受信する際の内容のフォーマット設定は、プログラムが使用する編集ルーチンによって決まります。

出力メッセージの形式と内容

端末または別のプログラムに送り返すために作成する出力メッセージの形式は、入力メッセージの形式に似ていますが、各フィールドには異なる情報が入っています。

出力メッセージには、長さフィールド、Z1 フィールド、Z2 フィールド、テキスト・フィールドの 4 種類のフィールドがあります。以下の表は、メッセージ出力のレイアウトを示しています。出力メッセージのフィールド名は、それぞれの表の第 1 行に示されています。各フィールド名の下の数値は、そのフィールドに定義された長さをバイト数で表しています。以下の表は、
AIBTDLI、ASMTDLI、CBLTDLI、CEETDLI、CTDLI および PASTDLI インターフェースの出力メッセージの形式を示しています。 PLITDLI の形式はいくらか異なります。

表 *72.* 出力メッセージ形式

フィールド名	フィールド長
LL	2
Z1	1
Z2	1
テキスト	可変

表 *73. PLITDLI* の場合の出力メッセージの形式

フィールド名	フィールド長
LLLL	4
Z1	1
Z2	1
テキスト	可変

出力メッセージ・フィールドの内容は以下のとおりです。

LL または LLLL
このフィールド長には、LL (または LLLL)、Z1、および Z2 フィールドを含む、メッセージの長さが 2 進数で入ります。出力メッセージ・セグメントの場合には、メッセージ・セグメントを送信する準備ができたらこの長さを指定します。

AIBTDLI、ASMTDLI、CBLTDLI、CEETDLI、CTDLI、および PASTDLI の各インターフェースでは、LL フィールドは 2 バイト長でなければなりません。PLITDLI インターフェースの場合は、LLLL フィールドが 4 バイト長で、メッセージ・セグメントから 2 バイト引いた長さが入ります。

Z1 Z1 フィールドは 1 バイトのフィールドで、必ず 2 進数のゼロでなければなりません。IMS TM 用に予約されています。

Z2 Z2 フィールドは、1 バイトのフィールドで、特別な装置依存の命令 (アラーム・ベルを鳴らす命令、交換回線を切断する命令、ページング命令など) や装置依存の情報 (構造化フィールド・データや MFS のバイパス方法などの情報) が入ります。

これらの命令をいずれも使用しない場合には、Z2 フィールドには必ず 2 進数のゼロを入れます。MFS の場合は、このフィールドには、このメッセージに使用するオプションの番号が入ります。

テキスト
: メッセージ・セグメントのテキスト部分には、論理端末またはアプリケーション・プログラムへ送信するデータが入ります。 (通例、テキスト・メッセージは EBCDIC 文字です。) テキストの長さは、送信するデータによって異なります。

メッセージ処理時

メッセージに対するプログラムの応答は、プログラムが受け取るメッセージのタイプによって異なります。端末からの情報要求は、トランザクション・コードによって、その要求を処理し応答することができるアプリケーション・プログラムに関連付けられます。処理するメッセージに、MPP に関連するトランザクション・コードが入っていると、IMS TM はその MPP をスケジュールします。

例: 在庫の照会に、トランザクション・コード「INVINQ」を処理する MPP を使用しているとします。 MPP が、端末ユーザーから、部品の在庫に関する情報の要求を受け取ります。ユーザーがアプリケーション・プログラムにトランザクション・コードを入力すると、IMS TM がこの要求を処理できるアプリケーション・プログラムをスケジュールします。

INVINQ と 1 つ以上の部品番号を入力すると、MPP は、各部品の在庫の数量と発注済みの部品の数量をユーザーのプログラムに送ります。

端末から INVINQ を入力すると、MPP で INVINQ を処理するために、IMS はメッセージをメッセージ・キューに入れます。さらに、IMS TM が MPP をスケジュールした後、MPP は GU 呼び出しおよび GN 呼び出しを発行してメッセージのリトリーブを行います。アプリケーション・プログラムは、LTERM1 からメッセージをリトリーブするために、メッセージの最初のセグメントに対して GU を発行し、次いで IMS TM が状況コード QD を返すまで GN 呼び出しを発行します。これは、プログラムがメッセージのすべてのセグメントのリトリーブを完了したことを意味します。次にプログラムは要求を処理して、ユーザーの論理端末へのキューに出力メッセージを送ります。 (論理端末名は I/O PCB 内にあります。) MPP が出力メッセージを送信する際、IMS TM はそのメッセージを論理端末へのキューに送り、メッセージは物理端末に送られます。次の図は、端末と MPP との間のメッセージのフローを示しています。

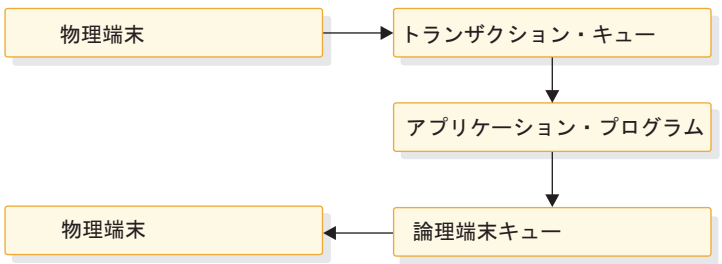


図 74. トランザクション・メッセージのフロー

次の例では、使用する呼び出し、状況コード、および在庫照会の入出力の様子を示しています。 GU および GN を使用してメッセージをリトリーブする方法、および複数セグメントのメッセージを挿入する方法を示すために、この例では 3 つのセグメントが入ったメッセージを示しています。この例の入出力メッセージが 1 つのセグメントから成るメッセージであれば、GU を 1 回だけ発行してメッセージ全体をリトリーブし、ISRT を 1 回だけ発行してメッセージを送信します。

ここで示しているメッセージ形式は 1 つの例であり、すべてのメッセージがこの形式になるわけではありません。プログラムが入出力域に入力メッセージを受信したとき、各セグメントの最初のフィールドには、そのセグメントの長さが入っています。これは図の中の LL フィールドです。分かりやすくするために、図ではこの長さを 10 進数で示していますが、実際に入力メッセージにおいては 2 進数です。 2 番目のフィールド (ZZ) は、IMS TM のために確保されており、長さは 2 バイトです。その予約済みの 2 バイトの後に、メッセージのテキストが続きます。最初のメッセージ・セグメントには、ZZ フィールドに続いて 8 バイトのトランザクション・コードが入ります。これらは、メッセージのテキスト部分の先頭からの 8 バイトです。

出力メッセージの形式は同じです。論理端末の名前を含める必要はありません。これは、その名前が I/O PCB の先頭からの 8 バイトに入るからです。

この例の PART、QTY、および ON ORDER はヘッダーです。これらの値は、端末画面に表示させる定数として定義することができます。MFS 出力メッセージにヘッダーを付けるには、これらの値をリテラルとして定義します。

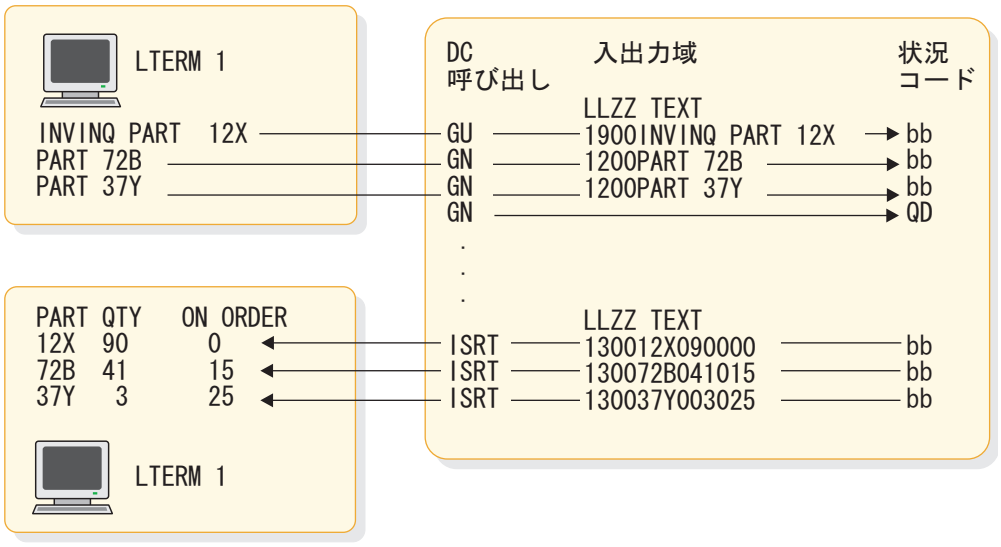


図 *75. MPP* の在庫照会の例

メッセージの結果: I/O PCB

プログラムが呼び出しを発行すると、IMS TM がその呼び出しの結果に関する戻り情報を I/O PCB に返します。呼び出しの結果を知るには、アプリケーション・プログラムで、IMS TM が I/O PCB に返す情報を検査しなければなりません。

ユーザーのアプリケーション・プログラムがメッセージをリトリーブする際、IMS TM は、そのメッセージに関する以下の情報を I/O PCB に返します。

- メッセージを送信した端末の名前。
- 呼び出しの結果を記述している 2 文字の状況コード。メッセージをリトリーブする呼び出しを発行した後にプログラムが QC 状況コードを受信した場合には、そのプログラムが処理すべきメッセージはもうありません。
- 現在の日付、時刻、およびメッセージのシーケンス番号。
- 端末使用者のユーザー ID、またはメッセージを送信したプログラムのトランザクション・コード。

I/O PCB はユーザーのプログラムの外のストレージにあるため、ユーザーは自分のプログラム内に PCB のマスクを定義し、IMS TM 呼び出しの結果を検査します。マスクには、I/O PCB と同じフィールドが同じ順序で入ります。

関連資料:

453 ページの『I/O PCB マスクの指定』

IMS TM によるメッセージの編集方法

アプリケーション・プログラムが端末との間でメッセージを受け渡すとき、IMS TM では、プログラムが端末からメッセージを受け取る前、および端末がアプリケーション・プログラムからメッセージを受け取る前にメッセージを編集します。

IMS TM では、端末画面とプログラムの入出力域の両方でのメッセージの表示方法に多数の選択項目があります。プログラムに指定されている編集ルーチンと、そのルーチンがプログラミングに与える影響を知っておく必要があります。

IMS TM では、LU 6.2 以外の端末に使用できる編集ルーチンが 3 種類あります。

基本編集
: MFS を使用しない場合、および LU6.1 装置からメッセージを発信しない場合に、基本編集機能を実行します。なんらかのフォーマット設定機能に制御文字を指定しなければなりません。

システム間連絡 **(ISC)** 編集
: LU6.1 装置から発信されるメッセージにはデフォルトの編集を行います。テキストに加えて 2 進データを入力することができます。

メッセージ形式サービス **(MFS)**
: 制御ブロックを介してメッセージの形式を設定します。制御ブロックを介してメッセージを表示する方法を定義します。

LU 6.2 装置の場合には、LU 6.2 編集出口ルーチンを使用して入出力メッセージを編集します。

関連資料: LU 6.2 の詳細については、「*IMS V15* コミュニケーションおよびコネクション」を参照してください。LU 6.2 編集出口ルーチンの詳細については、「*IMS V15* 出口ルーチン」を参照してください。

出力メッセージの印刷

出力メッセージを印刷するには、出力メッセージの形式を設定するために必要な、水平方向および垂直方向の制御文字を指定する必要があります。

端末のプリンターで出力メッセージを印刷するには、以下の制御文字をメッセージのテキスト内の必要な場所に組み込んでください。

X'05' タブ停止位置までスキップするが、同じ行にとどまる。

X'15' 改行して左マージンから始める。

X'25' 新しい行にスキップするが、水平方向には同じ位置にとどまる。

複数の行をスキップするには、改行 (X'15') してから、必要な行数分をスキップ (X'25') します。

基本編集の使用

MFS または LU 6.1 装置を使用しない場合には、IMS TM は自動的に何らかの編集を行います。 IMS TM が最初のメッセージ・セグメントに対して行う編集は、IMS TM が後続のメッセージ・セグメントに対して行う編集とは異なります。

基本編集の詳細については、「*IMS V15* コミュニケーションおよびコネクション」を参照してください。

入力メッセージの編集

IMS TM がユーザーのアプリケーション・プログラムについての入力メッセージの最初のセグメントを受信すると、IMS TM は以下のことを行います。

- 先頭および末尾の制御文字を除去する。
- 先行ブランクを除去する。
- バックスペース文字を除去する (端末のプリンターから)。
- システム定義 TRANSACT マクロに EDIT=UC 仕様で指定されている場合、大文字に変換する。

メッセージ・セグメントにパスワードが入っていれば、IMS TM は以下のことを行ってセグメントを編集します。

- パスワードを除去し、その代わりにブランクを挿入する。
- テキストの先頭の文字がブランクであれば、パスワードを除去する。IMS TM はブランクの挿入を行いません。
- セグメントのテキストを左に寄せる。

後続の入力メッセージ・セグメントについては、IMS TM はメッセージのテキストから先行ブランクを除去しません。その他のフォーマット設定機能は同じです。

出力メッセージの編集

出力メッセージについて、基本編集は以下のことを行います。

- データが出力装置に送られる前に、出力メッセージ内の非図形文字を変更する。
- 復帰改行文字、行送り文字、およびタブ文字のあとに必要な任意のアイドル文字を挿入する。

- 通信回線の操作のために、回線制御文字を追加する。

システム間連絡編集の使用

システム間連絡 (ISC) 編集は、LU 6.1 装置からのメッセージのデフォルトの編集機能です。その他の装置タイプについては無効です。ISC 編集を使用する利点の 1 つは、IMS TM がメッセージ・テキストを編集しないため、2 進データを入力できることです。

入力メッセージの編集

IMS TM が入力メッセージに対して行う編集は、機能管理 (FM) ヘッダーに SNA 定義の基本リソース名 (PRN) パラメーターが入っているかどうかによって異なります。どちらの場合も、IMS TM は、アプリケーション・プログラムが入力メッセージを受信する前に FM ヘッダーを除去します。

FM ヘッダーに PRN パラメーターが入っていない場合には、以下のようになります。

- アプリケーション・プログラムへの入力メッセージの最初のセグメントを受信したときに、IMS TM は先頭の制御文字およびブランクを除去する。
- メッセージ・セグメントにパスワードが入っている場合には、IMS TM がパスワードを除去して、その場所にブランクを挿入する。
- IMS TM はメッセージのテキスト (パスワードに続くデータ) を編集しない。

FM ヘッダーに PRN パラメーターが入っている場合には、以下のようになります。

- PRN がトランザクション・コードとして扱われ、メッセージ・セグメントの最初のフィールドとしてアプリケーション・プログラムによって受信される。
- IMS TM はメッセージ・セグメントを編集しない。

出力メッセージの編集

ISC 編集では、出力メッセージの編集は行われません。

メッセージ形式サービスの使用

メッセージ形式サービス (MFS) を使用して、MPP に送信するメッセージを形式設定します。制御ブロックで形式を定義します。

MFS 制御ブロックは、入出力メッセージをどのように配置したいのかを IMS TM に指示します。

- MFS 制御ブロックは、入力メッセージの場合、端末から MPP に送られるメッセージの入出力域での配置を定義します。
- MFS 制御ブロックは、出力メッセージでは、MPP から端末に送られるメッセージの、画面やプリンターでの配置を定義します。言葉やその他のデータを、画面上 (ヘッダーなど) には表示させて、プログラムの入出力域内には表示させないように定義することもできます。リテラルと呼ばれる、この種のデータが、アプリケーション・プログラムからの出力メッセージ内のフィールド、あるいは端末からの入力メッセージ内のフィールドになる場合があります。

端末および MFS

プログラムが MFS を使用するかどうかは、ネットワークで使用される端末と 2 次論理装置 (SLU) のタイプによって決まります。出力メッセージの MFS フォーマット設定は、3270 装置または SLU タイプ 2 の装置ではバイパスすることができます。MFS をバイパスする場合には、ユーザー・プログラム内で 3270 データ・ストリーム全体を組み立てます。

制約事項: MFS は、LU 6.2 装置 (APPC) では使用することはできません。

関連資料: LU 6.2 および APPC の詳細については、「*IMS V15* コミュニケーションおよびコネクション」を参照してください。

MFS の使用には、アプリケーション設計やアプリケーション・プログラミングといった作業とは別の、ハイレベルの設計決定が関係しています。MFS を使用する多くのインストール済み環境には、MFS を使用するすべてのアプリケーションの MFS 画面とメッセージ形式を設計する専門家がいます。

MFS を使用すると、MPP は、メッセージの読み取りと作成の方法を変更することなく、別のタイプの端末と通信することができるようになります。 MPP が端末からメッセージを受信する場合、MPP の入出力域のメッセージ形式は、送信した端末の種類に関係なく、指定されている MFS オプションによって決まります。 MFS は、メッセージを送信している物理デバイスを MPP が認識していなくても済むようにしますが、この方法は、DB PCB が、データベース内のデータの実際の形式と保管方法を、プログラムが認識していなくても済むようにしているのと同じです。

MFS 入力メッセージの形式

フィールドの MFS にメッセージを定義します。これは、データベース・セグメント内にいくつかのフィールドを定義するのと同じです。

1 つのメッセージ・セグメントを構成するときに、MFS に以下のような情報を与えます。

- フィールド長
- 定義されたフィールドの長さよりも入力データが短いときに使用される充てん文字
- フィールド内でデータを左寄せするか右寄せするか
- フィールドを切り捨てるときは左右どちらを切り捨てるか

メッセージ・セグメント内でのこれらのフィールドの配列順序と長さは、プログラムで使用している MFS オプションによって決まります。 MFS オプションは、ユーザーが MID 内で指定します。アプリケーション・プログラムにどのオプションを使用するかは、以下の項目に基づいて決定します。

- 入力データの複雑さ
- 入力データがどれだけ多く変わるか
- アプリケーション・プログラムの作成に使用した言語
- アプリケーション・プログラムの複雑さ
- パフォーマンス要因

MFS メッセージの Z2 フィールドには、プログラムとの間で受け渡すメッセージの形式を設定するのに使用する MFS フォーマット設定オプションが入ります。 IMS TM がユーザーの入出力域にメッセージを返した段階で異常が発生しており、その原因が使用中の MFS オプションにあると思われる場合には、このフィールドを検査すれば、IMS TM で正しいオプションが使用されているかどうかを確認することができます。このフィールドが X'00' であれば、それは MFS がメッセージをまったく形式設定しなかったことを表します。

各 MFS オプションによって入出力メッセージがどのように形式設定されるかを知る方法の 1 つは、各オプションの例を調べることです。

例: ユーザーが次の表に示す 4 つのメッセージ・セグメントを定義したと仮定します。各セグメントには、2 バイトの長さフィールドと 2 バイトの ZZ フィールドが入っています。最初のセグメントには、端末使用者がアプリケーション・プログラムを呼び出すために入力したトランザクション・コードが入っています。この図では、各フィールドに対して定義したバイト数が、フィールド名の下に表示されています。

PLITDLI インターフェースを使用する場合には、長さフィールドを 2 進数のフルワード LLLL で定義しなければなりません。AIBTDLI、ASMTDLI、CBLTDLI、CEETDLI、CTDLI、PASTDLI の各インターフェースを使用する場合は、長さフィールドをハーフワードの LL で定義する必要があります。PL/I アプリケーション・プログラムでは、実際のセグメント長から 2 バイトを差し引いた値をそのフィールドに入れなければなりません。例えば、出力のテキストが 10 バイトのとき、フルワードの LLLL の値は 14 です。これは、LLLL (4 バイト - 2 バイト) + Z1 (1 バイト) + Z2 (1 バイト) + TEXT (10 バイト) の合計の長さを表します。

表 *74. 4* セグメントのメッセージ :

セグメント番号	フィールド名	フィールド長	フィールド値
1	LL	2	0027
	ZZ	2	XXXX
	TRANCODE	8	YYYY
	テキスト	5	PATIENT#
	テキスト	10	NAME
2	LL	2	0054
	ZZ	2	XXXX
	テキスト	50	ADDRESAF
3	LL	2	0016
	ZZ	2	XXXX
	テキスト	6	CHARGES
	テキスト	6	PAYMENTS
4	LL	2	0024
	ZZ	2	XXXX
	テキスト	10	TREATMENT
	テキスト	10	DOCTOR

これらの例では、以下のことを仮定しています。

- トランザクション・コードは、MID ではリテラルとして定義されています。
- フィールドはすべて左寄せされます。
- 充てん文字はブランクとして定義されています。フィールド内のデータの長さが、定義されているフィールドの長さより短い場合は、MFS によって充てん文字が埋め込まれます。充てん文字としては、以下のものを使用できます。
  - ブランク
  - 1 つの EBCDIC 文字
  - 1 つの EBCDIC 図形文字
  - ヌル (X'3F' で指定)

  充てん文字としてヌルを指定した場合は、データの長さがフィールド長より短ければ、MFS はそのフィールドをデータの長さにまで圧縮します。

  前の表に示したメッセージのセグメント 4 のフィールドは、端末画面では次の図に示す形式で配置されます。

  例: 患者の名前と、その患者に請求した診察料金と支払金額を入力したとします。

```
PATIENT#:          NAME:  MC ROSS

ADDRESAF:

CHARGES: 106.50    PAYMENTS:  90.00

TREATMENT:

DOCTOR:
```

図 *76. MFS* の端末画面の例

MFS では、メッセージのフォーマット設定として 3 つのオプションが使用できます。

MFS オプション 1

このオプションは、そのプログラムがメッセージ・セグメント内のほとんどのフィールドを受信および転送するときに使用します。オプション 1 によるメッセージのフォーマット設定の方法は、セグメントの中のいずれかのフィールドに充てん文字としてヌルを定義したかどうかによって決まります。

メッセージ内のどのフィールドにも、充てん文字としてヌルを定義していない場合は、以下のようになります。

- プログラムがメッセージ内のすべてのセグメントを受信します。
- 各セグメントの長さは、MID 内でセグメントの長さとして指定されている長さです。
- 各セグメント内には、その中のすべてのフィールドが入っています。
- 各フィールドには、データ、データと充てん文字、または充てん文字のみが入っています。

次の表は、アプリケーション・プログラムが受け取ったセグメントのオプション 1 形式を示しています。

表 *75. MFS* オプション *1* メッセージ形式

セグメント番号	フィールド名	フィールド長	フィールド値
1	LL	2	0027
	Z1	1	XX
	Z2	1	01
	TRANCODE	8	YYYY
	テキスト	5	ブランク
	テキスト	10	MCROSSbbbb
2	LL	2	0054
	Z1	1	XX
	Z2	1	01
	テキスト	50	ブランク
3	LL	2	0016
	Z1	1	XX
	Z2	1	01
	テキスト	6	010650
	テキスト	6	009000
4	LL	2	0024
	Z1	1	XX
	Z2	1	01
	テキスト	10	ブランク
	テキスト	10	ブランク

オプション 1 の出力メッセージの形式は、入力メッセージの形式と同じです。プログラムは、前の図でセグメント 4 として示されている形式で出力メッセージを入出力域に作成します。プログラムは、以下のいずれかの方法で、フィールドを切り捨てたり省いたりすることができます。

- 短セグメントを挿入する方法
- フィールド内にヌル文字を入れる

1 つ以上のフィールドにヌルの充てん文字を入れるものとして定義した場合は、メッセージの形式は異なります。この場合、メッセージは以下の特性を持つことになります。

- ヌルの充てん文字が入ると定義されているフィールドに、端末からなにもデータが渡されなかった場合、そのフィールドは、メッセージ・セグメントから除去されます。
- セグメント内のすべてのフィールドにヌルの充てん文字が定義されており、どのフィールドにもリテラルが入っていない場合は、そのセグメントがメッセージから除去されます。

- セグメント内の一部のフィールドだけがヌルの充てん文字を持つと定義されている場合は、ヌルが入っているフィールドはセグメントから除去されます。このとき、セグメント内の残りのフィールドの相対位置は変更されます。
- 発信元端末から受信したデータの長さが、フィールドに対して定義されている長さより短い場合、フィールドは、データの長さにそろえて切り捨てられます。

MFS オプション 2

このオプションは、ほとんどのフィールドが転送されるが一部のセグメントが省略される複数セグメントのメッセージをプログラムが処理するときに使用します。オプション 2 は、オプション 1 と同じ方法でメッセージを形式設定します。ただし、これは IMS TM がリテラルを取り除いてしまっても端末からの入力データがまだセグメントに残っている場合です。この場合、または、メッセージ内のその他のセグメントに端末からの入力データが入っていない場合は、IMS TM がメッセージを終了させます。プログラムが受信する最後のセグメントは、端末からの入力データが入っている最後のセグメントです。

端末からの入力データが入っていないセグメントの後に、端末からの入力データが入っているセグメントがいくつか続くことがあります。このような場合、MFS は、セグメントの長さフィールドと Z フィールド、さらに、X'3F' が入っている 1 バイト・フィールドを続けて、プログラムに渡します。これによって、このセグメントがヌル・セグメントであることがプログラムにわかります。

481 ページの表 74 で示したメッセージ・セグメントがオプション 2 で形式設定される場合、そのメッセージ・セグメントは以下の表に示す形式になります。

表 *76. MFS* オプション *2* メッセージ形式

セグメント番号	フィールド名	フィールド長	フィールド値
1	LL	2	0027
	Z1	1	XX
	Z2	1	02
	TRANCODE	8	YYYY
	テキスト	5	ブランク
	テキスト	10	MCROSSbbbb
2	LL	2	0005
	Z1	1	XX
	Z2	1	02
	テキスト	1	X'3F'
3	LL	2	0016
	Z1	1	XX
	Z2	1	02
	テキスト	6	010650
	テキスト	6	009000

前の表のセグメント 2 はヌル・セグメントであるため、X'3F' しか入っていませんが、セグメント 3 にはデータが入っています。セグメント 4 はヌルであるため、このメッセージにはセグメント 4 は入っていません。

MFS オプション 3

このオプションは、そのプログラムがセグメント内の数個のフィールドのみを受信および転送するときに使用します。オプション 3 を使用すると、プログラムは、端末から受信したフィールドだけを受け取ります。プログラムは、発信元端末から受信したフィールドが入っているセグメントだけを受信します。オプション 3 でヌル充てん文字の使用を定義しておけば、セグメントとフィールドの長さを可変長にすることができます。

オプション 3 のメッセージ内のセグメントは、相対セグメント番号 (つまり、メッセージ内で占める位置) によって識別されます。セグメント内のフィールドは、セグメント内におけるフィールドのオフセット・カウントによって識別されます。

例: セグメント 1 の NAME フィールドは (MCROSSbbbb) です。17 という値は、NAME フィールドの前にあるすべてのフィールドの長さを合計した値で、8 バイトのトランザクション・コードと 5 バイトのブランク・フィールドも含まれています。この値には、2 バイトの相対セグメント番号フィールド (次の表のフィールド A)、2 バイトの長さフィールド (フィールド B)、それに 2 バイトの相対オフセット・フィールド (フィールド C) のいずれも含まれていません。

オプション 3 のメッセージには、MID 内で定義されているリテラルは入りません。これは、会話中の場合以外は、トランザクション・コードがメッセージから除去されることを意味しています。プログラムが会話型トランザクションを処理している場合は、トランザクション・コードはメッセージから除去されません。スクラッチパッド域 (SPA) にはトランザクション・コードが入っています。

プログラムが受け取る各セグメントには、メッセージ内のこのセグメントの相対番号が入っています (次の表のフィールド A)。さらに、セグメント内の各データ・フィールドの前にも以下の 2 つのフィールドがあります。

- 2 バイトの長さフィールド (B)。長さフィールド自体、2 バイトの相対フィールド・オフセット、およびフィールド内のデータ長が含まれます。
- 2 バイトの相対フィールド・オフセット (C)。MID 内で定義されているセグメント内でのフィールドの位置を示します。

これら 2 つのフィールドのあとにデータ・フィールドが続きます。MFS は、アプリケーション・プログラムに返す各フィールドごとに上記のフィールドを組み込みます。

481 ページの表 74 で示したメッセージ・セグメントがオプション 3 で形式設定される場合、そのメッセージ・セグメントは次の表に示す形式になります。セグメント 1 とセグメント 3 の最初の行に示されている文字 A、B、C、D については、この表の注で説明しています。

表 77. *MFS* オプション 3 メッセージ形式

セグメント番号	フィールド名	フィールド長	フィールド値
1	LL	2	0020
	Z1	1	XX
	Z2	1	03
	A	2	0001
	B	2	0014
	C	2	0017
	D	10	MCROSSbbbb
2	LL	2	0000
	Z1	1	XX
	Z2	1	03
	A	2	0003
	B	2	0010
	C	2	0004
	D	6	010650
	B	2	0010
	C	2	0010
	D	6	009000

前の表の注**:**

- A のマークが付いているフィールドには、相対セグメント番号が入ります。この番号は、メッセージ内におけるセグメントの位置を表しています。
- B のマークの付いているフィールドには、フィールド長が入ります。この長さは、B フィールド (2 バイト) + C フィールド (2 バイト) + D フィールド (データ長) の合計を表しています。
- C のマークが付いているフィールドには、相対フィールド・オフセットが入ります。これによって、そのセグメント内における各フィールドの位置がわかります。
- D のマークが付いているフィールドには、端末から入力したデータが入ります。この例では、充てん文字としてブランクが定義されているので、データ・フィールドの長さは、常に定義されているとおりの長さです。IMS TM は、その長さを切り捨てることはしません。充てん文字としてヌルを定義した場合には、セグメント内のデータ・フィールドの長さが、定義したとおりの長さとは異なることがあります。ヌルを充てん文字として使用した場合、端末から受信するデータの長さが、フィールドに対して定義されている長さより短ければ、IMS TM はそのフィールドをデータの長さにそろえて切り捨てます。オプション 3 でヌルの充てん文字を使用すると、メッセージに必要なスペースをさらに減らすことができます。

MFS 出力メッセージの形式

出力メッセージ形式は、アプリケーション・プログラムから MFS が受け取るセグメントとフィールドを定義するために使用されます。

オプション 1 またはオプション 2 を使用するのであれば、出力メッセージの形式は入力メッセージの形式と同じです。すべてのフィールドとセグメントを MFS に渡します。ヌル・セグメントを渡すこともできます。出力メッセージのすべてのフィールドは固定長で、固定位置です。出力メッセージにはオプション番号が入りません。

オプション 3 の出力メッセージは、オプション番号が入らない点を除けば、入力メッセージの場合と似ています。プログラムは、各セグメントに必要なフィールドを、その位置指定情報を付けて渡します。

LU 6.2 ユーザー編集出口ルーチンの使用 (オプション)

この出口ルーチンは、暗黙のアプリケーション・プログラム・インターフェース・サポートを使用するときに、LU 6.2 装置からの入出力メッセージを編集します。

指定しない場合には、メッセージが修正なしで表示されます。IMS は CPI-C ドリブン・トランザクションのために出口ルーチンを呼び出すことはありません。これは、アプリケーション・プログラムが CPI を直接使用するときに、IMS がデータ・フローに関与しないからです。

各メッセージ・セグメントごとに、またはインバウンド制御フローごとに、LU 6.2 ユーザー編集出口ルーチンが 1 回呼び出されます。データ・メッセージに対して出口ルーチンを呼び出したり、それを使用して以下のことを行うことができます。

- メッセージ・セグメントの内容の検査
- メッセージ・セグメントの内容の変更
- メッセージ・セグメントの内容の拡張または短縮
- メッセージ・セグメントの廃棄、および、もしあれば後続のセグメントの処理
- DEALLOCATE_ABEND コマンドによる会話の終了

LU 6.2 ユーザー編集出口ルーチンの詳細については、「*IMS V15* コミュニケーションおよびコネクション」および「*IMS V15* オペレーションおよびオートメーション」を参照してください。

DB2 のためのメッセージ処理の考慮事項

DB2 データベースにアクセスする従属領域のメッセージ処理機能と DL/I データベースだけにアクセスする従属領域のメッセージ処理機能は、大部分が同じです。

メッセージのリトリーブと送信、データベースの変更をバックアウトするためにプログラムが用いている方法は同じです。異なる点は、以下のとおりです。

- DL/I ステートメントのコーディング方法が SQL (構造化照会言語) ステートメントの場合とは異なる。
- IMS TM アプリケーション・プログラムが IMS TM から制御を受け取った時点では、IMS は既にプログラムがアクセスできるリソースを獲得している。IMS TM は、プログラムをスケジュールしますが、データベースの中には利用可能な状態でないものもあります。DB2 では、プログラムが最初の SQL ステートメントを出すまでは、そのプログラムに対してリソースを割り当てません。プログラ

ムが必要とするリソースを DB2 が割り振ることができない場合には、プログラムが最初の SQL 呼び出しを発行した時点で初期設定エラーをオプションで受け取るようにすることもできます。

- アプリケーションが出したチェックポイント呼び出しまたはメッセージ GU 呼び出しが成功すると、DB2 はプログラムが使用しているすべてのカーソルをクローズする。つまり、ユーザー・プログラムは、チェックポイント呼び出しまたはメッセージ GU の後で、OPEN CURSOR ステートメントを出さなければなりません。

IMS TM と DB2 は連携して、以下の方法によりデータ保全性を維持します。

- ユーザー・プログラムがコミット・ポイントに達すると、IMS TM は、プログラムが DL/I データベースに対して行った変更を永続変更とし、出力メッセージを宛先に送付し、プログラムがコミット・ポイントに達したことを DB2 に通知する。次に DB2 は、プログラムが DB2 データベースに対して行った変更を永続化させます。
- ユーザー・プログラムが異常終了するか、または IMS TM のロールバック呼び出し (ROLB、トークンなしの ROLS、または ROLL) のいずれかを出した場合、IMS TM は、ユーザー・プログラムが作成したすべての出力メッセージを取り消し、最後のコミット・ポイント以降にユーザー・プログラムが DL/I データベースに対して行った変更をバックアウトし、DB2 に通知します。DB2 は、最後のコミット・ポイント以降にユーザー・プログラムが DB2 データベースに対して行った変更をバックアウトします。

自動化操作プログラム・インターフェース (AOI) により、IMS TM アプリケーション・プログラムは、DB2 コマンドおよび IMS TM コマンドを発行できます。DB2 コマンドを発行するには、DB2 コマンドの前に IMS TM /SSR コマンドを発行します。 /SSR コマンドの出力は、マスター端末オペレーター (MTO) に送られます。

他の端末およびプログラムへのメッセージの送信

アプリケーション・プログラムは、端末から送られてきたメッセージを処理すると、通常は、入力メッセージを送ってきた端末に応答を送ります。ところが、場合によっては、出力メッセージを発信元端末以外の端末に送る場合や、発信元端末に加えて他の端末にも送る場合があります。また、別のアプリケーション・プログラムにメッセージを送ることもあります。

代替 PCB を使用する際には、以下のようになります。

- 出力メッセージを 1 つの代替宛先に送りたい場合は、その宛先についての代替 PCB を定義する。
- 出力メッセージを複数の代替宛先に送るときに、代替 PCB の宛先を変更できるようにしたい場合には、プログラム仕様ブロック (PSB) 生成時に代替 PCB を変更可能と定義する。次いで、ISRT 呼び出しを発行する前に、CHNG 呼び出しを発行して、変更可能代替 PCB の宛先を宛先プログラムまたは端末に設定します。

  特殊な代替 PCB である高速代替 *PCB* は、PSB の生成時に EXPRESS=YES を指定して定義されます。

高速代替 PCB を使用すると、その PCB を使用して送るメッセージは、最終的な宛先に即時に送られます。その他の PCB で送られたメッセージは、そのプログラムがコミット・ポイントに達するまでは、一時宛先に送られます。高速 PCB で送られるメッセージは、プログラムが後で異常終了するか、あるいはロールバック呼び出し ROLL、ROLB、または ROLS のいずれかを呼び出した場合でも送られます。このような状態で高速代替 PCB を使用することは、異常終了が発生してもプログラムを確実に端末使用者に通知できる方法の 1 つです。プログラムが異常終了するか、または ROLL、ROLB、あるいは ROLS 呼び出しのいずれかを発行した場合、すべての PCB について、挿入はされたが伝送用に使用可能にされなかったメッセージは取り消されますが、伝送用に使用可能にされたメッセージが取り消されることはありません。

非高速 PCB の場合には、プログラムがコミット・ポイントに達するまでは、メッセージを宛先に転送することはできません。コミット・ポイントは、プログラムが終了するとき、CHKP 呼び出しを発行したとき、または、次の入力メッセージを要求し、トランザクションが MODE=SNGL で定義されているときに発生します。

高 PCB では、IMS TM は、完結したメッセージであると判断したとき、そのメッセージを宛先に伝送できるようにします。コミット・ポイントでの発生に加えて、アプリケーション・プログラムが、その PCB を使用して PURG 呼び出しを発行したとき、または次の入力メッセージを要求したときにも、このメッセージの使用可能化が起こります。

PSBGEN は、代替 PCB を PSB 生成時に定義された代替応答 PCB として指定することもできます。

- LU 6.2 装置にメッセージを送る場合には、その装置に関連する LU 6.2 記述子名を指定することができる。IMS は、宛先名 (CNT または SMB) の大文字への変換を内部的に実行します。

関連資料:

458 ページの『代替 PCB マスクの指定』

他の端末へのメッセージの送信

発信元端末以外の端末に応答を返す場合も ISRT 呼び出しを使用しますが、そのときには TP PCB ではなく代替プログラム連絡ブロック (PCB) を使用します。

TP PCB がメッセージを送ってきた端末を表すのと同じように、代替 PCB は、送信するメッセージを受け取る端末を表します。

単一代替端末への送信

1 つの代替端末だけにメッセージを送るのであれば、PSB 生成時にその端末に対する代替 PCB を定義することができます。特定の宛先に対する代替 PCB を定義した場合は、プログラムの実行中にその宛先を変えることはできません。つまり、その PCB を参照する ISRT 呼び出しを発行するたびに、代替 PCB で名前が指定されている論理端末にメッセージが送られます。その端末にメッセージを送るには、一度に 1 つのメッセージ・セグメントを入出力域に入れ、TP PCB ではなく代替 PCB を指定した ISRT 呼び出しを発行します。

複数の代替端末への送信

複数の端末にメッセージを送るのであれば、PSB 生成時に、代替 PCB を変更可能として定義することができます。したがって、代替 PCB は複数の代替端末を表します。宛先は、プログラムの実行中に変更することができます。

代替 PCB の宛先を設定したり変更したりする前に、その代替 PCB に関してこれまでに作成したメッセージが終了したことを、IMS TM に知らせなければなりません。これを行うには、PURG 呼び出しを発行します。

PURG を使用すると、1 つの入力メッセージを処理しながら複数の出力メッセージを送信することができます。PURG を使用しない場合は、IMS TM はメッセージ・セグメントをまとめて 1 つのメッセージにしておき、プログラムが新規メッセージに対して GU を出したとき、終了するとき、または、コミット・ポイントに到達したときに、まとめておいたメッセージ・セグメントを送ります。 PURG 呼び出しは、この TP PCB または代替 PCB に対して (メッセージ・セグメントごとに ISRT 呼び出しを 1 回出すことによって) 作成中のメッセージが完成したことを、IMS TM に知らせます。IMS TM は、1 つの PCB に 1 つのメッセージとして挿入されたメッセージ・セグメントを収集し、参照した代替 PCB が指す宛先に送ります。

入出力域のアドレスが入っていない PURG 呼び出しは、このメッセージが完成したことを IMS TM に知らせます。入出力域のアドレスが入っている PURG 呼び出しは、ISRT 呼び出しと同じ働きをします。IMS TM は、入出力域内のデータを、新しいメッセージの最初のセグメントと見なします。PURG 呼び出しに入出力域を指定したときは、このメッセージを出す画面の形式を変更するために、MOD 名も指定することができます。MOD 名の指定は任意ですが、指定する場合は、1 つのメッセージにつき 1 回だけ指定するか、あるいはメッセージを開始する最初の ISRT または PURG でのみ指定します。

プログラムの実行中に、変更可能代替 PCB の宛先を設定するには、CHNG 呼び出しを使用します。 CHNG 呼び出しを発行するときに、メッセージの宛先である論理端末の名前を指定します。すると、以下のいずれかを行うまでは、代替 PCB にその宛先が設定されたままになります。

- 別の CHNG 呼び出しを発行して宛先をリセットする。
- メッセージ・キューに対し別の GU を出して新しいメッセージの処理を開始する。この場合、代替 PCB 内にはまだ名前が残っていますが、もはやその名前は無効です。
- プログラムを終了させる。この場合、IMS TM は宛先をブランクにリセットします。

代替 PCB の先頭の 8 バイトには、メッセージの宛先である論理端末の名前が入ります。

CHNG 呼び出しを発行するときには、使用する代替 PCB のアドレスと、その代替 PCB に設定したい宛先名とを IMS TM に与えます。

PURG 呼び出しを使用するときは、IMS TM に代替 PCB のアドレスだけを与えます。 IMS TM は、その PCB を使って作成したメッセージを送信します。

エラーの状態を表示するために、高速 PCB を指定して、ISRT 呼び出しと PURG 呼び出しを続けて出すことによりメッセージを送ることができます。これら 2 つの呼び出しにより、メッセージはその最終宛先に即時に送られます。

例: プログラムが以下のステップに従うこともあり得ます。

1. プログラムが、入力メッセージをリトリーブするために GU 呼び出し (および、必要ならば何回かの GN 呼び出し) を出す。
2. プログラムが、メッセージの処理中に異常条件を検出する。
3. プログラムが、PURG 呼び出しを発行して、新しいメッセージの開始を IMS TM に知らせる。
4. プログラムが、CHNG 呼び出しを発行して、高速 PCB の宛先を、発信元の論理端末の名前に設定する。プログラムは、I/O PCB の先頭の 8 バイトからこの名前を得ることができます。
5. プログラムが、必要なだけ ISRT 呼び出しを発行して、メッセージ・セグメントを送り出す。この ISRT 呼び出しでは、高速 PCB を参照します。
6. プログラムが、高速 PCB を参照する PURG 呼び出しを発行する。次に、IMS TM はメッセージを最終宛先に送ります。
7. その後でプログラムが異常終了するか、あるいは ROLL、ROLB、または ROLS のいずれかの呼び出しを発行し、最後のコミット・ポイント以降にプログラムが作成したデータベース更新をバックアウトし、出力メッセージを取り消す。

出力メッセージが 3 つのセグメントで構成されており、PURG 呼び出しを用いてメッセージの終わりを伝える (しかも次のメッセージ・セグメントを送らない) 場合は、以下の呼び出しシーケンスを使用することができます。

```
CHNG ALTPCB1, LTERMA
ISRT ALTPCB1, SEG1
ISRT ALTPCB1, SEG2
ISRT ALTPCB1, SEG3
PURG ALTPCB1
CHNG ALTPCB1, LTERMB
ISRT ALTPCB1, SEG4
ISRT ALTPCB1, SEG5
ISRT ALTPCB1, SEG6
```

他の IMS アプリケーション・プログラムへのメッセージの送信

プログラム間通信は、IMS 従属領域で実行されている IMS アプリケーション・プログラムが、IMS 従属領域で実行されている別の IMS アプリケーションにメッセージを送るときに行われます。

プログラム間通信を実行して、以下のいずれのタイプの IMS アプリケーションともメッセージの送受信を行うことができます。

- メッセージ処理プログラム (MPP) (message processing program (MPP))
- バッチ・メッセージ処理プログラム (batch message processing (BMP) program)
- Java メッセージ処理 (JMP) プログラム
- Java バッチ処理 (JBP) プログラム

メッセージを別のオンライン・プログラムに送信するには、メッセージを代替端末に送信するのと同様の方法で、代替プログラム連絡ブロック (PCB) を使用します。

1 つのアプリケーション・プログラムだけにメッセージを送るのであれば、そのアプリケーション・プログラムのトランザクション・コードを持つ代替 PCB を、PSB 生成時に定義しておくことができます。複数のアプリケーション・プログラムにメッセージを送るには、代替 PCB を変更可能 PCB として定義しておくことができます。

変更可能代替 PCB を使用する場合、CHNG 呼び出しを発行して変更可能代替 PCB の宛先を設定すると、IMS TM がセキュリティー検査を行います。端末から入力するトランザクション・コードでメッセージ通信を開始するためには、 CHNG 呼び出しで変更可能代替 PCB に入れられるトランザクション・コードを入力する許可を端末が持っていなければなりません。ISRT 呼び出しを発行した場合は、IMS TM によるセキュリティー検査は行われません。

IMS TM アプリケーション・プログラムが CHNG 呼び出しを発行すると、リソース・アクセス管理機能 (RACF) が呼び出され、発行されたトランザクション・コードを処理する権限が発信元端末にあるかどうかを判別する検査が行われます。CHNG 呼び出しを使う代わりに、プログラムで事前に設定した代替 PCB に対する ISRT 呼び出しを発行すると、環境に関係なくセキュリティー検査は行われません。

プログラム間メッセージ通信を実行する場合の考慮事項は、論理端末と通信を行う際の考慮事項と同様です。次の点に注意してください。

- 送られてくる最大のセグメントが入るだけの大きさの入出力域を作成する。
- メッセージを送信するには、TP PCB ではなく、代替 PCB を使用する。
- 代替 PCB の最初のフィールドにプログラムのトランザクション・コードを入れるために CHNG 呼び出しを発行してから、ISRT 呼び出しを発行する。PSBGEN の実行中にこのトランザクション・コードが代替 PCB に設定されていれば、ユーザーは ISRT 呼び出しを発行します。
- IMS TM はトランザクション・コードを認識していなければならない。このトランザクション・コードは、システム定義時に定義します。
- 非会話型プログラムは、別の非会話型プログラムとプログラム間メッセージ通信を行うが、会話型プログラムとの間ではメッセージ通信を行わない。
- 会話型プログラムは、別の会話型プログラムまたは非会話型プログラムのいずれとも、プログラム間メッセージ通信を行うことができる。

Open Transaction Manager Access (OTMA) のプログラム間通信には、次の制約事項があります。

- 同期 APPC/OTMA サポート (DFSDCxxx PROCLIB メンバーで AOS=Y) および RRS サポート (始動プロシージャーで RRS=Y) の両方が有効な共用キュー環境では、別の IMS システムとのアウトバウンド APPC 保護会話を開始するバックエンド IMS システム上で稼働するアプリケーション・プログラムは、単一のプログラム間通信に制限されます。
- APPC アウトバウンド保護会話を別の IMS システム上に割り振った後でアプリケーション・プログラムで複数のプログラム間通信を実行すると、結果は予測不能となり、メッセージ処理領域 (MPR) で WAIT-RRS/PC 状態が発生する可能性があります。

別の会話型プログラムへメッセージ通信が行われると、スクラッチパッド域 (SPA)、および発信元端末への応答義務が、メッセージを受け取る側の新しいアプリ

ケーション・プログラムに転送されます。非会話型プログラムへのメッセージ通信の場合は、会話型プログラムの責任は変わりません。つまり、会話型プログラムは、SPA を IMS TM に返し (SPA が修正されている場合)、発信元端末に応答する必要があります。次の表に、アプリケーション・プログラムへの出力メッセージの形式を示します。

表 *78. AIBTDLI*、*ASMTDLI*、*CBLTDLI*、*CEETDLI*、*CTDLI*、および *PASTDLI* の各インターフェースの場合のプログラム間メッセージ通信のメッセージ形式

フィールド名	フィールド長
LL	2
Z1	1
Z2	1
テキスト	可変

表 *79. PLITDLI* インターフェースのプログラム間メッセージ通信のメッセージ形式

フィールド名	フィールド長
LLLL	4
Z1	1
Z2	1
テキスト	可変

この形式は、端末への出力メッセージの形式と同じです。Z1 と Z2 のフィールドには、2 進数のゼロが入っていなければなりません。これらのフィールドは、IMS 用に予約されています。アプリケーション・プログラムに送るメッセージ・セグメントは、テキスト・フィールドに入ります。

IMS TM が、交換されるメッセージの最初のセグメントにトランザクション・コードを自動的に入れることはありません。そのため、メッセージを処理するプログラムでトランザクション・コードが必要な場合は、メッセージの最初のセグメントのメッセージ・テキストの一部として、受信側プログラムのトランザクション・コードを組み込んでおく必要があります。メッセージが端末または他のプログラムのいずれから送られるにしても、トランザクション・コードを最初のセグメントのメッセージ・テキストに入れておくことにすれば、すべてのメッセージの最初のセグメントが同じ形式に保たれます。

関連概念:

505 ページの『別の会話型プログラムへの会話の引き渡し』

関連タスク:

850 ページの『JMP および JBP アプリケーションでのプログラム間通信』

VTAM 入出力機能の VTAM 端末への影響

VTAM 端末は、IMS から送られた要求への応答に失敗する場合があります。マスター端末オペレーターまたは自動化操作プログラム・インターフェース・アプリケーション・プログラムは、オプションで「タイムアウト」機能を活動化させることができます。この機能を使用することにより、一定の時間が経過したことを伝えるメッセージを、マスター端末オペレーターに送信することが可能になります。

以下のいずれかを行うように IMS TM を設定しておくことができます。

- 何もしない。これは、端末が活動状態にないことを意味します。これはデフォルトです。
- マスター端末オペレーターに、指定の時間が経過したことを伝えるメッセージを送信する。なんらかのアクションが必要であれば、オペレーターがその後どのようなアクションをとるべきかを判別します。
- マスター端末オペレーターに、指定の時間が経過したことを伝えるメッセージを送信する。 IMS TM は、コマンド VTAM `VARY NET,INACT`、およびそれに続けて VTAM `VARY NET,ACT` を発行します。端末が IMS TM に対して非共用で動作可能と定義されており、しかも IMS TM がシャットダウンされていない場合、IMS TM はその端末について OPNDST を出します。

  制約事項: このオプションは、ISC 端末には適用されません。ユーザーのインストール先でこのオプションを選択し、ISC 端末がタイムアウトになると、指定の時間が経過したことを伝えるメッセージが、マスター端末に送信されます。何らかのアクションが必要であれば、オペレーターがどのようなアクションをとるべきかを判別します。

複数システム結合機能を使用した他の IMS TM システムとの通信

IMS TM システムのプログラムや端末との通信に加えて、複数システム結合機能 (MSC) を使用することにより、別の IMS TM システムのある端末やプログラムとも通信することができます。

MSC で、複数の別個の IMS TM システム間を連結することによって、このことが可能になります。各 IMS TM システムの端末やトランザクション・コードは、そのシステムに属するものとして定義されます。システム内の端末やトランザクション・コードは、「ローカル」と呼び、MSC リンクで接続されている他の IMS TM システムで定義されている端末やトランザクション・コードは、「リモート」と呼びます。

関連資料: MSC の概要については、「*IMS V15* コミュニケーションおよびコネクション」を参照してください。

プログラム・コーディングにおける MSC の意味

ほとんどの場合、リモート端末またはリモート・プログラムとの通信が、プログラムのコーディングに影響を与えることはありません。システム間のメッセージ経路指定は、MSC によって処理されます。

例えば、リモート端末から入力メッセージを受け取ってその端末に応答する場合は、I/O PCB に対して `ISRT` 呼び出しを発行します。これは、システムの端末に応答を送る場合とまったく同じです。

次の 2 つの状況では、MSC がプログラミングに影響を与える可能性があります。

- 入力メッセージがリモート端末またはローカル端末のどちらから出されたかを、プログラムが知る必要がある場合。例えば、それぞれ別個の IMS TM システム

内にある 2 つの端末の論理端末名が同じである場合は、どちらのシステムからメッセージが送られてきたかを知ることによって、プログラムの処理が違ってきます。

- 別の IMS TM システムの代替宛先にメッセージを送る場合。

制約事項: LU 6.2 装置で割り振られたトランザクションが MSC リンクを経てリモート・システムに送られる場合には、IMS は、メッセージ `TP_NOT_Avail_No_Retry` を出してそのトランザクションを拒否します。

直接経路指定により、入力メッセージが使用中のシステムから送られたかリモート・システムから送られたかをプログラムが判断することができ、また、出力メッセージの宛先を別の IMS TM システムの代替宛先に設定することができます。直接経路指定を使用すれば、別の IMS TM システムの代替宛先がユーザーのシステムでリモート宛先として定義されていなくとも、その宛先にメッセージを送ることができます。

制約事項: MSC 直接経路指定は、会話型トランザクションの間のプログラム間通信をサポートしません。

関連資料: LU 6.2 および MSC 直接経路指定についての詳細は、「*IMS V15* コミュニケーションおよびコネクション」を参照してください。

他の IMS TM システムからのメッセージの受信

アプリケーション・プログラムは、入力メッセージを受け取るとき、その入力メッセージが同じ IMS TM システムの端末またはプログラムから送られてきたのか、または別の IMS TM システムの端末またはプログラムから送られてきたのかを判別することができます。アプリケーション・プログラムでの処理は、入力メッセージがローカル端末ではなくリモート端末から送られたかどうかによって変わることがあります。

例えば、IMS TM システムをシステム A とし、システム B という別の IMS TM システムにリンクされているものとします。MSC リンクは一方向のリンクです。システム A からシステム B へのリンクを LINK1 とし、システム B からシステム A へのリンクを LINK2 とします。MPP1 という名前のアプリケーション・プログラムはシステム A で実行されます。両方のシステムとも、マスター端末の論理端末名は MASTER です。次の図は、システム A とシステム B を示しています。

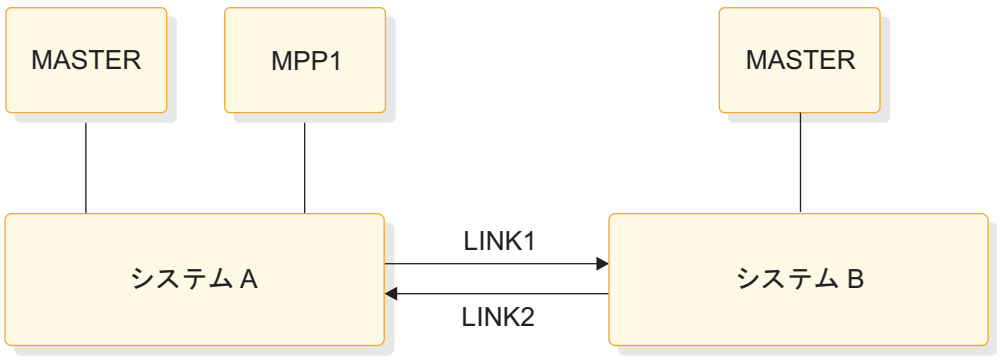


図 77. *MSC* の例

システム B の MASTER 端末が、システムをシャットダウン中であるというメッセージをシステム A の MPP1 に送る場合、メッセージがシステム B の MASTER から送られたものであって、システム A の MASTER からのものではないということを、MPP1 は知る必要があります。

IMS TM システム定義時に TRANSACT マクロで ROUTING=YES が指定されていると、別の IMS TM システムの端末から送られてきたメッセージであることをプログラムに知らせるために、IMS TM は 2 つのことを行います。

最初に、IMS TM は、I/O PCB の最初のフィールドに、論理端末名ではなく、MSC 論理リンク名を入れます。この例の場合は、LINK1 です。これは、システム定義の際に MSNAME マクロで指定した論理リンク名です。ただし、その後でメッセージが発信元システムへ送り返されると、I/O PCB の最初のフィールドには、発信元端末の LTERM 名が再び入れられます。

2 番目に、IMS TM は、I/O PCB の IMS 用に予約されているフィールドの 1 つのビットをオンにします。オンにされるのは、2 バイト・フィールドの初めのバイトの 2 番目のビットです。次の図は、予約済みフィールド内のこのビットの位置を示しています。

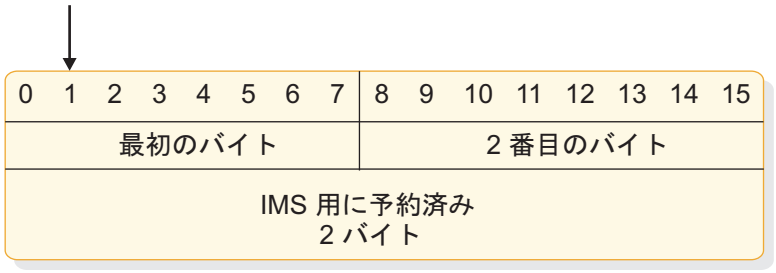


図 *78. I/O PCB* 内の宛先指示ビット

MPP1 は、このビットを検査して、メッセージがシステム A の MASTER から送られたものであるかどうかを判別します。そうであれば、MPP1 は即時に終了しなければなりません。ただし、メッセージがシステム B の MASTER から送られてきた場合は、 MPP1 で何らかのローカル処理を実行し、システム B のトランザクションをメッセージ・キューに送って、後でシステム B が再び始動したときにそれらのトランザクションを処理できるようにしておくことができます。

他の IMS TM システムの代替宛先へのメッセージの送信

出力メッセージを別の IMS TM システムの代替端末に送信するには、使用中のシステムと、メッセージの送信先となるシステムとの間に MSC リンクを設定しておく必要があります。

これを行うには、代替 PCB に対する CHNG 呼び出しを発行し、2 つの IMS TM システムを接続する MSC リンクの名前を指定します (この例では LINK1)。

例えば、別の端末からメッセージを受け取った後で、システム B の TERMINAL 1 にメッセージを送る場合は、まず次の CHNG 呼び出しを発行します。

```
CHNG altpcb, LINK1
```

その後で、ローカル端末へメッセージを送る場合とまったく同じように、ISRT 呼び出し (1 つ以上) を出してメッセージを送ります。次の表に、宛先指示出力メッセージの形式を示します。

表 *80. AIBTDLI*、*ASMTDLI*、*CBLTDLI*、*CEETDLI*、 *CTDLI*、および *PASTDLI* インターフェースの宛先指示出力メッセージの形式

フィールド名	フィールド長
LL	2
ZZ	2
DESTNAME	1 - 8
b	1
テキスト	可変

表 *81. PLITDLI* インターフェースの宛先指示出力メッセージの形式

フィールド名	フィールド長
LLLL	4
ZZ	2
DESTNAME	1 - 8
b	1
テキスト	可変

宛先指示出力メッセージのフィールドの形式は、次のとおりです。

- LL フィールドと ZZ フィールドは、それぞれ 2 バイト (ただし、PLITDLI インターフェースの場合は 4 バイトのフィールド LLLL を使用)。 LL (または LLLL) には、メッセージ全体の長さが入ります。これは、LL フィールドも含めたメッセージのすべてのフィールドの長さの合計です (PL/I では、LLLL には全長から 2 を引いた値が入ります)。ZZ は IMS 用に予約されています。
- DESTNAME に入る宛先名は、メッセージの送信先の論理端末名。このフィールドの長さは 1 から 8 バイトで、このフィールドのうしろにはブランクを 1 個入れなければなりません。

  宛先が別のシステムの宛先である場合、IMS TM はメッセージから DESTNAME 除去します。宛先が別のシステムのプログラムである場合は、IMS TM はメッセージから DESTNAME を除去しません。
- TEXT フィールドには、メッセージのテキストが入ります。長さは、送信しているメッセージにより異なります。

メッセージにセキュリティー違反がある場合、MSC が受信側システム (この場合は、システム B) でこれを検出し、発信元端末 (システム A) の使用者にこの旨を報告します。

IMS の会話型処理

IMS Transaction Manager との会話型処理を実行する、会話型プログラムを作成できます。

会話型プログラムと非会話型プログラムの相違点は、以下のとおりです。

会話型プログラム
: これは、複数のステップで構成されるトランザクションを処理するメッセージ処理プログラム (MPP) です。このトランザクション全体の処理は、一度には行われません。会話型プログラムでは、処理を、端末 - プログラム - 端末と送られる一連の関連した対話に分割します。会話型処理を使用するのは、1 つのトランザクションがいくつかの部分に分かれている場合です。

非会話型プログラム
: これは、端末からメッセージを受信し、要求を処理し、メッセージを端末へ返信するメッセージ処理プログラムです。会話型プログラムは端末からメッセージを受信し、その端末に応答を返しますが、トランザクションのデータはスクラッチパッド域 (SPA) に保管します。その後、端末使用者がさらにデータを入力すると、プログラムは最後のメッセージのデータを SPA に保管してあるので、端末使用者が同じデータを再び入力しなくても、要求を処理し続けることができます。

会話方式の例

ここでは、顧客が車のローンを組む資格があるかどうかを判断するための会話型処理の使用方法を示します。

この照会は 2 つの部分で構成されます。まず、ローンを申請した人の名前、住所、および申請人が希望するローンの年数を入力します。これらの情報を入力後、IMS TM により、車に関する情報 (モデル、年式、費用) が尋ねられます。この情報を入力すると、IMS TM がこの情報を処理するプログラムを呼び出し、呼び出されたプログラムがローンの可否を知らせます。

MFS を使用する場合には、プロセスは以下のステップで構成されます。

1. まず、形式コマンド (/FORMAT) と MOD 名を入力します。これによって、メッセージ出力記述子 (MOD) で定義されている方式に従って画面を形式設定することを IMS に指示したことになります。

   MOD 名が CL であるとすると、コマンドは次のようになります。

   ```
   /FORMAT CL
   ```

   IMS TM は、MFS ライブラリーからこの MOD を取り出し、MOD で定義されているとおりに画面を形式設定します。車のローンのアプリケーションの MOD によって画面が形式設定されると、画面は以下のようになります。

   ```
   CARLOAN
   NAME:
   ADDRESS:
   YEARS:
   ```

   「CARLOAN」というのは、このアプリケーションのトランザクション・コードです。各トランザクション・コードは、アプリケーション・プログラムと関連付けられているため、IMS TM は、「CARLOAN」というトランザクション・コードを受信した場合に、この要求に対してスケジュールすべきアプリケーション・プログラムを把握しています。
2. 顧客の名前と住所、およびローンの期間を入力します。この情報を入力すると、画面は以下のようになります。

```
CARLOAN
NAME:    JOHN EDWARDS
ADDRESS: 463 PINEWOOD
YEARS:   5
```

3. IMS TM は、トランザクション・コードの CARLOAN を読み取って、このトランザクション・コードを処理するプログラムを呼び出します。MFS は、画面に入力された情報を、MPP の入出力域に合うように DIF と MID を使用して形式設定します。

   MPP が最初の呼び出し (たいていの場合、SPA を要求する GU) を出すと、IMS TM は SPA を 2 進数のゼロで消去してから、それをアプリケーション・プログラムに渡します。

4. 次に、MPP は端末からの入力データを処理し、さらに 2 つのことを行います。それは、保管しておく必要があるデータを SPA に移すことと、端末あての出力メッセージを入出力域に作成することです。MPP が SPA に保管する情報は、要求の 2 番目の部分が端末から入力されたときに MPP にとって必要になる情報です。データベースから取り出すことのできる情報は、SPA には保管しません。この例では、ローンを申し込んでいる顧客の名前を保管します。これは、この顧客のローンが許可されたときに、更新する情報をデータベースに入れるために、プログラムが顧客の名前を使用するからです。

   この後、プログラムは、SPA を IMS に返すために ISRT 呼び出しを発行し、また出力メッセージを端末に送るために別に ISRT 呼び出しを発行します。

   MPP は、端末に送る応答の中に、次の会話サイクルのために画面を形式設定する MOD の名前を入れて、IMS TM に渡します。そのサイクルでは、John Edwards が買いたい車のモデル、年式、費用を端末から入力する必要があります。画面は以下のようになります。

   ```
   MODEL:
   YEAR:
   COST:
   ```

5. IMS TM は、トランザクション・コードと結び付けられている装置入力形式 (DIF) とメッセージ入力記述子 (MID) を再び使用して、MPP にこの情報を送り返します。 MPP は、この間ずっと実行されていません。IMS TM は、トランザクション・コードが CARLOAN である端末入力を受信すると、今回の会話サイクルでそのトランザクションを再度処理する MPP を呼び出します。

6. IMS TM は、MPP が GU を出すと、更新済みの SPA を MPP に返し、次に MPP が GN を出すと、メッセージを MPP に返します。MPP は要求された処理を行い (この場合は、ローンが許可されるかどうかを判断し、必要に応じてデータベースを更新すること)、会話を終了する準備を整えます。会話を終わらせるには、MPP は SPA 内のトランザクション・コードをブランクで消してから、IMS に SPA を挿入し、ローンが許可されるかどうかを示すメッセージを端末に送信します。

会話型プログラムの構造

会話型プログラムの構造は、プログラムと端末使用者との間でどのような対話を行うかによって決まります。

プログラムの構造を組み立てる前に、以下のことを知っておく必要があります。

- エラーが発生した場合にプログラムが行わなければならないこと

  会話中のプログラムが異常終了すると、IMS TM は会話の最後のサイクルだけをバックアウトします。会話のサイクルとは、端末とプログラムとの間の 1 つの対話単位です。会話中は、どのサイクルでも異常終了する可能性があるため、会話を簡単にリカバリーする方法を知っている必要があります。

  – 会話型プログラムでは、ROLB または ROLS 呼び出しを使用して、プログラムが最後のコミット・ポイント以降に行ったデータベース更新をバックアウトすることができます。会話型プログラムで ROLL を使用することもできますが、これは会話を終了させてしまいます。
  – できることならば、会話の結果行われるデータベース更新のレベルが異ならないようにするために、データベースの更新は、会話の最後のサイクルの一部として行ってください。
  – プログラムがエラーを検出したために終了しなければならない場合は、発信元端末、および必要ならマスター端末オペレーターにメッセージを送るために、高速代替 PCB (プログラム連絡ブロック) を使用することができます。

    そのためには、プログラムはまず、高速代替 PCB に対し CHNG 呼び出しを発行し、TP PCB に入っている論理端末名もその呼び出しで同時に指定します。次に、その高速代替 PCB と、メッセージが入っている入出力域を参照する ISRT 呼び出しを発行します。さらにプログラムは別の CHNG 呼び出しを発行して、高速代替 PCB の宛先をマスター端末あてに設定し、その PCB、ならびに出力メッセージが入っている入出力域を参照する別の ISRT 呼び出しを発行します。

- アプリケーション・プログラムは会話の各サイクルを処理するのか

  1 つの会話を 1 つ以上のアプリケーション・プログラムで処理することができます。プログラムが会話の各ステージを処理する (つまり、プログラムが端末からの各入力メッセージを処理する) のであれば、プログラムは、各入力メッセージを受け取ったときに、会話のどのステージを処理中であるかを知っていなければなりません。

  端末使用者が会話を開始させるトランザクション・コードを入力すると、IMS TM は SPA を 2 進数のゼロに消去し、プログラムが GU 呼び出しを発行したときに SPA をプログラムに渡します。ただし、以降の受け渡しでは、プログラムが会話のどのステージにいるのかがわかるようにし、その処理を行うプログラムの部分に分岐できるようにしておかなければなりません。

  プログラムが会話のどのサイクルを処理中であるかを判別するために使用できる手法の 1 つとして、SPA 内にカウンターを設けておきます。プログラムは、会話の各ステージごとに、このカウンターの値を増やします。そうしておけば、プログラムが会話の新しいサイクルを開始するたびに (SPA をリトリーブするために GU 呼び出しを発行することによって)、プログラムは SPA のカウンターを調べて、どのサイクルを処理中であるのかを突き止め、該当する部分に分岐します。

- プログラムから別の会話型プログラムに会話の制御権を渡す方法

複数のアプリケーション・プログラムを使用して 1 つの会話を処理する方が効率がよいこともあります。このようにしても、端末使用者に影響を与えることはありません。この方法をとるかどうかは必要とされる処理によって決まります。

車のローンの例で考えると、ある MPP が会話の前半 (名前、住所、ローン年数の処理) を受けもち、別の MPP がその会話の後半 (車についてのデータの処理とローンの可否の応答) を受けもつことができます。

会話型プログラムは、以下の 2 つのタイプのプログラム間通信を実行できます。

据え置きプログラム間通信
: 送信元の端末に応答を返しますが、端末からの次の入力を、別の会話型プログラムに送ることができます。

即時プログラム間通信
: 会話を別の会話型プログラムに直接渡します。このプログラムは端末に応答を返さずに、別の会話型プログラムに SPA (およびオプションでメッセージ) を渡します。この場合、発信元端末に応答を返す義務は 2 番目のプログラムにあります。

会話型プログラムは、以下のことを行わなければなりません。

1. GU 呼び出しと GN 呼び出しを用いて、SPA とメッセージをリトリーブする。

   MPP がこの会話を開始している場合は、SPA の可変域にゼロがあるかどうか検査して、これが会話の始まりであるかどうかを判別する。SPA にゼロが入っていないということは、既に会話は始まっており、現在はそれより後のステージであることを意味します。その場合には、会話のこのステージを処理するプログラム部分に分岐して、会話を続けます。

   会話を続けるために別の MPP からユーザーの MPP へ制御権が渡された場合は、メッセージを処理するのに必要なデータが SPA に入っていることがわかっているので、SPA がゼロであるかどうかをテストする必要はない。ただちにメッセージの処理を開始します。
2. メッセージを処理し、必要ならばデータベースにアクセスする。
3. I/O PCB に対して ISRT 呼び出しを発行することによって、端末に出力メッセージを送る。このステップは、ステップ 4 の後でもかまいません。
4. (ユーザー・プログラム、またはユーザーがプログラムから制御権を受け取るプログラムが、プロセスを続行するのに必要な) データを、I/O PCB への ISRT 呼び出しを用いて SPA に保管する。(このステップは、ステップ 3 の前に行ってもかまいません。) IMS TM は、下記の表にあるセグメントの ZZZZ フィールドを検査することにより、どのセグメントが SPA であるかを判別します。

会話を終了させるには、トランザクション・コードが入っている SPA の区域にブランクを入れ、ISRT 呼び出しを発行して I/O PCB を参照し、SPA を挿入して IMS TM に返します。

ユーザーの MPP が別の会話型プログラムに会話を渡す場合は、プログラムでメッセージを処理した後のステップがやや異なります。

また、I/O PCB への最初の GU 呼び出しでアプリケーション・プログラムにメッセージが返されないときに発生する状態に対応できるようなプログラムの設計が必要です。プログラムが GU 呼び出しを発行するより前に、端末使用者が /EXIT コマンドを入力して会話を取り消すと、このようなことが起こる可能性があります。(この端末からのメッセージが、プログラムのメッセージ・キューに入っている唯一のメッセージである場合に、このようなことが起こります。)

SPA の内容

GU の発行時に IMS TM からプログラムに渡される SPA には、次の表に記載されている 4 つの構成要素があります。

表 *82. AIBTDLI*、*ASMTDLI*、*CBLTDLI*、*CEETDLI*、 *CTDLI*、および *PASTDLI* インターフェースの *SPA* 形式

フィールド名	フィールド長
LL	2
ZZZZ	4
TRANCODE	8
ユーザー作業域	可変

表 *83. PLITDLI* インターフェースの *SPA* 形式

フィールド名	フィールド長
LLLL	4
ZZZZ	4
TRANCODE	8
ユーザー作業域	可変

SPA の形式のフィールドは以下のとおりです。

LL または LLLL
SPA の全体の長さを示す長さフィールド。この長さには、LL フィールド自体の 2 バイトも含まれています。(PLITDLI インターフェースの場合には、4 バイトのフィールドを使用します。そこには LLLL の長さの 4 バイトも含まれており、さらにその値から 2 バイトを引いた値が入ります。)

ZZZZ
IMS TM 用に予約されている 4 バイトのフィールドで、プログラムではここを変更してはなりません。

TRANCODE
この会話用の 8 バイトのトランザクション・コード。

ユーザー作業域
会話を続けるうえで必要な情報を保管するためにユーザーが使用する作業域。この区域の長さは、保管するデータの長さによって決まります。この長さはシステム定義時に定義します。

プログラムが会話を開始する GU を用いて SPA をリトリーブすると、IMS TM はメッセージからトランザクション・コードを取り除きます。最初のメッセージ・セグメントでは、ユーザーは、端末使用者が入力したメッセージからのデータだけを受け取ります。

以下で、アプリケーション・プログラムが SPA を処理する方法について説明します。プログラムでは、以下のことを行う必要があります。

- プログラムは、SPA の最初の 6 バイト (LL と ZZZZ) を修正してはならない。IMS TM は、これらのフィールドを使用して SPA を識別します。

  プログラムが SPA を修正した場合は、SPA を IMS TM に (プログラム間通信の場合は別のプログラムに) 返さなければならない。
- プログラムは、会話の 1 サイクル中に SPA を IMS TM に複数回返してはならない。
- プログラムは、非会話型トランザクション・コードまたは論理端末を表す代替 PCB へ SPA を挿入してはならない。プログラムが代替応答 PCB を使用できるのは、それが発信元の論理端末と同じ物理端末を表しているときだけです。

制約事項: MFS を使用している場合は、IMS TM が必ずしもトランザクション・コードを削除するとはかぎりません。

会話型のメッセージの外観

最初のセグメントには SPA が入っているので、会話型入力メッセージは少なくとも 2 つのセグメントで構成されます。入力メッセージは、2 番目のメッセージ・セグメントから始まります。

会話中の入力メッセージ・セグメントには、端末から入力されたデータだけが入ります。IMS TM は、会話の最初のステップで、入力メッセージからトランザクション・コードを除去し、それを SPA に入れます。プログラムが最初の GU を出すと、IMS TM から SPA が返されます。最初のメッセージ・セグメントをリトリーブするためには、プログラムが GN を出さなければなりません。

端末に送られる出力メッセージの形式は、非会話型プログラムにおける出力メッセージの形式と変わりません。

SPA への情報の保管

メッセージを処理し、端末へ応答を返す準備が整ったら、ユーザーは必要なデータを SPA に保管しておくことができます。データを保管しておけるのは、SPA の作業域部分です。作業域にデータを保管するには、ISRT 呼び出しを使用します。これは、ISRT 呼び出しの特殊な用法です。つまり、SPA を端末に送るわけではなく、将来使用するために SPA を保管します。

プログラムが会話の各ステージを処理する場合には、 I/O PCB への ISRT 呼び出しを発行するときに、SPA が入っている入出力域の名前を指定するだけです。以下に例を示します。

```
ISRT   I/O PCB, I/O AREA
```

これにより、更新済みの SPA が IMS TM に返されるので、会話の次のサイクルでは IMS TM からプログラムへこの SPA が渡されます。

SPA を修正しない場合には、それを IMS に返す必要はありません。しかし、会話の次のサイクルで、SPA は IMS TM によってユーザーのプログラムに渡されます。

関連概念:

『ROLB、ROLL、および ROLS を使用した会話型処理』

505 ページの『別の会話型プログラムへの会話の引き渡し』

端末への応答

会話を継続させるためには、発信元端末が各入力メッセージに対する応答を受け取る必要があります。端末使用者は、応答を端末で受け取るまでは、処理対象となるデータを端末から入力することはできません (IMS TM コマンドを除く)。

会話を継続するには、プログラムで、必要な ISRT 呼び出しを発行して、端末に出力メッセージを送信することによって、発信元端末に応答を送る必要があります。発信元端末にメッセージを送る ISRT 呼び出しでは、TP PCB か代替応答 PCB のいずれかを参照しなければなりません。応答が返される端末に 2 つの構成装置 (例えば、プリンターと穿孔装置) がついており、しかも、入力メッセージを送ってきた構成装置とは別の構成装置に出力メッセージを送りたい場合は、会話の中で代替応答 PCB を使用します。その場合、プログラムが参照する代替応答 PCB には、入力メッセージを送った論理端末と同じ物理端末が定義されていなければなりません。

プログラムは、入力メッセージ 1 つにつき 1 つの出力メッセージしか端末に送れません。出力メッセージには複数のセグメントを入れることができますが、複数の出力メッセージを送るために PURG 呼び出しを、プログラムで使用することはできません。会話型プログラムが PURG 呼び出しを発行すると、IMS TM はアプリケーション・プログラムに状況コード AZ を返し、その呼び出しを処理しません。

ROLB、ROLL、および ROLS を使用した会話型処理

会話型プログラムで ROLB または ROLS を出すと、 IMS TM は、アプリケーション・プログラムが送ったメッセージをバックアウトします。

アプリケーション・プログラムが ROLB または ROLS を発行し、発信元端末に必要な応答を送らないうちにコミット・ポイントに達すると、IMS TM は会話を終了し、メッセージ DFS2171I NO RESPONSE CONVERSATION TERMINATED を発信元端末に送ります。

会話中に ROLL を出すと、IMS TM は更新をバックアウトし、出力メッセージを取り消し、さらに会話を終了させます。

変更メッセージ・ドリブン IMS アプリケーションの会話型処理

処理に関する以下の考慮事項は、IMS ROLB 呼び出しを発行する変更メッセージ・ドリブン IMS アプリケーションに適用されます。この呼び出しでは、保護された入力メッセージを OTMA または APPC/MVS から受信し、保護されたアウトバウ

ンド作業を他の z/OS リソース・リカバリー・サービス (RRS) リソース・マネージャーに対して発行することができます。

- 保護された入力データを使用する変更メッセージ・ドリブン IMS アプリケーション・プログラムが ROLB 呼び出しを発行した場合、ROLB 呼び出しは IMS アプリケーションに隔離されるため、保護された作業単位全体に影響を及ぼすことはありません。ROLB 呼び出しが発行されると、コミット・ポイントに達するまで、保護された入力メッセージは IMS アプリケーションにとっては処理中のままになります。
- 変更メッセージ・ドリブン IMS アプリケーション・プログラムがアウトバウンド保護会話を発行した場合、そのアウトバウンド保護会話は ROLB 処理に含まれません (つまり、アウトバウンド保護会話は ROLB 呼び出しの一環としてバックアウトされません)。変更メッセージ・ドリブン IMS アプリケーション・プログラムには、バックアウト対象の保護されたすべてのアウトバウンド作業を明示的にクリーンアップする責任があります。

関連概念:

499 ページの『会話型プログラムの構造』

別の会話型プログラムへの会話の引き渡し

会話型プログラムは、据え置き通信または即時通信を実行することにより、別の会話型プログラムに会話を渡すことができます。

会話型プログラムは、次の 2 とおりの方法で別の会話型プログラムに会話を渡すことができます。

- 据え置き通信

  プログラムは端末に応答を返し、以下の方法により端末からの次の入力を別の会話型プログラムに送るようにすることができます。

  – 端末に応答を返すために、I/O PCB に対して ISRT 呼び出しを発行する。
  – 新しい会話型プログラムのトランザクション・コードを SPA に入れる。
  – SPA を IMS TM に返すために I/O PCB と SPA を指定した ISRT 呼び出しを発行する。

  この後 IMS TM は、端末からの次の入力メッセージを、SPA で指定されたトランザクション・コードと関連付けられているプログラムに送ります。他の会話型プログラムは、SPA 内のトランザクション・コードを変えることによって、プログラム間通信を続けることができます。

- 即時通信

  プログラムは、宛先が別の会話型プログラムに設定されている代替 PCB に対して ISRT 呼び出しを発行することによって、別の会話型プログラムに会話を直接渡すことができます。

  最初の ISRT 呼び出しによって SPA を別のプログラムに送らなければなりません。しかし、制御権を渡すプログラムは、新しいプログラムにメッセージを送るために ISRT 呼び出しを何回か続けて発行することができます。プログラムがこのようなことを行うと、IMS TM は、SPA を別の会話型プログラムに送る他

に、プログラムが IMS に SPA を返した場合と同じように SPA を更新します。即時通信を行ったプログラムは、SPA を IMS TM に返すことも、発信元端末に応答を返すこともできません。

会話の引き渡しについての制約事項

別の会話型プログラムへの会話の引き渡しについては、以下の制約事項が適用されます。

- 即時プログラム間通信が行われ、MPP が状況コード XE を受け取った場合、プログラムは SPA を高速代替 PCB に挿入しようとします。その PCB から EXPRESS=YES オプションを取り除くか、あるいは高速ではない別の PCB を定義して使用します。この制約事項があるのは、最初のトランザクションが SPA 挿入後に異常終了した場合、2 番目のトランザクションに会話を続行させないようにするためです。

  端末使用者は、`/SET CONV XX` コマンドを出すことができます。この XX は、会話の次のステップを処理するためにスケジュールされるプログラムです。
- APPC または OTMA 保護のトランザクションでは、即時プログラム間通信または据え置きプログラム間通信は許されません。これらの通信のいずれかが行われた場合、MPP は X6 状況コードを受け取ります。

SPA サイズの定義

SPA サイズは TRANSACT マクロで定義してください。切り捨てられたデータを取り込むオプションも TRANSACT マクロで定義されます。形式は次のとおりです。

```
TRANSACT SPA=(size,STRUNC|RTRUNC)
```

デフォルトは、切り捨てデータをサポートする (STRUNC) です。会話が最初に開始されたときおよび各プログラム間通信ごとに、切り捨てデータのオプションが検査され、指定どおりに設定または再設定されます。切り捨てデータのオプションが設定されると、会話が継続する間ずっと、またはそのオプションをリセットするよう指定しているトランザクションへのプログラム間通信が起こるまで設定されたまま残ります。

例えば、以下のように定義された 3 つのトランザクションがあるとします。

```
TRANA SPA=100
TRANB SPA=050
TRANC SPA=150
```

TRANC が TRANA からの切り捨てられたデータ (TRANB が受け取らない、TRANA からの 2 番目の 50 バイト) を受け取るには、次の指定のセットのいずれかを使用できます。

- TRANA - STRUNC または指定なし、TRANB - STRUNC または指定なし、TRANC - STRUNC または指定なし
- TRANA - RTRUNC、TRANB - STRUNC、TRANC - STRUNC または指定なし

会話型処理と MSC

インストール・システムに 2 つ以上の IMS TM システムがあり、MSC をとおしてそれらが相互に連結されている場合は、あるシステムのプログラムが、別のシステムで開始された会話を処理することができます。

- システム A の会話型プログラムが、システム B の応答代替 PCB を参照する ISRT 呼び出しを発行した場合、システム B は必要な妥当性検査を行います。これは、宛先が入力システムで暗黙に指定されているからです。システム B の検査には、応答代替 PCB に示された論理端末が入力メッセージを送った論理端末と同じ物理端末に設定されているかどうかを判別することが含まれています。同じでない場合は、システム B (会話を開始したシステム) は、アプリケーション・プログラムに状況コードを出さずに、会話を異常終了させます。
- プログラム A が、システム B の端末から開始された会話を処理するものとし、プログラム A は、SPA のトランザクション・コードを変更することにより、別の会話型プログラムにこの会話を渡すものとします。プログラム A が入れたトランザクション・コードが誤っていると、システム B (開始システム) は、アプリケーション・プログラムに状況コードを返さずに、会話を異常終了させます。

会話の終了

プログラムは会話を終了させるには、SPA のトランザクション・コードをブランクにしたうえで、ISRT 呼び出しを発行して I/O PCB と SPA を参照し、IMS TM に SPA を返します。こうすることにより、端末が応答を受信するとすぐに会話は終了します。

プログラムは、非会話型トランザクション・コードを SPA のトランザクション・フィールドに入れ、その SPA を IMS に返すことで、会話を終了させることもできます。こうすると、端末使用者が次のメッセージを入力するまで、会話はアクティブなままになっています。トランザクション・コードは、SPA から入力メッセージの最初のセグメントに挿入されます。この後 IMS TM は、端末からのこのメッセージを、SPA で指定されたトランザクション・コードを処理する MPP または BMP に送ります。

プログラムが会話を終了させることができる他に、発信元端末使用者、マスター端末オペレーター、および IMS も会話を終了させることができます。

- 発信元端末使用者は、以下のいずれかのコマンドを出すことによって、会話を終了させることができます。

  /EXIT 端末使用者は /EXIT コマンドだけを入力することも、あるいは /EXIT コマンドに続けて、IMS TM システムによって割り当てられた会話識別番号を入力することもできます。

  /HOLD

  /HOLD コマンドは、会話を一時的に停止して、会話を IMS TM が保留している間に、端末使用者が別のトランザクションを入力できるようにします。IMS TM は /HOLD コマンドへの応答として、端末使用者が後で会話を再度アクティブにするときに使用する ID を返してきます。/RELEASE コマンドにこの ID を付けて出すと、会話が再開されます。

- **/START LINE**。マスター端末オペレーターは、会話中の端末に対して /START LINE コマンド (PTERM を指定しない) または /START NODE コマンドを入力するか、あるいは会話中のサインオフされた動的ユーザーに対して /START USER コマンドを出すことにより、会話を終了させることができます。
- IMS TM が会話を終了させるのは、GU 呼び出し、または SPA を返す ISRT 呼び出しを発行することに成功したにもかかわらず、プログラムが端末に応答を送らなかった場合です。この場合は、IMS TM が端末に「DFS2171I NO RESPONSE, CONVERSATION TERMINATED」というメッセージを送ります。その後、IMS TM は会話を終了させ、アプリケーション・プログラムに対するコミット・ポイントの処理を実行します。

関連概念:

491 ページの『他の IMS アプリケーション・プログラムへのメッセージの送信』

499 ページの『会話型プログラムの構造』

関連タスク:

852 ページの『会話型 JMP アプリケーションに対する据え置きプログラム間通信』

850 ページの『JMP および JBP アプリケーションに対する即時プログラム間通信』

APPC 会話でのメッセージ通信

システム・サービス DFSAPPC を使用して、別々の LU 6.2 装置間、および LU 6.2 装置と IMS TM がサポートする別の端末との間で、メッセージの転送を行うことができます。DFSAPPC によるメッセージの送達は非同期なので、メッセージは送達できるまでは、IMS TM メッセージ・キュー上で保留状態になります。

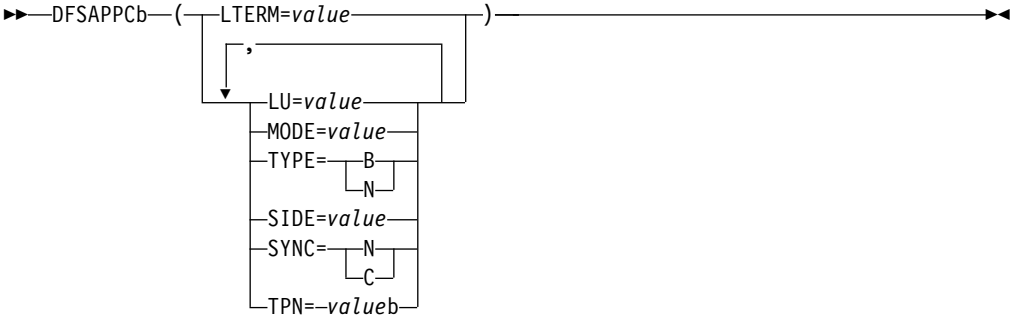
DFSAPPC でメッセージを送るには、IMS TM 端末の論理端末名、または LU 6.2 装置のトランザクション・プログラム (TP) 名を指定します。

DFSAPPC の形式

DFSAPPC のメッセージ形式は以下のとおりです。

```
DFSAPPC (options)user_data
```

DFSAPPC は、以下のようにコーディングすることができます。



DFSAPPC と指定オプションの間には、1 つのブランク (b) が必要です。

ブランクは指定するオプションの中では有効ですが、キーワードまたは値の中では無効になります。オプションとオプションとの間には、区切り文字としてコンマとブランクのいずれでも使用できますが、TP 名ではコンマを使用できるので、うしろに少なくとも 1 つのブランクが必要です。

LU 6.2 の会話が他のソース (例えば、CPI-C ドリブン・アプリケーション・プログラム中) から開始されたものでない場合、パートナー LU 6.2 装置との会話を開始するために DFSAPPC が使用されます。 DFSAPPC でオプションを指定しないと、IMS TM のデフォルト・オプションが使用されます。

オプション・キーワード

LTERM=

IMS TM 論理端末の LTERM 名を指定します。LTERM 名には、最大 8 文字の英数字または国別文字 (@、$、#) を入れることができます。LTERM を指定すると、他のオプション・キーワードを指定することはできません。

LU=

LU 6.2 会話のパートナーの LU 名を指定します。 LU 名には最大 8 文字の英数字または国別文字を入れることができますが、最初の文字は英字または国別文字でなければなりません。 LU と SIDE の両オプションを指定すると、LU は、サイド情報項目に入っている LU 名を上書きしますが、LU 名を変更することはありません。

LU 名がネットワーク修飾名の場合、名前は最高 17 文字の長さで、発信元システムのネットワーク ID の後に '.' が付き、その後に LU 名が続いた形になります (例えば、netwrkid.luname)。LU 名とネットワーク ID はどちらも 1 から 8 文字の長さです。

MODE=

LU 6.2 会話のパートナーの MODE 名を指定します。 MODE 名には最大で 8 文字の英数字または国別文字を入れることができますが、最初の文字は英字または国別文字でなければなりません。 MODE と SIDE の両オプション・キーワードを指定すると、MODE は、サイド情報項目に入っている MODE 名を上書きしますが、その名前を変更することはありません。

TPN=

LU 6.2 会話でのパートナーのトランザクション・プログラム (TP) 名を指定します。TP 名には、00640 文字セットから最大 64 文字を入れることができます。文字セットにはコンマを使用することができるので、TP 名のあとには少なくとも 1 つのブランクが必要です。TPN と SIDE の両オプション・キーワードを指定すると、TPN は、サイド情報項目に入っている TP 名を上書きしますが、その名前を変更することはありません。

関連資料:「*CPI Communications Specification*」では、すべての英数字と国別文字、および 20 個の特殊文字を含む 00640 文字セットについて説明しています。

SIDE=

LU 6.2 会話でのパートナーのサイド情報項目の名前を指定します。サイド情報項目名には、01134 文字セットから最大 8 文字を入れることができます。SIDE オプション・キーワードを指定すれば、LU、MODE、および TPN の各オプション・キーワードで上書きすることができます。

関連資料:「*CPI Communications Specification*」では、英大文字と 0 から 9 の数字を含む 01134 文字セットについて説明しています。

SYNC=N|C
LU 6.2 会話の同期レベルを指定します。同期レベルは、N を選択するとなしに、C を選択すると確認になります。

TYPE=B|M
LU 6.2 会話の会話タイプを指定します。 B を選択すると基本的会話タイプに、M を選択するとマップ式会話タイプになります。

APPC による会話処理

APPC/IMS は、標準の、修正された CPI 通信ドリブン・アプリケーション・プログラムをサポートします。

APPC/IMS では、3 種類のアプリケーション・プログラムがサポートされています。

- 標準: CPI 通信機能を明示的に使用しない。
- 修正済み: I/O PCB を使用して元の入力端末と通信する。CPI 通信呼び出しを使用して、新しい会話を割り振り、データの送受信を行う。
- CPI 通信ドリブン: CPI 通信呼び出しを使用して、着信メッセージを受け取り、同じ会話で返信を送信する。DL/I APSB 呼び出しを使用して、IMS データベースと代替 PCB にアクセスするための PSB を割り振る。

修正済みまたは CPI 通信ドリブン・アプリケーション・プログラムでは、APPC 会話が SYNCLVL=SYNCPT を指定して割り振られていると、z/OS が、APPC 会話の参加者、すなわちアプリケーション・プログラムと IMS の同期点プロセスを管理します。トランザクションのロールバックおよびスケジュール変更が可能です。これは、修正済み IMS APPC APPC アプリケーション・プログラムのために、IMS が `SRRCMIT` または `SRRBACK` 呼び出しを発行するからです。CPI-C ドリブン・プログラムが、以前のリリースで要求されていたように IMS スタブ・コード (DFSCPIR0) とリンクされていると、IMS は SRRCMIT または SRRBACK 呼び出しも発行します。プログラムがスタブ・コードとリンクされていないと、アプリケーションがこれらの呼び出しを発行したとき、IMS は z/OS 同期点マネージャーにより駆動されます。z/OS を同期点マネージャーとして使用した場合は、失敗もバックアウトできます。

標準および修正済みのアプリケーション・プログラムは、MSC または APPC/MVS を使用すると、ローカルでもリモートでもスケジュールすることができます。ローカル・スケジューリングとリモート・スケジューリングでは、論理フローが異なります。

APPC/MVS 会話が SYNCLVL=SYNCPT を指定されていると、プログラムを MSC 経由でリモートでスケジュールすることはサポートされません。

APPC 会話の終了

LU 6.2 装置を使用する会話は、CPI-C verb である DEALLOCATE を出すか、IMS 会話型トランザクションの場合は SPA にブランクのトランザクション・コードを挿入することで終了できます。

制約事項: LU 6.2 会話の場合、/EXIT コマンドは使用できません。

LU 6.2 会話の終了時には、以下のようなエラー条件が発生する場合があります。

- アプリケーション・プログラムで、会話を割り振り解除する直前に LU 6.2 装置にデータを送ると、IMS TM は、エラー・メッセージ DFS1966 の SENDERROR および SENDDATA を出します。これは、トランザクションが終了したが、最後のメッセージを送達できなかったことを示します。SENDERROR を活動化するには、同期レベルに CONFIRM を指定します。
- IMS TM が IMS TM 会話型トランザクションから LU 6.2 装置に出力を送るときにエラーを検出した場合、その出力は廃棄され、会話は IMS TM と LU 6.2 の両方について終了されます。
- LU 6.2 会話中に IMS TM 会話型アプリケーション・プログラムが異常終了すると、エラー・メッセージ DFS555 が発信元の LU 6.2 装置に送られ、会話は IMS TM と LU 6.2 の両方について終了されます。

会話型プログラムのコーディング

会話型プログラムをコーディングする前に、以下の情報を入手する必要があります。

- 制御権を渡す先のプログラムに使用するトランザクション・コード
- SPA に保管する必要のあるデータ
- データの最大の長さ

SPA には以下の 4 つのフィールドが入ります。

- 2 バイトの長さフィールド
- IMS TM 用に予約済みの 4 バイトのフィールド
- 8 バイトのトランザクション・コード
- 会話データを保管する作業域。このフィールドの長さは、システム定義時に定義します。

標準 IMS アプリケーション・プログラム

標準 IMS アプリケーション・プログラムは、既存の IMS 呼び出しインターフェースを使用します。 IMS 標準の API を使用するアプリケーション・プログラムは、LU 6.2 プロトコルを利用することができます。

標準 IMS アプリケーション・プログラムは、DL/I の GU 呼び出しを使用して着信トランザクションを取り出します。また、これらの標準 IMS アプリケーション・プログラムは、LU 6.2 が使用されているかどうかに関わらず、DL/I の ISRT 呼び出しを使用して、同じ端末または異なる端末への出力メッセージを生成します。LU 6.2 端末でも LU 6.2 以外のタイプの端末でも、同じプログラムが正常に動作します。IMS は、APPC/MVS サービスに対する適切な呼び出しを生成します。

非メッセージ・ドリブン BMP は、明示的な API を使用しない場合、標準 IMS アプリケーション・プログラムと見なされます。

拡張プログラム間通信機能 (APPC) アプリケーション・プログラムが、リモート IMS で実行される IMS トランザクションに入ると、APPC アプリケーション・プログラムとローカル IMS システムの間で LU 6.2 の会話が確立されます。ローカ

ル IMS は、LU 6.2 の会話のパートナー LU と見なされます。その後、トランザクションはローカル IMS システムのリモート・トランザクション・キューに入れられます。この時点で、トランザクションは正規の MSC 処理を終了します。リモート IMS システムでトランザクションが実行されると、ローカル IMS システムに出力が返され、その後で発信元の LU 6.2 アプリケーション・プログラムに送られます。

修正済みの IMS アプリケーション・プログラム

修正済みの IMS アプリケーション・プログラムは、DL/I の GU 呼び出しを使用して着信トランザクションを取り出します。これらの修正済みの IMS アプリケーション・プログラムも、LU 6.2 が使用されているかどうかにかかわらず、DL/I の ISRT 呼び出しを使用して同じ端末または異なる端末への出力メッセージを生成します。

非メッセージ・ドリブン BMP は、明示的な API を使用する場合、修正済みの標準 IMS アプリケーション・プログラムと見なされます。標準 IMS アプリケーション・プログラムとは異なり、修正済み IMS アプリケーション・プログラムは CPI 通信呼び出しを使用して新しい会話を割り振り、データの送受信を行います。IMS は、 CPI 通信会話を直接制御しません。

修正済みの IMS トランザクションは、プログラムの実行まで、標準 IMS トランザクションと区別できません。実際、同じアプリケーション・プログラムが実行時によって標準 IMS アプリケーション・プログラムとなったり、修正済み IMS アプリケーション・プログラムとなったりします。このときの区別は、アプリケーション・プログラムが CPI 通信リソースを使ったかどうかという点だけです。

修正済みの IMS プログラムは IMS TM によってスケジュールされ、DL/I 呼び出しは DL/I 言語インターフェースによって処理されます。ただし、会話は APPC/MVS によって維持され、APPC/MVS についての障害が IMS TM によってバックアウトされることはありません。修正済みの IMS アプリケーション・プログラムの一般的な形式を、次のコードの例に示します。

```
• GU     IOPCB
    ALLOCATE
    SEND
    RECEIVE
    DEALLOCATE
• ISRT   IOPCB
```

図 *79.* 修正済み *DL/I* アプリケーション・プログラムの一般的形式

制約事項**:** APPC 会話は同期点を超えることができません。会話が同期点に達する前に割り振り解除されないと、IMS は TP 呼び出し (ATBCMTP) を発行して、会話を終了させます。新規の APPC 会話は、各同期点の後で割り振ることが可能です。

APPC プログラムが、リモート IMS システムで実行される IMS トランザクションに入ると、APPC プログラムとローカル IMS システムの間で LU 6.2 の会話が開設されます。ローカル IMS システムは、LU 6.2 の会話のパートナー LU と見なされます。その後、トランザクションはローカル IMS システムのリモート・ト

ランザクション・キューに入れられます。この時点で、トランザクションは正規の MSC 処理を終了します。リモート IMS でトランザクションが実行されると、ローカル IMS システムに出力が返され、その後で発信元の LU 6.2 プログラムに送られます。

関連資料: 障害リカバリーおよび修正済みの DL/I アプリケーション・プログラムの設計について詳しくは、「*IMS V15* アプリケーション・プログラミング *API*」を参照してください。

CPI-C ドリブン・アプリケーション・プログラム

CPI 通信ドリブン・アプリケーション・プログラムは、APPC/MVS の TP_Profile データ・セットだけに定義され、IMS に対しては定義されません。IMS の再始動の後、APPC/MVS の TP_Profile 定義に基づいて、APPC/MVS によるスケジューリングを行うためトランザクションが渡されている場合は、この定義は IMS によって動的に作成されます。この定義のキーとして使用されるのは TP 名です。APPC/MVS は TP_Profile 情報を管理します。

CPI 通信ドリブンのトランザクション・プログラムが PSB を要求する場合、その PSB は、システム定義では APPLCTN マクロによって、また、APPLCTN PSB= が指定されるときは PSBGEN または ACBGEN によって IMS に定義されている必要があります。 APPLCTN GPSB= を指定する場合は、PSBGEN または ACBGEN は必要ありません。

CPI-C ドリブン・アプリケーション・プログラムは、CPI-C verb の ACCEPT および RECEIVE で始まり、LU 6.2 会話を開始しなければなりません。そうすると、APSB 呼び出しを発行して、アプリケーション・プログラムで使用するために PSB を割り振ることができます。 APSB 呼び出しを発行すると、割り振られた PCB を使用して追加の DL/I 呼び出しを発行することができます。その後で、SRRCMIT verb を発行して変更をコミットするか、または SRRBACK verb を発行して変更をバックアウトします。SRRCMIT および SRRBACK を使用するには、アプリケーション・プログラムは DFSCPIR0 とリンクされていなければなりません。

制約事項: I/O PCB は、CPI-C ドリブン・アプリケーション・プログラムによるメッセージ処理呼び出しに使用することはできません。 CPI に固有の制約事項については、各呼び出しの説明を参照してください。

使用中の PSB を割り振り解除するには、DPSB 呼び出しを発行します。そうすると、別の APSB 呼び出しを発行するか、または CPI-C verb の DEALLOCATE を使用して会話を終了させることができます。

CPI-C ドリブン・アプリケーション・プログラムは、(SYNCLVL=SYNCPT を指定して割り振られていない限り) IMS TM によって廃棄可能と扱われます。したがって、システム障害時に自動的な回復は行われません。SYNCLVL=SYNCPT を指定して割り振られている場合は、どのような障害からも回復するため、2 フェーズ・コミット・プロセスが使われます。 CPI-C ドリブン・アプリケーション・プログラムの一般的形式を、次のコードの例に示します。

- ACCEPT
- RECEIVE
  - APSB
    GU DBPCB
    REPL DBPCB
    SRRCMIT
  - DPSB
- DEALLOCATE

図 *80. CPI-C* ドリブン・アプリケーション・プログラムの一般的形式

関連概念:

CPI-C ドリブン・アプリケーション・プログラム (コミュニケーションおよびコネクション)

OTMA による会話処理

OTMA を通じて、IMS 会話型トランザクションを実行できます。

「*IMS V15* コミュニケーションおよびコネクション」を参照してください。

前のコミット・ポイントへのバックアウト: ROLL、ROLB、および ROLS 呼び出し

プログラムが処理の一部を無効と判断した場合、ロールバック呼び出し ROLL、データベース PCB を使用する ROLS、入出力域またはトークンを指定しない ROLS、および ROLB の各ロールバック呼び出しを使用して正しくない処理の影響を取り除くことができます。

これらの呼び出しのいずれかを出すと、IMS は以下のことを行います。

- プログラムの最後のコミット・ポイント以降に行われたデータベース更新をバックアウトする。
- プログラムの最後のコミット・ポイント以降に作成された非高速出力メッセージを取り消す。

これらの呼び出しの主な違いは、ROLB は更新のバックアウトと出力メッセージの取り消し後、アプリケーション・プログラムに制御権を返し、ROLS はアプリケーション・プログラムに制御権を返さず、ROLL はユーザー異常終了コード 0778 とともに終了するという点です。 ROLB は最後のコミット・ポイント以降の最初のメッセージ・セグメントをプログラムに返すことができますが、ROLL および ROLS ではできません。

ROLL 呼び出しおよび ROLB 呼び出し、ならびにトークンを指定されていない ROLS 呼び出しは、PSB に汎用順次アクセス方式 (GSAM) データ・セットのための PCB が入っている場合に有効です。ただし、最後のコミット・ポイント以降に GSAM データ・セットに挿入されたセグメントは、これらの呼び出しではバックアウトされません。拡張チェックポイント再始動を使用して、再始動時に GSAM データ・セットを位置変更することができます。

ROLS 呼び出しを使用して、前のコミット・ポイントにバックアウトするか、あるいは前の SETS 呼び出しで設定した中間バックアウト・ポイントにバックアウトすることができます。このセクションでは、前のコミット・ポイントにバックアウトする ROLS の形式についてのみ説明します。

関連概念:

331 ページの『中間バックアウト・ポイントへのバックアウト: SETS、SETU および ROLS』

ROLB、ROLL、および ROLS の比較

以下の表では、ROLB、ROLL、および ROLS の呼び出しを比較しています。

表 *84. ROLB*、*ROLL*、および *ROLS* の比較 :

取られるアクション	ROLB	ROLL	ROLS
最後のコミット・ポイント以降のデータベース更新をバックアウトする。	X	X	X
最後のコミット・ポイント以降に作成された出力メッセージを取り消す。	X[1]	X[1]	X[1]
処理中のメッセージをキューから削除する。最後のコミット・ポイント以降に処理された以前のメッセージがある場合は、それを再処理のためにキューに戻す。		X	
最後のコミット・ポイント以降の最初の入力メッセージの最初のセグメントを返す。	X[2]		
3303 異常終了となり、処理済みの入力メッセージがメッセージ・キューに返される。			X[3]
0778 異常終了 (ダンプなし)		X	
異常終了しない (プログラムは処理を続行)。	X		

注:

1. ROLB、ROLL、または ROLS は、プログラムで PURG を出さないかぎり、高速 PCB で送られた出力メッセージを取り消します。

   例えば、プログラムが、以下のような呼び出しシーケンスを出した場合、MSG1 は宛先に送られます。これは、PURG により MSG1 が完了していて入出力域に次のメッセージ (この例では MSG2) の最初のセグメントが入っていることが IMS に通知されるためです。ただし、MSG2 は取り消されます。

   ```
   ISRT    EXPRESS PCB, MSG1
   PURG    EXPRESS PCB, MSG2
   ROLB    I/O PCB
   ```

   IMS に完全なメッセージ (MSG1) があるため、さらに高速 PCB を使用しているために、コミット・ポイントの前にメッセージを送ることができます。
2. 入出力域のアドレスを、呼び出しパラメーターの 1 つとして指定した場合のみ返されます。
3. トランザクションは延期され、以降の処理のために再キューイングされます。

ROLL

ROLL 呼び出しは、データベースの更新をバックアウトし、プログラムが最後のコミット・ポイント以降に作成した非高速出力メッセージをすべて取り消します。また、現行の入力メッセージも削除します。最後のコミット・ポイント以降に処理されたその他すべての入力メッセージは、キューに返されて再処理されます。IMS はユーザー異常終了コード 0778 とともにプログラムを終了します。このタイプの異常終了は、ストレージ・ダンプなしでプログラムを終了します。

ROLL 呼び出しを発行するときに指定するパラメーターは、呼び出し機能の ROLL だけです。

バッチ・プログラムでは、ROLL 呼び出しを使用することができます。システム・ログが直接アクセス・ストレージにある場合、および BKO 実行パラメーターの使用を通じて動的バックアウトが指定されている場合には、最後のコミット・ポイント以降のデータベース更新はバックアウトされます。それ以外の場合はバックアウトされません。バッチ・プログラムで ROLL を出す理由の 1 つとして、互換性の維持があります。

バックアウトが完了すると、元のトランザクションは廃棄可能であれば廃棄され、再実行されません。IMS は、TPI を指定する APPC/MVS verb ATBCMTP TYPE (ABEND) を出してリモート・トランザクション・プログラムに通知します。APPC/MVS verb を出すと、アクティブなすべての会話 (アプリケーション・プログラムにより作成されたすべての会話を含む) が DEALLOCATED TYP (ABEND_SVC) になります。

ROLB

ROLB 使用の利点は、ROLB の実行後、IMS がプログラムに制御権を返すので、プログラムが処理を続行できる点にあります。

ROL のパラメーターは以下のとおりです。

- 呼び出し機能 ROLB
- I/O PCB または AIB の名前

ROLB 呼び出しの総合的な効果は、それを出した IMS アプリケーションのタイプによって異なります。

- 現行 IMS アプリケーション・プログラムの場合:

  IMS バックアウトが完了すると、元のトランザクションが IMS アプリケーション・プログラムに返されます。IMS がロールバックすることができないリソースはすべて無視されます。例えば、高速代替 PCB に送られた出力、および PURG 呼び出しが ROLB の前に出されます。
- 修正 IMS アプリケーション・プログラムの場合:

  現行の IMS アプリケーション・プログラムの場合と同じ考慮事項があります。作成されたすべての会話に ROLB が出されたことを通知するのは、アプリケーション・プログラムの責任です。
- CPI-C ドリブン IMS アプリケーション・プログラムの場合:

IMS リソースのみが影響を受けます。すべてのデータベース変更がバックアウトされます。高速代替 PCB 以外の代替 PCB に挿入されたすべてのメッセージが廃棄されます。また、PURGE 呼び出しが出されていない高速 PCB に挿入されたすべてのメッセージも廃棄されます。ROLB 呼び出しが出されたことを、発信元のリモート・プログラムおよび作成されたすべての会話に通知するのは、アプリケーション・プログラムの責任です。

MPP およびトランザクション指向 BMP の場合

プログラムが ROLB パラメーターの 1 つとして入出力域のアドレスに指定すると、ROLB 呼び出しはメッセージ・リトリーブ呼び出しとして機能し、最後のコミット・ポイント以降の最初の入力メッセージの最初のセグメントを返します。これは、最後のコミット・ポイント以降にプログラムがメッセージ・キューに対して GU 呼び出しを発行した場合にだけ当てはまります。GU 呼び出しを発行しなかった場合には、ROLB 呼び出しを発行したときにプログラムはメッセージを処理していなかったことになります。

ROLB を出した後にプログラムが GN をメッセージ・キューに発行すると、 IMS は、ROLB が発行されたときに処理していたメッセージの次のセグメントを返します。メッセージにこれ以上セグメントがない場合には、IMS は状況コード QD を返します。

ROLB 呼び出しの後にプログラムが GU をメッセージ・キューに出すと、 IMS は、次のメッセージの最初のセグメントをアプリケーション・プログラムに返します。プログラムのメッセージ・キュー上にこれ以上処理すべきメッセージがない場合、IMS は QC 状況コードをプログラムに返します。

入出力域をパラメーターに含めていても、最後のコミット・ポイント以降に GU 呼び出しをメッセージ・キューに適切に発行できなかった場合は、IMS は、QE 状況コードをプログラムに返します。

ROLB 呼び出しに入出力域のアドレスを指定しない場合には、IMS の処理は同じです。プログラムがコミット処理中に GU を発行してこれが正常に実行された後、GN を発行すると、IMS は QD 状況コードを返します。プログラムが ROLB の後に GU を出すと、IMS は、次のメッセージの最初のセグメントを、またはプログラムにこれ以上メッセージがない場合には QC 状況コードを返します。

最後のコミット・ポイント以降に GU を出して成功しておらず、ROLB 呼び出しに入出力域パラメーターを指定しない場合には、IMS は、データベース更新をバックアウトし、最後のコミット・ポイント以降に作成された出力メッセージを取り消します。

バッチ・プログラムの場合

システム・ログが直接アクセス・ストレージにある場合、および BKO 実行パラメーターの使用を通じて動的バックアウトが指定されている場合には、バッチ・プログラムでは ROLB 呼び出しを使用することができます。ROLB 呼び出しは、メッセージ処理プログラム (MPP) のようにはメッセージを処理しません。これは、最後のコミット・ポイント以降のデータベース更新をバックアウトし、制御権をプログラムに返します。その呼び出しでパラメーターの 1 つを入出力域のアドレスに指定することはできません。指定すると、状況コード AD がユーザー・プログラムに返さ

れます。ただし、プログラムに I/O PCB を指定しなければなりません。プログラムの PSB の PSBGEN ステートメントの CMPAT キーワードに CMPAT=YES を指定します。

関連資料: CMPAT キーワードの使用の詳細については、「*IMS V15* システム・ユーティリティー」を参照してください。ROLB 呼び出しのコーディングについては、「*IMS V15* アプリケーション・プログラミング *API*」のトピック『ROLB 呼び出し』を参照してください。

ROLS

ROLS 呼び出しを使用して前のコミット・ポイントにバックアウトし、処理済みの入力メッセージを後で再処理するために IMS に返すことができます。

プログラムでは、次のいずれかを行うことができます。

- I/O PCB を使用して、ROLS 呼び出しを発行する。ただし、呼び出しに入出力域とトークンは指定しません。この形式の ROLS 呼び出しのパラメーターは以下のようになります。
  - 呼び出し機能 ROLS
  - I/O PCB または AIB の名前
- データを使用できない状況コードの 1 つを受け取ったデータベース PCB を使用して、ROLS 呼び出しを発行する。この場合は、使用不能なデータが検出された場合、および INIT 呼び出しが発行されなかった場合と同じ結果になります。ROLS は、必ず PCB の次の呼び出しになります。他の PCB を使用して呼び出しに介入することもできます。

トークンを指定した ROLS では、IMS によって処理されたすべてのメッセージを含む、すべての非高速メッセージについて、メッセージ・キューの位置変更が行われる場合があります。APPC/MVS を使用するこの処理は、初期のメッセージ・セグメントを呼び出して組み込みます。元の入力トランザクションは、IMS アプリケーション・プログラムに返すことができます。入出力の位置付けは、SETS 呼び出しで決定されます。この位置付けは、現行および変更 IMS アプリケーション・プログラムに適用され、CPI-C ドリブンの IMS プログラムには適用されません。IMS アプリケーション・プログラムは、すべてのリモート・トランザクション・プログラムに、ROLS について通知しなければなりません。

トークンなしの ROLS では、IMS は、トランザクション・プログラム・インターフェース (TPI) を指定する APPC/MVS verb の ATBCMTP TYPE (ABEND) を出します。この verb を出すと、アプリケーション・プログラムに関連したすべての会話が DEALLOCATED TYPE(ABEND_SVC) になります。元のトランザクションが LU 6.2 装置から入力され、IMS が APPC/MVS からメッセージを受け取った場合、廃棄可能なトランザクションは廃棄され、廃棄不能トランザクションなどの延期キューに入れられることはありません。

関連資料: LU 6.2 の詳細については、「*IMS V15* コミュニケーションおよびコネクション」を参照してください。

この形式の ROLS 呼び出しのパラメーターは以下のようになります。

- 呼び出し機能 ROLS

- 状況コード BA または BB を受け取った DB PCB の名前

どちらの用法で ROLS を使用した場合も、ROLS 呼び出しを発行すると、3303 異常終了となり、制御権はアプリケーション・プログラムに返されません。IMS は、今後の処理のために入力メッセージを保管します。

中間バックアウト・ポイントへのバックアウト: SETS/SETU および ROLS

ROLS 呼び出しを使用して、前の SETS 呼び出しまたは SETU 呼び出しで設定された中間バックアウト・ポイントにバックアウトするか、あるいは前のコミット・ポイントにバックアウトすることができます。

このセクションでは、中間バックアウト・ポイントにバックアウトする ROLS の形式についてのみ説明します。 ROLS のその他の形式については、『前のコミット・ポイントへのバックアウト: ROLL、ROLB、および ROLS 呼び出し』を参照してください。

中間ポイントにバックアウトする ROLS 呼び出しがバックアウトするのは、DL/I の変更だけです。この種の ROLS 呼び出しは、CICS ファイル制御または CICS 一時データを使用する CICS の変更には影響しません。

SETS 呼び出しおよび ROLS 呼び出しは、アプリケーション・プログラムの呼び出し処理に中間バックアウト・ポイントを設定し、これらのいずれかのポイントにデータベース変更をバックアウトします。最大 9 つの中間バックアウト・ポイントを設定することができます。SETS 呼び出しは、各ポイントにトークンを指定します。そして、IMS はこのトークンを現行の処理ポイントに関連付けます。次に続く ROLS 呼び出しは同じトークンを使用し、すべてのデータベース変更をバックアウトし、同じトークンの SETS 呼び出しに続いて実行されたすべての非高速メッセージを廃棄します。次の図は、SETS 呼び出しと ROLS 呼び出しが連携して機能する仕組みを表しています。

さらに、ROLS 呼び出しに続くその他の変数の再確立でアプリケーション・プログラムを援助するために、ユーザー・データを SETS 呼び出しの入出力域に組み込むことができます。そうすると、このデータは、同じトークンの ROLS 呼び出しが出されたときに返されます。

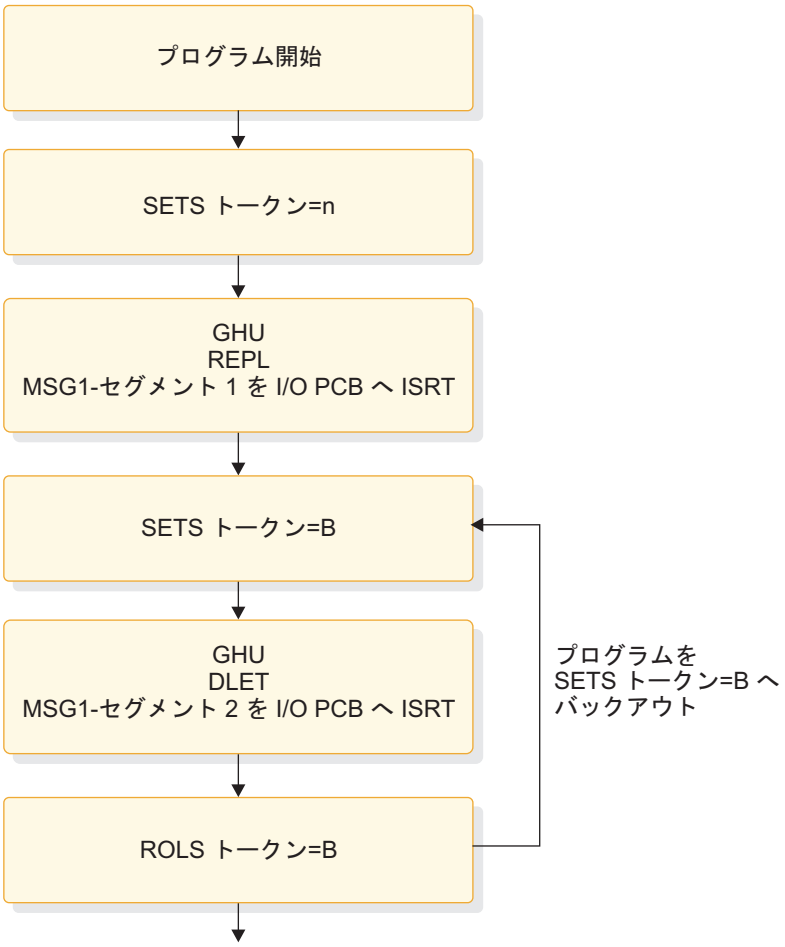


図 *81. SETS* 呼び出しと *ROLS* 呼び出しの連係動作

SETS/SETU

SETS 呼び出しは、9 個までの中間バックアウト・ポイントを設定するか、または既存のすべてのバックアウト・ポイントを取り消します。 SETS 呼び出しを使用することにより、作業の各部分をバックアウトすることができます。作業の一部を完了するために必要なデータが使用不能である場合には、作業の別の部分を完了してから前の部分に戻ることができます。

中間バックアウト・ポイントを設定するには、I/O PCB を使用する呼び出しを発行し、入出力域およびトークンを指定します。入出力域の形式は LLZZ ユーザー・データとなります。ここで、LL は LLZZ の部分の長さを含む、入出力域内のデータの長さです。ZZ フィールドには、常に 2 進数のゼロが入ります。入出力域内のデータは、関連する ROLS 呼び出しでアプリケーション・プログラムに返されます。ROLS 呼び出しで返されるデータの中に保管したくないものがある場合は、入出力域の長さを定義する LL を 4 に設定しなければなりません。

PLITDLI の場合には、LL フィールドは、他の言語の場合のようなハーフワードではなく、フルワードで定義しなければなりません。PLITDLI の場合の LL フィールドの内容は、LLZZ の形式を使用する他の呼び出しの入出力域と整合します。つまり、その内容は、4 バイトの LL フィールドから 2 を引いた長さを含む区域の全体の長さです。

現行の処理ポイントに関連する 4 バイトのトークンも必要です。このトークンは、このプログラム実行のための新しいトークンを指定するか、または先行する SETS 呼び出しで出されたトークンと一致します。トークンが新しい場合は、先行する SETS 呼び出しは取り消されません。トークンが先行する SETS 呼び出しのトークンと一致する場合、現行の SETS 呼び出しは先行する呼び出しの位置を使用します。この場合、一致するトークンを持つ SETS 呼び出し以降に出されたすべての SETS 呼び出しが取り消されます。

この形式の SETS 呼び出しのパラメーターは、以下のようになります。

- 呼び出し機能 SETS
- I/O PCB または AIB の名前
- ユーザー・データが入っている入出力域の名前
- トークンが入っている区域の名前

SETS 呼び出しの形式については、「*IMS V15* アプリケーション・プログラミング *API*」のトピック『SETS/SETU 呼び出し』を参照してください。

前のバックアウト・ポイントをすべて取り消すときは、I/O PCB を使用して呼び出しを発行しますが、入出力域またはトークンは指定しません。呼び出しに入出力域を指定しない場合には、前の SETS 呼び出しで設定されたすべての中間バックアウト・ポイントが取り消されます。

この形式の SETS 呼び出しのパラメーターは、以下のようになります。

- 呼び出し機能 SETS
- I/O PCB または AIB の名前

コミットされたデータをバックアウトすることはできないので、コミット・ポイントを処理すると、未処理のすべての SETS が取り消されます。

DEDB、MSDB、および GSAM の各編成の PCB が PSB にある場合、または接続されたサブシステムにプログラムがアクセスする場合には、部分的なバックアウトはできません。この場合、SETS 呼び出しは拒否され、SC 状況コードが戻されます。代わりに SETU 呼び出しを使用すれば、サポートされない PCB が原因で拒否されることはありませんが、警告として状況コード SC が返されます。この警告は、サポートされない PCB が PSB に入っていること、およびその機能がこれらのサポートされない PCB に適用できないことを知らせるものです。

関連資料: SETS 呼び出しの後で返される状況コード、およびそれらの状況コードと必要な対応に関する説明については、「*IMS V15* アプリケーション・プログラミング *API*」を参照してください。

ROLS

ROLS 呼び出しは、前の SETS または SETU 呼び出しで設定された処理ポイントまで、または前のコミット・ポイントまでデータベースの変更をバックアウトし、処理済みの入力メッセージをメッセージ・キューに返します。

前の SETS 呼び出し以降に行われたデータベース変更およびメッセージ・アクティビティーをバックアウトするには、I/O PCB を使用し、呼び出しに入出力域およびトークンを指定する ROLS 呼び出しを発行します。トークンが先行する SETS 呼び

出しで設定されたトークンと一致しない場合には、エラー状況が返されます。トークンが先行する SETS 呼び出しのトークンと一致する場合には、この対応する SETS 呼び出し以降に行われたデータベース更新がバックアウトされ、対応する SETS 呼び出し以降に挿入されたすべての非高速メッセージが廃棄されます。ROLS 呼び出しは、呼び出しが処理されるとブランクを返し、エラーまたは警告が発生すると状況コードを返します。SETU を ROLS と併用し、外部サブシステムがある場合には、ROLS 呼び出しは拒否されず、状況コード RC が警告として返されます。バックアウトされた処理の一部として出されたすべての SETS ポイントは取り消され、サポートされるすべての PCB の既存のデータベースの位置はリセットされます。ROLS 呼び出しの形式については、「*IMS V15* アプリケーション・プログラミング *API*」のトピック『ROLB 呼び出し』を参照してください。

この形式の ROLS 呼び出しのパラメーターは以下のようになります。

- 呼び出し機能 ROLS
- I/O PCB または AIB の名前
- ユーザー・データを受け取る入出力域の名前
- 4 バイトのトークンが入っている区域の名前

関連資料: ROLS 呼び出しの後に返される状況コード、およびそれらの状況コードと必要な対応に関する説明については、「*IMS V15* メッセージおよびコード 第 *4* 巻 *: IMS* コンポーネント・コード」を参照してください。

メッセージ・ドリブン・プログラムの作成

メッセージ・ドリブン・プログラムは MPP と似ています。つまりメッセージをリトリーブして処理する他に、MSDB、DEDB、および全機能データベースの読み取りと更新を行うことができます。

メッセージ・ドリブン・プログラムは、以下の宛先にメッセージを送信することができます。

- 入力メッセージを送った論理端末。この場合、プログラムでは、I/O PCB を参照する ISRT 呼び出しを発行します。
- 入力メッセージを送った物理端末の別の構成装置。この場合、プログラムでは、代替応答 PCB を参照する ISRT 呼び出しを発行します。
- 入力メッセージを送った物理端末とは別の物理端末。この場合、プログラムでは、代替 PCB を参照する ISRT 呼び出しを発行します。

メッセージ・ドリブン・プログラムに使用できるメッセージ処理機能にはいくつかの制約事項があります。これらの制約事項は、I/O PCB によって送受信されたメッセージについてのみ適用されます。メッセージ・ドリブン・プログラムの入力メッセージは、単一セグメントのメッセージでなければなりません。したがって、入力メッセージを入手するために使用できる呼び出しは GU だけです。また、I/O PCB によって送られる応答メッセージも、単一セグメントのメッセージでなければなりません。

トランザクションは応答モードになっています。これは、次のメッセージを送るためには応答を返さなければならないことを意味します。SPA を使用することはできません。これは、メッセージ・ドリブン・プログラムを会話型プログラムにすることはできないためです。

すべてのシステム・サービス呼び出しが使用可能というわけではありません。以下のシステム・サービス呼び出しは、メッセージ・ドリブン領域で有効です。

CHKP (基本)

DEQ

INIT

LOG

SETS

ROLB

ROLS

ただし、他の条件により、この環境でのこれらの機能が制限される場合があります。代替端末 PCB を使用して出されるオプションまたは呼び出しに制約はありません。

DC 呼び出しとデータ域のコーディング

DC 呼び出しとデータ域をコーディングする方法は、使用するアプリケーション・プログラミング言語によって異なります。

プログラムをコーディングする前に

ユーザーは、プログラムが行うデータベース処理に関する情報の他に、メッセージ処理に関する情報も知っている必要があります。コーディングを開始する前に、このような情報を見落としていないかを確かめる必要があります。また、プログラムの作成上考慮しなければならないインストール・システムの標準仕様についても知っておく必要があります。

プログラムの設計に関して必要な情報:

- プログラムの通信相手となる論理端末の名前
- プログラムがメッセージを送るアプリケーション・プログラムの MPP スケルトンのトランザクション・コード (ある場合)
- プログラムでの DC 呼び出しの構造
- 送信する各出力メッセージの宛先
- プログラムがメッセージを送る代替宛先の名前

入力メッセージに関して必要な情報:

- プログラムが受け取る入力メッセージのサイズとレイアウト (わかっている場合)
- プログラムが入力メッセージを受け取るときの形式
- プログラムが使用する編集ルーチン
- 入力メッセージの有効データの範囲
- 入力メッセージのデータのタイプ

- 入力メッセージ・セグメントの最大および最小の長さ
- メッセージのセグメントの個数

出力メッセージに関して必要な情報:

- アプリケーション・プログラムの MPP スケルトンから IMS に渡す出力の形式
- 出力メッセージの宛先
- 出力メッセージ・セグメントの最大および最小の長さ

MPP のサンプル・コード

MPP アプリケーションは、アセンブラー言語、COBOL、C、Pascal、および PL/I で作成することができます。

次のサンプル・コードでは、プログラムに、標準的な MPP が備えているすべての処理ロジックが含まれているわけではありません。これらのサンプル・プログラムは、アセンブラー言語、COBOL、C 言語、Pascal、および PL/I で基本的な MPP 構造を示すためのものです。どのプログラムも以下のステップに従います。

1. プログラムは、I/O PCB に GU 呼び出しを発行することによって、端末から、入力メッセージ・セグメントをリトリーブします。この呼び出しはメッセージの最初のセグメントをリトリーブします。このメッセージにセグメントが 1 つしか入っていないことがわかっている場合以外、プログラムでは I/O PCB に GN 呼び出しを何回か出して、メッセージの残りのセグメントをリトリーブします。IMS は、呼び出しで指定した入出力域に入力メッセージ・セグメントを入れます。MPP スケルトンの各例では、これは MSG-SEG-IO-AREA です。
2. プログラムは、DB PCB に GU 呼び出しを発行することによって、データベースからセグメントをリトリーブします。この呼び出しでは、要求を限定するために SSA (ここでは SSA-NAME) を指定します。 IMS は、呼び出しで指定された入出力域にデータベース・セグメントを入れます。この場合の入出力域の名前は DB-SEG-IO-AREA です。
3. プログラムは、代替 PCB に ISRT 呼び出しを発行することによって、代替宛先に出力メッセージを送ります。プログラムは、ISRT 呼び出しを発行する前に、入出力域に出力メッセージ・セグメントを作成し、その入出力域を ISRT 呼び出しの中で指定しなければなりません。この呼び出しで使用する入出力域の名前は ALT-MSG-SEG-OUT です。

サンプル・プログラムは、説明のために単純化されています。例えば、サンプル・プログラムでは、同期点開始のための呼び出しが含まれていません。完全なアプリケーション・プログラムには、他の IMS 呼び出しも含める必要があります。

アセンブラー言語での MPP プログラム・コーディング

アセンブラー言語の MPP のコーディング規則は、DL/I アセンブラー・プログラムのコーディング規則と同じです。

アセンブラー言語の MPP は、入り口ステートメントが実行されるときに、PCB パラメーター・リスト・アドレスをレジスター 1 で受け取ります。このリストの中の最初のアドレスは、TP PCB を指すポインターです。I/O PCB アドレスの後に、プ

ログラムが使用する代替 PCB のアドレスが続き、さらにプログラムが使用するデータベース PCB のアドレスが続きます。最後のアドレス・パラメーターのビット 0 は、1 に設定されています。

C 言語での MPP プログラム・コーディング

下記のプログラムは、C 言語で書かれたスケルトンの MPP です。

プログラムの右側の数字は、このプログラムの後の注の番号に対応しています。C 言語アプリケーション・プログラムから IMS への呼び出しのパラメーター・リストで参照されるすべての記憶域は、拡張仮想記憶域に置くことができます。

C で作成された MPP スケルトン

```
                                                                    NOTES
#pragma runopts(env(IMS),plist(IMS))                                  1
  #include <ims.h>
  #include <stdio.h>
  /*                                                          */
  /*                     ENTRY POINT                          */
  /*                                                          */
  main() {                                                            2
    static const char func_GU[4] = "GU  ";                            3
    static const char func_ISRT[4] = "ISRT";
   .
    #define io_pcb ((IO_PCB_TYPE *)(_pcblist[0])
4
    #define alt_pcb (_pcblist[1])
    #define db_pcb  (_pcblist[2])
   .
    int rc;
5
   .
    #define io_pcb ((IO_PCB_TYPE *)(_pcblist[0])
6
    #define alt_pcb (_pcblist[1])
    #define db_pcb  (_pcblist[2])
   .
   rc = ctdli(func_GU, io_pcb, msg_seg_io_area);
7
  .
    rc = ctdli(func_GU, db_pcb, db_seg_io_area, ssa_name);
8
  .
    rc = ctdli(func_ISRT, alt_pcb, alt_msg_seg_out);
9
  .
 }
10
C language interface
11
```

注:

1. IMS のもとで呼び出されると、env(IMS) は正しい操作環境を確立し、plist(IMS) は正しいパラメーター・リストを設定します。 ims.h ヘッダー・ファイルには、PCB レイアウト、__pcblist、および ctdli ルーチンの宣言が入っています。 PCB レイアウトは、プログラムが構造として使用する DB PCB のマスクを定義します。これらの定義を行っておけば、プログラムで DB PCB 内のフィールドを検査することができます。

stdio.h ヘッダー・ファイルには、sprintf に関する宣言が入っています。これは、SSA の構築に役立ちます。

2. IMS は、アプリケーション・プログラムの PSB をロードした後、このエントリー・ポイントを介してアプリケーション・プログラムに制御を渡します。
3. これらは、機能コードを定義するのに便利です。 include ファイルの 1 つに組み入れることもできます。
4. これらを構造の形にしても、効率が下がることはありません。
5. DL/I 呼び出しの戻りコード (状況値) の受取と使用を分離して行うことができます。
6. C 言語の実行時に __pcblist の値がセットされます。 PCB を参照する順序は、PSB で定義されている順序と同じでなければなりません。つまり、最初に TP PCB、次にプログラムが使用する代替 PCB、最後にプログラムが使用するデータベース PCB の順です。
7. プログラムは、最初のメッセージ・セグメントをリトリーブするために、I/O PCB に GU 呼び出しを発行します。他の方法で状況を調べる場合は、rc = を省くことができます。
8. プログラムは、データベース・セグメントをリトリーブするために、DB PCB に GU 呼び出しを発行します。これら 2 つの呼び出しの機能コードは同じです。IMS は、それぞれの呼び出しが参照する PCB によって両者を識別します。
9. 次に、プログラムは、代替 PCB に ISRT 呼び出しを発行することによって、代替宛先に出力メッセージを送ります。
10. プログラムが処理するメッセージがこれ以上ないときは、プログラムは、main から戻るか、exit() を呼び出すことによって、IMS に制御権を返します。
11. IMS は、IMS への共通インターフェースとなる言語インターフェース・モジュール (DFSLI000) を用意しています。このモジュールは、バインド時にアプリケーション・プログラムによって使用できるようになっていなければなりません。

COBOL での MPP プログラム・コーディング

下記のプログラムは、COBOL で書かれたスケルトンの MPP であり、MPP の主な要素を示しています。

プログラムの各部分の右側の数字は、このプログラムの後の注の番号と対応しています。 IBM COBOL for z/OS & VM プログラムをプリロードする場合は、コンパイラー・オプション RENT を使用する必要があります。VS COBOL II プログラムをプリロードする場合は、コンパイラー・オプション RES と RENT を使用する必要があります。

IBM COBOL for z/OS & VM コンパイラーを使用して、AMODE(31) で実行されるプログラムを z/OS でコンパイルする場合は、コンパイラー・オプション RENT を使用してください。 VS COBOL II コンパイラーを使用して、z/OS で AMODE(31) で実行されるプログラムをコンパイルする場合は、コンパイラー・オプション RES と RENT を使用する必要があります。IMS に対する呼び出しのパラメーター・リストで参照される記憶域はすべて、オプションで拡張仮想記憶域に置くこともできます。

IBM COBOL for z/OS & VM プログラムおよび VS COBOL II プログラムは、同じアプリケーション内に同時に存在させることができます。

COBOL で作成された MPP スケルトン

```
                                                                  NOTES:
  ENVIRONMENT DIVISION.
      .
      .
      .
  DATA DIVISION.
  WORKING-STORAGE SECTION.                                         1
      77 GU-CALL PICTURE XXXX VALUE 'GU  '.
      77 ISRT-CALL PICTURE XXXX VALUE 'ISRT'.
      77 CT PICTURE S9(5) COMPUTATIONAL VALUE +4.
      01 SSA-NAME.
      .
      01 MSG-SEG-IO-AREA.                                          2
      .
      01 DB-SEG-IO-AREA.
      .
      01 ALT-MSG-SEG-OUT.
      .
  LINKAGE SECTION.
      01 IO-PCB.                                                   3
      .
      01 ALT-PCB.
      .
      01 DB-PCB.
      .
  PROCEDURE DIVISION USING IO-PCB, ALT-PCB, DB-PCB                 4
      .
      CALL 'CBLTDLI' USING GU-CALL, IO-PCB,                        5
          MSG-SEG-IO-AREA.
      .
      CALL 'CBLTDLI' USING GU-CALL, DB-PCB,                        6
          DB-SEG-IO-AREA, SSA-NAME.
      .
      CALL 'CBLTDLI' USING ISRT-CALL, ALT-PCB,                     7
          ALT-MSG-SEG-OUT.
      .
      GOBACK.                                                      8
COBOL LANGUAGE INTERFACE                                           9
```

注:

1. プログラムで使用する呼び出し機能は、それぞれを、WORKING-STORAGE の 77 または 01 レベルのステートメントを用いて定義します。呼び出し機能の値は、4 桁の英数字として PICTURE 文節で割り当てます。
2. メッセージ・セグメントのために使用する入出力域は、それぞれを 01 レベルの WORKING-STORAGE のステートメントとして定義します。
3. プログラムの LINKAGE SECTION では、プログラムで使用する各 PCB を、01 レベルの項目として定義します。入り口ステートメントに列挙する順序で PCB をリストすることができますが、必ずしもそうする必要はありません。
4. プロシージャー・ステートメントで、プログラムで使用する PCB を、プログラムの PSB に定義されている順序でリストします。つまり、最初に TP PCB、次に代替 PCB、最後にプログラムが使用するデータベース PCB の順です。
5. プログラムは、入力メッセージの最初のセグメントをリトリーブするために、I/O PCB に GU 呼び出しを発行します。

6. SSA-NAME 域で記述されているセグメントをリトリーブするために、プログラムは DB PCB に GU 呼び出しを発行します。
7. プログラムは、代替 PCB を用いて、出力メッセージ・セグメントを代替宛先に送ります。
8. MPP が処理するメッセージがなくなった時点で、GOBACK ステートメントを実行し、制御権を IMS に返します。

   タスクのすべての COBOL プログラムを VS COBOL II でコンパイルする場合は、GOBACK ステートメントを COBOL で定義されている通常のセマンティクスで使用することができます。

   重要: STOP RUN ステートメントおよび EXIT PROGRAM ステートメントは、サポートされません。これらのステートメントを使用すると、結果が予測不能となるか異常終了する場合があります。
9. COBOL コンパイラー・オプション NODYNAM を指定する場合は、言語インターフェース・モジュール DFSLI000 を、コンパイル済みの COBOL 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。COBOL コンパイラー・オプション DYNAM を指定する場合は、DFSLI000 とコンパイル済みの COBOL プログラムのリンク・エディットはしないでください。

Pascal での MPP プログラム・コーディング

下記のプログラムは、Pascal で書かれたスケルトンの MPP です。

プログラムの右側の数字は、このプログラムのあとの注の番号と対応しています。Pascal アプリケーション・プログラムから IMS への呼び出しのパラメーター・リストで指定するすべての記憶域は、拡張仮想記憶域に置くこともできます。

Pascal で作成された MPP スケルトン

```
                    NOTES:
  segment PASCIMS;                                          1
  type
    CHAR4 = packed array [1..4] of CHAR;2
    CHARn = packed array [1..n] of CHAR;
    IOPCBTYPE = record                                      3
                  (* Field declarations *)
                end;
    ALTPCBTYPE = record
                  (* Field declarations *)
                end;
    DBPCBTYPE = record
                  (* Field declarations *)
                end;
procedure PASCIMS (var SAVE: INTEGER;                       4
                     var IOPCB:  IOPCBTYPE;
                     var ALTPCB: ALTPCBTYPE;
                     var DBPCB:  DBPCBTYPE); REENTRANT;
procedure PASCIMS;
type                                                         5
  SSATYPE   = record
                (* Field declarations *)
              end;

  MSG_SEG_IO_AREA_TYPE = record
                           (* Field declarations *)
                         end;
```

```
  DB_SEG_IO_AREA_TYPE = record
                          (* Field declarations *)
                        end;

  ALT_MSG_SEG_OUT_TYPE = record
                           (* Field declarations *)
                         end;
var                                                          6
  MSG_SEG_IO_AREA : MSG_SEG_IO_AREA_TYPE;
  DB_SEG_IO_AREA : DB_SEG_IO_AREA_TYPE;
  ALT_MSG_SEG_OUT : ALT_MSG_SEG_OUT_TYPE;
const                                                        7
  GU   = 'GU  ';
  ISRT = 'ISRT';
  SSANAME  = SSATYPE(...);
procedure PASTDLI; GENERIC;                                  8
begin
  PASTDLI(const GU,                                          9
          var   IOPCB,
          var   MSG_SEG_IO_AREA);
  PASTDLI(const GU,                                          10
          var   DBPCB,
          var   DB_SEG_IO_AREA,
          const SSANAME);
  PASTDLI(const ISRT,                                        11
          var   ALTPCB,
          var   ALT_MSG_SEG_OUT);
end;                                                         12
Pascal language interface                                    13
```

注:

1. Pascal のコンパイル単位の名前を定義します。
2. プログラムで使用する PCB に必要なデータ・タイプを定義します。
3. プログラムで使用する PCB データ・タイプを定義します。
4. IMS により呼び出される REENTRANT プロシージャーのヘッダーを宣言します。パラメーター・リストの最初の語は INTEGER でなければなりません。この語は VS Pascal が使用するために予約されています。パラメーターの残りは IMS から渡される PCB のアドレスです。
5. SSA および入出力域に必要なデータ・タイプを定義します。
6. SSA および入出力域で使用される変数を宣言します。
7. PASTDLI DL/I 呼び出しで使用される定数 (機能コード、SSA など) を定義します。
8. GENERIC 指示により IMS インターフェース・ルーチンを宣言します。GENERIC は、複数のパラメーター・リスト・フォーマットが使用できる外部ルーチンを識別します。 GENERIC ルーチンのパラメーターは、ルーチンが呼び出されるときに初めて「宣言されます」。
9. プログラムは、入力メッセージの最初のセグメントをリトリーブするために、入出力 PCB に GU 呼び出しを発行します。プログラムのパラメーターの宣言は、プログラムによってこの例と異なることがあります。
10. プログラムは、データベース・セグメントをリトリーブするために、GU 呼び出しを DB PCB に対して発行できます。これら 2 つの呼び出しの機能コードは同じです。IMS は、それぞれの呼び出しが参照する PCB によって両者を識別します。プログラムのパラメーターの宣言は、プログラムによってこの例と異なることがあります。

11. プログラムは、代替 PCB に ISRT 呼び出しを発行することによって、代替宛先に出力メッセージ・セグメントを送ります。プログラムのパラメーターの宣言は、プログラムによってこの例と異なることがあります。
12. MPP が処理するメッセージがこれ以上ないときは、PASCIMS プロシージャーを終了させることによって IMS に制御権を返します。または、RETURN ステートメントをコーディングして、他の場所で返すこともできます。
13. プログラムは、コンパイル後、IMS 言語インターフェース・モジュール DFSLI000 にバインドしなければなりません。

PL/I での MPP プログラム・コーディング

下記のプログラムは、PL/I で書かれたスケルトンの MPP です。

プログラムの右側の番号は、プログラムに付けられた注に対応しています。 PL/I アプリケーション・プログラムから IMS への呼び出しのパラメーター・リストで指定するすべての記憶域は、拡張仮想記憶域に置くこともできます。

PL/I プログラムの実行を 31 ビット・アドレッシング・モードで計画している場合は、「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。

PL/I で作成された MPP スケルトン

```
                                                              NOTES
/*                                                          */
/*                    ENTRY POINT                           */
/*                                                          */
    UPDMAST:  PROCEDURE (IO_PTR, ALT_PTR, DB_PTR)                    1
    OPTIONS (MAIN);
DCL   FUNC_GU     CHAR(4)   INIT('GU  ');                            2
DCL   FUNC_ISRT   CHAR(4)   INIT('ISRT');
.
DCL   SSA_NAME...;
.
DCL   MSG_SEG_IO_AREA    CHAR(n);                                    3
DCL   DB_SEG_IO_AREA     CHAR(n);
DCL   ALT_MSG_SEG_OUT    CHAR(n);
.
DCL   1 IO_PCB BASED (IO_PTR),...;                                   4
DCL   1 ALT_PCB BASED (ALT_PTR),...;
DCL   1 DB_PCB BASED (DB_PTR),...;
.
DCL   THREE FIXED BINARY(31) INIT(3);                                5
DCL   FOUR FIXED BINARY(31)  INIT(4);
DCL   PLITDLI ENTRY EXTERNAL;
.
CALL PLITDLI (THREE, FUNC_GU, IO_PTR, MSG_SEG_IO_AREA);              6
.
CALL PLITDLI (FOUR, FUNC_GU, DB_PTR, DB_SEG_IO_AREA,                 7
     SSA_NAME);
.
CALL PLITDLI (THREE, FUNC_ISRT, ALT_PTR, ALT_MSG_SEG_OUT);           8
.
END UPDMAST;                                                         9
PL/I LANGUAGE INTERFACE                                              10
```

注:

1. これは、PL/I 最適化コンパイラー MPP への標準のエントリー・ポイントです。このステートメントには、MPP が使用する各 PCB へのポインターが含まれています。 PCB は、PSB にリストされている順序つまり、最初に TP

PCB、次にプログラムが使用する代替 PCB、最後にプログラムが使用するデータベース PCB の順で参照しなければなりません。

2. プログラムのデータ域に、プログラムで使用する各呼び出し機能を定義します。PL/I の場合は、文字ストリングとして機能コードを定義し、適切な値を割り当てます。
3. PCB マスクを、PROCEDURE ステートメントで渡されたアドレスに基づいて大構造として定義してください。例には示されていませんが、マスクが関連付けられる PCB のタイプに応じて、構造に適切な追加のフィールドをコーディングします。
4. ユーザーの PCB を定義するには、大構造の宣言を使用します。
5. PL/I の呼び出しには、COBOL プログラムやアセンブラー言語プログラムにはないパラメーターがあります。 parmcount というこのパラメーターは、常に最初に指定します。プログラムが出す各呼び出しで必要な値を parmcount として定義しておきます。parmcount は、parmcount の後に続くパラメーターの個数を表します。
6. プログラムは、最初のメッセージ・セグメントをリトリーブするために、I/O PCB に GU 呼び出しを発行します。
7. プログラムは、データベース・セグメントをリトリーブするために、GU 呼び出しを DB PCB に対して発行できます。これら 2 つの呼び出しの機能コードは同じです。IMS は、それぞれの呼び出しが参照する PCB によって両者を識別します。
8. 次に、プログラムは、代替 PCB に ISRT 呼び出しを発行することによって、代替宛先に出力メッセージを送ります。
9. プログラムが処理するメッセージがこれ以上ないときは、プログラムは、END ステートメントまたは RETURN ステートメントを出すことによって、IMS に制御権を返します。
10. プログラムは、コンパイル後、IMS 言語インターフェース・モジュール DFSLI000 にバインドしなければなりません。

DB2 のためのメッセージ処理の考慮事項

DB2 データベースにアクセスする従属領域のメッセージ処理機能と DL/I データベースだけにアクセスする従属領域のメッセージ処理機能は、大部分が同じです。メッセージのリトリーブと送信、データベースの変更をバックアウトするためにプログラムが用いている方法は同じです。

異なる点は、以下のとおりです。

- DL/I ステートメントのコーディング方法が SQL (構造化照会言語) ステートメントの場合とは異なる。
- IMS TM アプリケーション・プログラムが IMS TM から制御を受け取った時点では、IMS は既にプログラムがアクセスできるリソースを獲得している。IMS TM は、プログラムをスケジュールしますが、データベースの中には利用可能な状態でないものもあります。DB2 では、プログラムが最初の SQL ステートメントを出すまでは、そのプログラムに対してリソースを割り当てません。プログラムが必要とするリソースを DB2 が割り振ることができない場合には、プログラムが最初の SQL 呼び出しを発行した時点で初期設定エラーをオプションで受け取るようにすることもできます。

- アプリケーションが出したチェックポイント呼び出しまたはメッセージ GU 呼び出しが成功すると、DB2 はプログラムが使用しているすべてのカーソルをクローズする。つまり、ユーザー・プログラムは、チェックポイント呼び出しまたはメッセージ GU の後で、OPEN CURSOR ステートメントを出さなければなりません。

IMS TM と DB2 は連携して、以下の方法によりデータ保全性を維持します。

- ユーザー・プログラムがコミット・ポイントに達すると、IMS TM は、プログラムが DL/I データベースに対して行った変更を永続変更とし、出力メッセージを宛先に送付し、プログラムがコミット・ポイントに達したことを DB2 に通知する。次に DB2 は、プログラムが DB2 データベースに対して行った変更を永続化させます。
- ユーザー・プログラムが異常終了するか、または IMS TM のロールバック呼び出し (ROLB、トークンなしの ROLS、または ROLL) のいずれかを出した場合、IMS TM は、ユーザー・プログラムが作成したすべての出力メッセージを取り消し、最後のコミット・ポイント以降にユーザー・プログラムが DL/I データベースに対して行った変更をバックアウトし、DB2 に通知します。DB2 は、最後のコミット・ポイント以降にユーザー・プログラムが DB2 データベースに対して行った変更をバックアウトします。

自動化操作プログラム・インターフェース (AOI) により、IMS TM アプリケーション・プログラムは、DB2 コマンドおよび IMS TM コマンドを発行できます。DB2 コマンドを発行するには、DB2 コマンドの前に IMS TM /SSR コマンドを発行します。 /SSR コマンドの出力は、マスター端末オペレーター (MTO) に送られます。

第 26 章 IMS スプール API

IMS が CHNG 呼び出しおよび SETO 呼び出しの印刷データ・セット・オプションにエラーを検出すると、IMS スプール API サポートは、アプリケーション・プログラムにフィードバックします。

アプリケーション・プログラムは IMS プリンターにメッセージを送ったり、他のアクションを実行して、これらのエラーを表示するので、ダンプ・リストを見なくてもパラメーター・リストおよびフィードバック域を調べることができ便利です。この情報は、スプール API サポートと共に使用される呼び出しにのみ適用されます。

IMS スプール API の全体設計の管理

IMS スプール API (アプリケーション・プログラミング・インターフェース) は、IMS アプリケーション・プログラミング・インターフェースの拡張です。これを使用すると、アプリケーションは JES に直接インターフェースをとったり、ジョブ入力サブシステム (JES) スプールに印刷データ・セットを作成したりすることができます。これらの印刷データ・セットは、印刷管理機能およびスプール・サーバーで使用することができ、アプリケーションが必要としているものを提供します。

IMS スプール API 設計

IMS スプール API 設計により、アプリケーション・プログラムは、標準 DL/I 呼び出しインターフェースを使用して、JES スプールに印刷データ・セットを作成できるようになります。

次の機能が提供されます。

データ・セット出力特性の定義

データ・セットの割り振り

データ・セットへの印刷行の挿入

データ・セットのクローズおよび割り振り解除

JES インターフェースの制限内での非コミット・データのバックアウト

未確定印刷データ・セットの制御の援助

IMS スプール API サポートは、既存の DL/I 呼び出しを使用して、データ・セットの割り当て情報を指定したり、印刷データ・セットにデータを配置したりします。以下はこれらの呼び出しです。

- CHNG 呼び出し。この呼び出しは、割り振られる印刷データ・セット用に印刷データ・セット特性を指定できるように拡張されています。この処理では、印刷データ・セットに関連したインターフェース・ブロックとして、代替 PCB を使用します。
- ISRT 呼び出し。この呼び出しは、最初の挿入時に印刷データ・セットの動的割り振りを実行し、データ・セットにデータを書き込むよう拡張されています。このデータ・セットは、作業単位 (UOW) が終了するまで未確定と見なされます。

可能であれば、同期点処理は、異常終了する作業単位については、未確定のデータ・セットをすべて削除し、正常に終了する作業単位については、データ・セットをクローズして割り振り解除します。

- SETO 呼び出し。このサポートのためにインストールされた呼び出し、SETO (オプション設定) です。この呼び出しを使用して動的出力テキスト単位を作成し、以降の CHNG 呼び出しで使用します。多くの印刷データ・セットが同じ出力記述子を使用している場合、 SETO 呼び出しを使用してオーバーヘッドを削減し、動的出力処理のためテキスト単位を必要なものに事前構築します。

JES スプール・データ・セットへのデータの送信

アプリケーション・プログラムは、代替端末装置への出力の送信に使用されるのと同一方式を使用して JES スプール・データ・セットにデータを送信できます。 DL/I 呼び出しを使用して、出力宛先を JES スプール・データ・セットに変更します。

メッセージを挿入するには、DL/I ISRT または PURG 呼び出しを使用してください。

CHNG 呼び出しおよび SETO 呼び出しのオプション・リスト・パラメーターには、データ・セットのプリンター処理オプションもあります。これらのオプションは、適切な IMS スプール API データ・セットへ出力を指示します。 DL/I 呼び出しにおけるこれらのオプションは、MVS スケジューラー JCL 機能 (SJF) により妥当性検査されます。オプションが無効である場合、エラー・コードがアプリケーションに戻されます。エラーに関する情報を受信するには、アプリケーション・プログラムは、 CHNG あるいは SETO DL/I 呼び出しパラメーター・リストに、フィードバック域を指定します。フィードバック域がある場合には、オプション・リスト・エラーに関する情報がアプリケーションに直接戻されます。

IMS スプール API のパフォーマンスの考慮事項

IMS スプール API インターフェースは、他の IMS トランザクションおよびサービスに対する、z/OS サービスの与えるパフォーマンス上の影響を最小化しつつ、IMS アプリケーション内で z/OS サービスを使用します。

このため、IMS スプール API サポートは、中間記憶装置に対し、 IMS メッセージ・キューを使用する代わりに、挿入時に、印刷データを直接 JES スプールに出力します。 IMS スプール API 要求の処理は、従属領域の TCB 下で実行され、N-way プロセッサーを最大限に利用しようとします。この設計は、エラー・リカバリーおよび JES ジョブ方向付けの問題を削減します。

JES イニシエーターの考慮事項

通常、従属領域は、長時間稼働するジョブなので、従属領域が IMS スプール API を使用している場合、開始プログラムあるいはジョブ指定の一部を、変更しなければならない可能性があります。

動的割り振りおよび割り振り解除のメッセージの組み込みのため、従属領域が使用する JES スプール・スペースの量を、限定する必要がある場合があります。例えば、JOB ステートメントの MSGLEVEL を使用して、従属領域に対するジョブ・ロ

グから、動的割り振りメッセージを除去することができます。 z/OS 開始タスクとして実行している従属領域に対するこれらのメッセージを、除去できる可能性があります。

その他の開始プログラムの考慮事項は、従属領域に対する JES ジョブ・ジャーナルの使用に関することです。ジョブ・ステップに関連するジャーナルがある場合、z/OS チェックポイント再始動に関連する情報は、ジャーナルに記録されます。IMS 従属領域は、z/OS チェックポイント再始動を使用することができないので、JES2 イニシエーター・プロシージャー、および従属領域実行クラスに関連した JES3 クラスについては、JOURNAL=NO を指定してください。また、ジョブとして実行される従属領域については、 JES3 //*MAIN ステートメントに JOURNAL= を指定することもできます。

アプリケーション管理のテキスト単位

アプリケーションは、IMS の代わりに動的記述子テキスト単位を管理できます。アプリケーションがテキスト単位を管理する場合、構文解析およびテキスト単位構築のオーバーヘッドを削減できます。

SETO 呼び出しを使用して、IMS 構築の動的記述子テキスト単位を入手してください。いったん構築すれば、これらのテキスト単位を以降の CHNG 呼び出しとともに使用して、データ・セットの印刷特性を指定することができます。

テキスト単位を管理してオーバーヘッドを削減するには、テキスト単位を多くの変更呼び出しとともに使用してください。この例としては、入力待ち (WFI) トランザクションがあります。すべての印刷データ・セットに対し、同じデータ・セット属性が使用されます。最初に処理されたメッセージに対し、アプリケーションは、動的記述子に対するテキスト単位を構築する SETO 呼び出しと、その後に、事前構築テキスト単位を参照する TXTU= パラメーターを指定した、 CHNG 呼び出しを使用します。以降のメッセージすべてに対して必要なものは、事前構築テキスト単位を使用する、 1 つの CHNG 呼び出しだけです。

注**:** 事前構築テキスト単位を使用して節約される可能性があるオーバーヘッド量を判別するテストは、今のところ実施されていません。

BSAM 入出力域

スプール・メッセージの入出力域が、非常に大きなものである可能性があります。32K バイトの長さの入出力域は珍しくはありません。大容量のバッファーを移動することによって生じるオーバーヘッドを削減するため、 IMS はアプリケーションの入出力域から、スプール・データ・セットへの書き込みを試みます。

BSAM は、SYSOUT ファイルに対して 31 ビット・ストレージ内での入出力域をサポートしません。 IMS がアプリケーションの入出力域が 31 ビット・ストレージにあることを検出すると、以下のことが行われます。

- 作業域は、24 ビット・ストレージから獲得されます。
- アプリケーションの入出力域は、作業域へ移動されます。
- スプール・データ・セットは、作業域から書き込まれます。

アプリケーションの入出力域を 24 ビット・ストレージに簡単に配置できる場合、入出力域の移動を避けることができ、可能なパフォーマンスの改良が達成されます。

注: パフォーマンス向上の可能性を量的に判別するテストは行われていません。

アプリケーションの入出力域から BSAM がレコードを直接書き込むので、入出力域は、BSAM が予定した形式でなければなりません。その形式には、以下のものが必要です。

- 可変長レコード
- ブロック記述子ワード (BDW)
- レコード記述子ワード (RDW)

IMS スプール API のアプリケーション・コーディングの考慮事項

使用中のアプリケーションは、印刷データ・セットを使用して、JES スプールまたは印刷サーバーへデータを送信できます。障害が生じた場合のメッセージ保全性およびデータ・リカバリーに関するオプションを設定できます。

印刷データ形式

IMS スプール API は、アプリケーションに透過インターフェースを提供して、JES スプールへデータを挿入しようと試みます。そのデータの形式は、行、ページ、IPDS、AFPDS、あるいは、印刷データ・セットを処理する JES スプールまたは印刷サーバーが操作できるフォーマットです。 IMS スプール API は、JES スプールに挿入されるデータの変換や修正は行いません。

メッセージ保全性のオプション

IMS スプール API は、メッセージ保全性をサポートします。

これが必要になるのは、以下のように、 IMS が印刷データ・セットの後処理を正しく制御できない場合です。

- ハードウェアあるいはソフトウェアに問題があり、IMS 異常終了が実行されない場合
- 印刷データ・セットについて、動的割り振り解除エラーがある場合
- IMS コードに論理エラーがある場合

このような条件下にあると、IMS は、JES サブシステムが部分印刷データ・セットを印刷するのを停止できなくなる可能性があります。また、JES サブシステムは、2 フェーズ同期点をサポートしません。

印刷の後処理

Advanced Function Printing (AFP) (高機能印刷) を使用する最も一般的なアプリケーションは、TSO ユーザーおよびバッチ・ジョブです。障害が生じた際に、これらのアプリケーションのどれかが印刷データ・セットを作成している場合、部分印刷データ・セットはおそらく印刷され、手動で処理されることになります。印刷データ・セットを作成する多くの IMS アプリケーションは、同じ方法で部分印刷データ・セットを管理できます。部分印刷データ・セットの自動印刷を JES によりさらに制御する必要があるアプリケーションについては、IMS スプール API に次の

整合性オプションがあります。しかし、これらのオプションのみでは、部分印刷データ・セットに正しい後処理を保証できません。これらのオプションは、CHNG 呼び出しに使用される、 IAFP キーワードに続く **b** 変数です。

b=0

データ・セット保護なしを指示します。

これは、最も一般的なオプションです。このオプションが選択されると、印刷データ・セットの割り振りあるいは割り振り解除の間、IMS は特別な操作は行いません。このオプションが選択された場合に、IMS が印刷データ・セットの後処理を正しく処理するのを妨げる何らかの条件が発生すると、部分データ・セットは、おそらく印刷されますが、手動で制御されなければなりません。

b=1

SYSOUT HOLD の保護を指示します。

このオプションは、JES オペレーターが直接処置をとらなければ、部分印刷データ・セットが印刷のために解放されることがないようにします。データ・セットが割り振られる場合、この印刷データ・セットを SYSOUT HOLD 状況にするように、割り振り要求が JES に指示します。 IMS が何らかの理由でデータ・セットを割り振り解除できなかった場合、このデータ・セットについて SYSOUT HOLD 状況が維持されます。印刷データ・セットが HOLD 状況にあるので、JES オペレーターは部分印刷データ・セットを識別して、このデータ・セットを削除あるいは印刷するために、JES コマンドを出す必要があります。

印刷データ・セットを削除、あるいは印刷できない場合、

- 印刷データ・セットが割り振り解除できない場合、メッセージ DFS0012I が出されます。
- 未確定印刷データ・セットが検出された場合、 IMS 緊急時再始動の間にメッセージ DFS0014I が出されます。このメッセージは、適切な印刷データ・セットを検出して、適切な印刷後処理を行うことができる情報を、JES オペレーターに提供します。

  その情報のいくつかは、以下のとおりです。

  – JOBNAME
  – DSNAME
  – DDNAME
  – データ・セットに対する適切な後処理であると IMS が認識しているものに関する勧告(例えば、印刷あるいは削除)

スプール表示および検索機能 (SDSF) を使用すると、保留データ・セットの表示、 DDNAME および DSNAME による未確定印刷データ・セットの識別、および適切な JES コマンドを出して、印刷データ・セットを削除あるいは解放することが可能になります。

b=2

選択不能な宛先を指示します。

このオプションは、部分印刷データ・セットの自動印刷を妨げます。 IMS スプール API 機能は、データ・セットを割り振る際、データ・セットに対して IMSTEMP のリモート宛先を要求します。 JES システムには、その宛先に送信

するどのようなデータ・セットの印刷も、 JES が試みないように定義された、IMSTEMP のリモート宛先がなければなりません。

b=2 である場合、CHNG 呼び出しを出す際には、印刷データ・セットに対するリモート宛先の名前を、呼び出しパラメーター・リストの宛先名フィールドに指定しなければなりません。 IMS が同期点のデータ・セットを割り振り解除する場合、およびデータ・セットを印刷する場合は、要求された最終リモート宛先へのデータ・セットの転送を IMS は要求します。

リモート宛先を JES システムに定義していない場合、動的割り振り障害が起こります。このリモート宛先が選択不能として定義されている場合、あるいは IMS が印刷データ・セットを割り振り解除したり、適切な後処理を制御するのが不可能である場合、z/OS が割り振り解除した際に、印刷データ・セットは、リモート宛先 IMSTEMP と関連付けられたままになります。

割り振り解除エラーが起こった場合、メッセージ DFS0012I が出され、割り振り解除エラーの詳細が分かります。それは、オペレーター・アクションを要求する印刷データ・セットの識別に役立ちます。部分印刷データ・セットがこの特殊なリモート宛先に残される場合、 JES オペレーターは、この JES 宛先と関連するすべての印刷データ・セットを表示して、処置を要求したデータ・セットを見つけることができます。 **b=2** オプションは、部分印刷データ・セットを見つけるオペレーターの作業を単純化します。

メッセージ・オプション

IAPF キーワードの 3 つ目のオプションは、 IMS スプール API サポートが出す通知メッセージを制御します。これらのメッセージは、処置が必要な未確定データ・セットを JES オペレーターに通知します。

c=0
印刷データ・セットに対し、 DFS0012I あるいは DFS0014I メッセージが出されないことを指示します。 **b=0** が指定されている場合のみ、**c=0** を指定できます。

c=m
必要であれば、DFS0012I および DFS0014I メッセージが出されることを指示します。 **c=m** を指定できます。 **b=1** あるいは **b=2** の場合は、デフォルトになります。

オプション**c** は、メッセージ DFS0013E を出すのに影響しません。

IMS 緊急時再始動

IMS 緊急時再始動を実行する際、印刷データ・セットの適切な後処理が未確定であることを IMS が検出すると、再始動の結果として DFS0014I メッセージが出される可能性があります。印刷データ・セットに対するメッセージ・オプションが要求されているか、あるいは IAFP 変数が **c=m** である場合は、このメッセージが出されるだけです。 DFS0014I メッセージを受信した場合、JES オペレーターは、印刷データ・セットの検出および正しい処理を行うことが必要になる場合があります。DFS0014I メッセージから、推奨される後処理 (削除あるいは印刷) が分かります。

宛先名 (LTERM) の使用法

標準 CHNG 呼び出しパラメーター・リストには、宛先名フィールドがあります。典型的なメッセージ呼び出しでは、このフィールドには、この代替 PCB を使用して送信されるメッセージの宛先になる、 LTERM あるいはトランザクション・コードがあります。 ISRT 呼び出しが PCB に対して出される場合、データは LTERM あるいはトランザクションへ送信されます。

しかし、宛先名フィールドは、IAFP キーワードの後に **b=2** を指定しない限り、IMS スプール API 機能に対して意味を持ちません。

b=2 が指定される場合は、以下のようになります。

- その名前は、印刷データ・セットを受信する JES システムがサポートする、有効なリモート宛先でなければなりません。
- その名前が有効なリモート宛先でない場合、動的割り振り解除中にエラーが発生します。

2 以外のオプションを選択している場合、IMS はその名前を使用しません。

LTERM 名は、エラー・メッセージおよびログ・レコード内に現れます。印刷データ・セットを作成するルーチンを、識別する名前を使用してください。この情報は、アプリケーション・プログラム・エラーのデバッグに役立ちます。

構文解析エラーについて

複数の構文解析エラー戻りコードを診断するとき、最も情報を与えてくれるのは、通常、最初に返されるコードです。

キーワード

CHNG 呼び出しおよび SETO 呼び出しのキーワードには、2 つのタイプがあります。キーワードのタイプによって、IMS が実行するキーワード妥当性検査のタイプが決まります。キーワードのタイプは次のとおりです。

- 呼び出しで有効なキーワード (例えば、IAFP、PRTO、TXTU、および OUTN)
- PRTO キーワードのオペランドとして有効なキーワード (例えば、CLASS および FORMS)

長さフィールドの指定を誤ると、IMS が有効なキーワードを検査するときにエラーを生じることになります。IMS が CHNG 呼び出しおよび SETO 呼び出しでキーワードの妥当性を検査しているときは、キーワードの 1 セットが有効になります。IMS が PRTO キーワードでキーワードの妥当性を検査しているときは、もう一方のキーワードのセットが有効です。この理由から、長さフィールドの指定を誤ったためにスキャンが早く終了しすぎたり、呼び出しリスト内の位置のために、有効なはずのキーワードが実際には無効になったりすることがあります。長さフィールドが誤っているキーワードや、呼び出しリスト内の位置が誤っているキーワードを検出すると、IMS は有効なキーワードが無効になっていると報告します。

状況コード

呼び出しで返される状況コードは、エラーの場所も示します。例外もありますが、一般に状況コード AR は、呼び出しに対してキーワードが無効のときに返されます。キーワードが PRTO オプションとして無効なときは、状況コード AS が返されます。

エラー・コード

このトピックでは、アプリケーション・プログラムが受け取ることのできるスプール API のエラー・コードについて説明します。トピック「診断例」には、エラーの例と、エラーの結果としてアプリケーション・プログラムに提供されるエラー・コードが記載されています。

エラー・コード
: 理由

(0002) オプション・キーワードが認識されない。

このエラーの理由としては、以下が考えられます。

- キーワードのスペルが間違っている。
- キーワードのスペルは正しいが、後続の区切り文字が無効である。
- PRTO を表す長さが指定されたフィールドが、オプションの実際の長さよりも短い。
- 指示された呼び出しに対してキーワードが無効である。

(0004) オプション変数に指定した文字数が少なすぎるか、または多すぎる。呼び出しのオプション・リストのキーワードに続くオプション変数が、オプションの長さの制限を超えています。

(0006) オプション変数の長さフィールド (LL) が長すぎて、オプション・リストに入らない。オプション・リストの長さフィールド (LL) は、指定したオプション変数の終わりに達する前に、オプション・リストが終了することを示します。

(0008) オプション変数に無効な文字が入っているか、または英字で始まっていない。

(000A)

必須指定のオプション・キーワードが指定されなかった。

このエラーの理由としては、以下が考えられます。

- オプション・リストに 1 つ以上のキーワードが指定されたため、1 つ以上のキーワードを追加する必要がある。
- オプション・リストに指定した長さはゼロより大きいが、リストにオプションが入っていない。

(000C)

指定したオプション・キーワードの組み合わせが無効である。このエラーの原因としては、以下が考えられます。

- オプション・リストに指定された他のキーワードのためにそのキーワードが許可されない。
- オプション・キーワードが複数回指定されている。

(000E) 印刷データ・セット記述子の解析中に、IMS が 1 つ以上のオペランドでエラーを検出した。通常、IMS は z/OS サービス (SJF) を使用して印刷記述子 (PRTO= オプション変数) の妥当性検査を行います。 IMS が SJF を呼び出すとき、IMS は TSO OUTDES コマンドと同様の妥当性検査を要求します。したがって、IMS は出力記述子の変更を重要と見なしません。ユーザーのシステムについて有効な記述子は、MVS リリース・レベルの機能です。有効な記述子のリストおよび正しい構文については、TSO HELP OUTDES コマンドを使用してください。

IMS は、まず PRTO オプションの形式が SJF サービスを使用できる形式であることを確認します。それ以外の形式の場合には、IMS は、状況コード **AS**、エラー・コード (000E)、および記述エラー・メッセージを返します。SJF 処理中にエラーが検出された場合、SJF からのエラー・メッセージには、(R.C.=xxxx,REAS.=yyyyyyyy) 形式の情報およびエラーを示すエラー・メッセージが含まれます。

関連資料**:** SJF の戻りコードおよび理由コードの詳細については、「*z/OS MVS Programming: Authorized Assembler Services Guide*」を参照してください。

変数によっては、その範囲が初期設定パラメーターによって制御されます。初期設定パラメーターによって影響を受ける変数の例としては、コピーの最大数の値、許容されるリモート宛先、クラス、および用紙名があります。

診断例

以下の例では、スプール API のさまざまなエラー・コードが生成される可能性があるエラーと、その問題の診断について示します。

例を説明する際に必要のない長さフィールドは省略されています。複数行にわたって示されるフィードバックおよびオプション・リストは、連続しています。

エラー・コード (0002)

このセクションでは、エラー・コード 0002 の例を 2 つ示します。

最初の例では、オプション・リストに、キーワード PRTO と TXTU が両方、含まれています。キーワード TXTU は、SETO 呼び出しでは無効です。

```
CALL = SETO
  OPTIONS LIST = PRTO=04DEST(018),CLASS(A),TXTU=SET1
  FEEDBACK = TXTU(0002)
  STATUS CODE = AR
```

2 つ目の例では、PRTO オプションの長さフィールドが、すべてのオプションを含むには短すぎます。つまり、IMS が COPIES および FORMS キーワードを PRTO オプション・リスト域外に検出し、このキーワードが CHNG 呼び出しで無効であることを示しています。

```
CALL = CHNG
  OPTIONS LIST = IAFP=N0M,PRTO=0FDEST(018),LINECT(200),CLASS(A),
                                  COPIES(80),FORMS(ANS)
  FEEDBACK = COPIES(0002),FORMS(0002)
  STATUS CODE = AR
```

エラー・コード (0004)

この例では、OUTN キーワードのオペランドが 9 バイトの長さで、OUTPUT JCL ステートメントの最大値を超えています。

```
CALL = CHNG
  OPTIONS LIST = IAFP=N0M,OUTN=OUTPUTDD1
  FEEDBACK = OUTN(0004)
  STATUS CODE = AR
```

エラー・コード (0006)

この呼び出しに対するオプション・リストの長さが、PRTO キーワードのオペランドをすべて含むには短すぎます。

この例では、正しい長さである X'48' バイト長のオプション・リストを示しています。 PRTO キーワードの長さフィールドは、誤って X'5A' の長さを示しています。 PRTO オプションの長さがオプション・リスト全体の長さを超えているため、IMS は PRTO キーワードを無視し、有効なキーワードについて残りのオプション・リストをスキャンします。フィードバック域には、PRTO (0006) コード (長さエラーを示す) および (0002) コード (PRTO キーワードのエラーを示す) が入っています。これは、オプション・リストの長さフィールドで指定された長さ以下で、最初の PRTO キーワードより後にあるキーワードが、呼び出しに対する有効なキーワードを探すためにスキャンされていたからです。AR という状況コードは、キーワードがこの呼び出しで無効であると考えられ、PRTO キーワードではないことを示しています。

```
CALL = CHNG
                  0400              05
  OPTIONS LIST = 0800IAFP=N0M,PRTO=0ADEST(018),LINECT(200),CLASS(A),
                                    COPIES(3),FORMS(ANS)
  FEEDBACK = PRTO(0006),LINECT(0002),CLASS(0002),COPIES(0002),
                        FORMS(0002)
  STATUS CODE = AR
```

エラー・コード (0008)

この例では、IAFP キーワードのメッセージ・オプションが、誤って「Z」と指定されています。

```
CALL = CHNG
                                00
  OPTIONS LIST = IAFP=N0Z,PRTO=0BDEST(018)
  FEEDBACK = IAFP(0008) INVALID VARIABLE
  STATUS CODE = AR
```

エラー・コード (000A)

この例では、有効なキーワード TXTU が指定されていますが、この呼び出しで TXTU キーワードが使用される場合は、IAFP キーワードの指定も必要です。

```
CALL = CHNG
  OPTIONS LIST = TXTU=SET1
  FEEDBACK = TXTU(000A)
  STATUS CODE = AR
```

エラー・コード (000C)

状況コード **AR** とエラー・コード (000C) が返されます。これは、問題は呼び出しオプションにあり、PRTO オプションは問題ないことを表します。

呼び出しオプション・リストには、PRTO キーワードと TXTU キーワードが含まれています。これらのオプションを、同じオプション呼び出しリストで使用することはできません。

```
CALL = CHNG
                                00
  OPTIONS LIST = IAFP=A00,PRTO=0BCOPIES(3),TXTU=SET1
  FEEDBACK = TXTU(000C)
  STATUS CODE = AR
```

エラー・コード (000E)

この例では、COPIES パラメーターに、オペランドの 1 つとして、誤った値「RG」が指定されています。エラー・メッセージは、オペランドの値は数値でなければならないことを示しています。

```
CALL = CHNG
                                01
  OPTIONS LIST = IAFP=A00,PRTO=0BCOPIES((3),(8,RG,18,80))
  FEEDBACK = PRTO(000E) (R.C.=0004,REAS.=00000204) COPIES/RG VALUE
                        MUST BE NUMERIC CHARACTERS
  STATUS CODE = AS
```

次の例には、無効な PRTO オペランドが含まれています。結果の理由コード X'000000D0' は、XYZ オペランドが無効であることを示しています。

```
CALL = CHNG
                                00
  OPTIONS LIST = IAFP=A00,PRTO=0AXYZ(018)
  FEEDBACK = PRTO(000E) (R.C.=0004,REAS.=000000D0) XYZ
  STATUS CODE = AS
```

割り振りエラーについて

IMS スプール API インターフェースは、データが実際にデータ・セットに挿入されるまで印刷データ・セットの動的割り振りを据え置きます。CHNG 呼び出しまたは SETO 呼び出しの誤ったデータ・セット印刷オプションにより、動的割り振り中にエラーが生じることがあります。印刷データ・セット・オプションは、CHNG 呼び出しおよび SETO 呼び出しの処理中に構文解析されますが、*destination name* パラメーターなど、動的割り振り中にのみ妥当性検査が行われるものもあります。

IMS がデータ・セットへの最初の挿入を実行したときに、印刷オプションの 1 つが誤っていて、動的割り振りに失敗すると、IMS は **AX** 状況コードを ISRT 呼び出しに返します。さらに、IMS はメッセージ DFS0013E を出して診断ログ・レコード (67D0) を作成するので、これを問題を評価するのに使用できます。エラー・メッセージの形式は、呼び出されたサービスのタイプと、エラーに対応する戻りコードおよび理由コードを示します。エラー・メッセージは、次のサービスを示します。

DYN　MVS 動的割り振り (SVC99)

OPN　MVS データ・セットのオープン

OUT MVS 動的出力記述子の作成 (SVC109)

UNA MVS 動的割り振り解除 (SVC99)

WRT MVS BSAM 書き込み

DFS0013E メッセージが、このサービスのいずれかからのエラー戻りコードを示すときは、エラー・コードの詳細について、関連する MVS 資料を調べてください。動的割り振り、動的割り振り解除、動的出力記述子の作成に関するサービスについては、「*z/OS MVS Programming: Authorized Assembler Services Guide*」から、適切な戻りコードおよび理由コードを参照してください。

よくある誤りとして、IMSTEMP の宛先が JES に定義されていないときに、不適切な宛先を使用したり、整合性オプション 2 (選択できない宛先) を選択したりすることがあります。**destination name** パラメーターに不適切な宛先を指定すると、IMS が印刷データ・セットを割り振り解除するときに、呼び出しが動的割り振り解除エラーになります。

印刷データ・セットの動的出力について

IMS では、印刷データ・セットの動的出力 (SVC109) に対して z/OS サービスを使用できます。IMS はこのサービスを使用して、作成される印刷データ・セット用に、アプリケーションが提供する属性を指定します。このサービスは、PRTO、TXTU、および OUTN オプションの CHNG 呼び出しで使用されます。

関連資料**:** さらに詳しくは、「*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。

PRTO オプションを使用する CHNG 呼び出し

CHNG 呼び出しおよび PRTO オプションを使用するときは、IMS が SJF を活動化して、動的出力の z/OS サービスを呼び出す印刷オプションを検査します。こうして、印刷データ・セットが割り振られるときに使用される出力記述子が作成されます。アプリケーションで印刷データ・セット特性を入手するには、これが最も簡単な方法です。ただし、各 CHNG 呼び出しに対して構文解析の必要があるので、オーバーヘッドは最も大きくなります。アプリケーションが WFI であるか、または同じ印刷オプションで複数データ・セットを作成する場合は、他のオプションを使用して構文解析の影響を削減してください。このオプションでは、IAFP オプション・キーワードを指定する必要があります。

TXTU オプションを使用する CHNG 呼び出し

アプリケーションで動的出力に必要なテキスト単位を管理できる場合は、多くの印刷データ・セットの構文解析を回避することができます。これは、次の 2 つの方法のうちの 1 つで可能になります。

- アプリケーションが、必要な形式でアプリケーション域内にテキスト単位を作成し、CHNG 呼び出しおよび TXTU オプションを使用して、そのテキスト単位を IMS に渡す。
- アプリケーションが、SETO 呼び出しを使用して IMS に印刷オプションを提供し、テキスト単位を構成するために作業域を提供する。z/OS が構文解析とテキスト構成を終了すると、正常な SETO 呼び出しの後に、渡された作業域に、動的

出力に必要なテキスト単位が入ります。作業域にはアドレス依存情報が含まれているので、この作業域をアプリケーションが再配置すべきではありません。

アプリケーションがテキスト単位を管理する方式に関係なく、テキスト単位を再使用できるアプリケーションは、CHNG 呼び出しの TXTU オプションを使用して、パフォーマンスを高めることができます。

このオプションでは、IAFP オプション・キーワードを指定する必要があります。

OUTN オプションを使用する CHNG 呼び出し

従属領域の JCL には、OUTPUT JCL ステートメントを含めることができます。アプリケーションでこの方式を使用できる場合は、CHNG 呼び出しおよび OUTN オプションを使用して、OUTPUT JCL ステートメントを参照することができます。OUTN オプションを使用すると、IMS は、動的割り振りで OUTPUT JCL ステートメントを参照します。JES は、OUTPUT JCL ステートメントから印刷データ・セット特性を入手します。このオプションでは、IAFP オプション・キーワードを指定する必要があります。

スプール API を使用するサンプル・プログラム

スプール API によって、アプリケーション・プログラムは、IMS 固有のプリンターにデータを送るのと同じ手法で、IMS スプールにデータを書き込めるようになります。

スプール API には、不適切な OUTDES パラメーターに対するエラー検査などの機能があります。エラー検査機能によって、アプリケーション・プログラムはより複雑になります。このアプリケーション・プログラムを単純化するには、エラー情報を管理する共通ルーチンを開発し、スプール API からの診断情報を問題判別に使用できるようにします。

このセクションのサンプル・プログラムは、データを IMS スプールに送るために DL/I 呼び出しをコーディングする方法を示しています。アプリケーション・プログラムのうち、DL/I 呼び出しの形式を理解するために必要な部分だけが示してあります。この例は、アセンブラー言語で作成されています。

アプリケーション PCB 構造

アプリケーション PCB は次のようになっています。

I/O PCB

ALTPCB1

ALTPCB2

ALTPCB3

ALTPCB4

I/O PCB への GU 呼び出し

IMS アプリケーション・プログラムは、初期設定と、入力メッセージを獲得するための I/O PCB への呼び出しで始まります。次のサンプル・コードは、I/O PCB に GU 呼び出しを発行する方法を示しています。

I/O PCB への GU 呼び出しを完了した後、アプリケーション・プログラムは IMS スプール用に出力データを作成します。

I/O PCB に GU 呼び出しを発行する

```
*********************************************************************
*       ISSUE GU ON IOPCB                                           *
*********************************************************************
         L     9,IOPCB             I/O PCB ADDRESS
         LA    9,0(9)
         MVC   FUNC,=CL4'GU'       GU FUNCTION
         CALL  ASMTDLI,(FUNC,(9),IOA1),VL
         BAL   10,STATUS           CHECK STATUS
*        ADDITIONAL PROGRAM LOGIC HERE
FUNC     DC    CL4' '
IOA1     DC    AL2(IOA1LEN),AL2(0)
TRAN     DS    CL8                 TRANSACTION CODE AREA
DATA     DS    CL5                 DATA STARTS HERE
         DC    20F'0'
IOA1LEN  EQU   *-IOA1
```

代替 PCB への CHNG 呼び出し

他のプログラムが CHNG 呼び出しを使用して出力の宛先を指定するのと同じ方法で、このプログラムでは、出力宛先として IMS スプールを指定します。IMS 固有のプリンターに関しては、DEST NAME パラメーターが出力 LTERM 名を示します。 IAFP= キーワードを含む CHNG 呼び出しを発行するときに DEST NAME パラメーターを使用するのは、整合性オプション '2' が指定されている場合のみです。オプション '2' が指定されていない場合は、アプリケーション・プログラムが DEST NAME パラメーターを使用して、変更呼び出しを作成するルーチンのような別のものを識別します。印刷データ・セットの宛先は、初期設定パラメーターまたは OUTDES パラメーターの組み合わせを使用して設定されます。

次のサンプル・コードは、変更可能代替 PCB に CHNG 呼び出しを発行する方法を示しています。

CHNG 呼び出しが出された後で、アプリケーション・プログラムは、ISRT 呼び出しを発行して印刷データ・セットを作成します。

変更可能代替 PCB に CHNG 呼び出しを発行する

```
*********************************************************************
*       ISSUE CHNG ON ALTPCB4                                       *
*********************************************************************
         L     9,ALTPCB4           ALT MODIFIABLE PCB
         LA    9,0(9)              CLEAR HIGH BYTE/BIT
         MVC   FUNC,=CL4'CHNG'     CHNG FUNCTION
         CALL  ASMTDLI,(FUNC,(9),DEST2,OPT1,FBA1),VL
         BAL   10,STATUS           CHECK STATUS OF CALL
*        ADDITIONAL PROGRAM LOGIC HERE
FUNC     DC    CL4' '
DEST2    DC    CL8'IAFP1'          LTERM NAME
*
         DC    C'OPT1'             OPTIONS LIST AREA
OPT1     DC    AL2(OPT1LEN),AL2(0)
         DC    C'IAFP='
OCC      DC    C'M'                DEFAULT TO MACHINE CHAR
OOPT     DC    C'1'                DEFAULT TO HOLD
OMSG     DC    C'M'                DEFAULT TO ISSUE MSG
         DC    C','
         DC    C'PRTO='
```

```
PRTO1    EQU   *
         DC    AL2(PRTO1LEN)
         DC    C'COPIES(2),CLASS(T),DEST(RMT003)'
PRTO1LEN EQU   *-PRTO1
         DC    C' '
OPT1LEN  EQU   *-OPT1
*
FBA1     DC    AL2(FBA1LEN),AL2(0)
         DC    CL40' '
FBA1LEN  EQU   *-FBA1
```

代替 PCB への ISRT 呼び出し

IMS スプールが PCB の宛先として一度指定されると、ISRT 呼び出しを代替 PCB に対して出すことができます。

次のサンプル・コードは、変更可能代替 PCB に ISRT 呼び出しを発行する方法を示しています。

印刷データ・ストリームは、アプリケーション・プログラムの要件、および作成されるデータ・セットのタイプに応じて、データベースに保管されたり、アプリケーション・プログラムによって生成されたりします。

変更可能代替 PCB に ISRT 呼び出しを発行する

```
*************************************************************************
*       ISSUE ISRT TO ALTPCB4                                           *
*************************************************************************
         L     9,ALTPCB4           ALT MODIFIABLE PCB
         LA    9,0(9)              CLEAR HIGH BYTE/BIT
         MVC   FUNC,=CL4'ISRT'     ISRT FUNCTION
         CALL  ASMTDLI,(FUNC,(9),IOA2),VL
         BAL   10,STATUS           CHECK STATUS OF CALL
*        ADDITIONAL PROGRAM LOGIC HERE
FUNC     DC    CL4' '
IOA2     DC    AL2(IOA2LEN),AL2(0)
IOA21    DC    AL2(MSG2LEN),AL2(0)
         DC    C' '                CONTROL CHARACTER
         DC    C'MESSAGE TO SEND TO IMS SPOOL'
MSG2LEN  EQU   *-IOA21
IOA2LEN  EQU   *-IOA2
```

プログラムの終了

呼び出しを発行した後で、プログラムはメッセージを発信元端末に送り返し、I/O PCB に GU 呼び出しを発行するか、または正常に終了します。

第 27 章 IMS メッセージ形式サービス

IMS メッセージ形式サービス (MFS) は、端末装置への、または端末装置からのメッセージを形式設定する IMS Transaction Manager 環境の 1 つの機能です。この機能により、IMS アプリケーション・プログラムは、入力または出力メッセージで装置固有の特性を取り扱う必要がなくなります。

さらに、MFS は、リモート・コントローラーやサブシステムのユーザー作成プログラムへの (またはプログラムからの) メッセージも形式設定するので、アプリケーション・プログラムでは、リモート・コントローラーの送信固有の特性を取り扱う必要がなくなります。

MFS が使用するのは、入出力メッセージの配置形式を IMS に指示するためにユーザーが指定する制御ブロックです。

- 入力メッセージの場合。MFS 制御ブロックでは、装置からアプリケーション・プログラムに送られるメッセージをプログラムの入出力域にどのように配置するかを定義します。
- 出力メッセージの場合。MFS 制御ブロックでは、アプリケーション・プログラムから装置へ送られるメッセージを、画面やプリンターにどのように配置して出すかを定義します。リテラルのような、画面には表示されるがプログラムの入出力域には表示されないデータも定義することができます。

IMS Transaction Manager システムでは、アプリケーション・プログラムと端末間、またはアプリケーション・プログラムとリモート・プログラム間で受け渡されるデータは、MFS または基本編集機能を用いて編集することができます。アプリケーション・プログラムが MFS を使用するかどうかは、端末のタイプまたはネットワークが使用する 2 次論理装置 (SLU) によって決まります。

制約事項: MFS は LU 6.2 装置に対するメッセージ・フォーマット設定をサポートしません。

MFS 使用の利点

MFS の使用により、端末向けアプリケーションの開発および保守を簡略化でき、オンライン処理用の制御ブロックを使用してオンラインのパフォーマンスを向上させることができます。

開発と保守の簡略化

IMS アプリケーションの開発と保守を簡略化するため、MFS は多数のアプリケーション・プログラムに共通する機能を実行し、特定の装置やリモート・プログラムからのアプリケーション・プログラムの高度の独立性を実現します。

MFS により装置独立性が提供されるので、1 つのアプリケーション・プログラムは、複数の装置タイプとの間で、装置それぞれの各種の機能を使用しながらデータ

を送受して処理することができます。したがって、新しい端末タイプが追加された場合でも、MFS を使用していれば、アプリケーション・プログラムの変更箇所は最小で済みます。

MFS を使用すると、アプリケーション・プログラムは、メッセージの読み取りと作成の方法を変更しなくとも、別のタイプの端末と通信することができるようになります。アプリケーションが端末から受信するメッセージが、プログラムの入出力域にどのような形で入るかは、メッセージを送信した端末の種類とは無関係で、形式はプログラムで指定されている MFS オプションによって決まります。アプリケーションが、次に別のタイプの端末からメッセージを受け取る場合でも、アプリケーションに手を加える必要はありません。DB プログラム連絡ブロック (PCB) が、データベース内のデータの実際の形式や保管方法によってプログラムへ影響が及ばないようにしているのと同様に、MFS は、メッセージを送信している物理デバイスによってアプリケーションに影響が及ばないようにします。

MFS によって実行されるその他の共通機能としては、データの左寄せまたは右寄せ、埋め込み、妥当性検査のための出口ルーチン、時刻および日付スタンプ、ページおよびメッセージの番号付け、およびデータの順序付けとセグメント化があります。MFS がこれらの機能を受け持つ場合、アプリケーション・プログラムでは、メッセージ・データの実際の処理だけを行えばよいことになります。

次の図は、MFS によってアプリケーション・プログラムをどのように装置独立とすることができるかを示しています。その方法として、装置またはリモート・プログラムからの入力データは表示用にフォーマットして IMS に渡し、アプリケーション・プログラムのデータは表示用にフォーマットして出力装置またはリモート・プログラムに渡します。

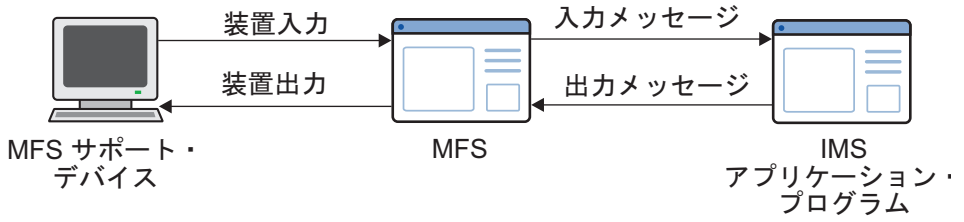


図 *82. MFS* を使用したメッセージのフォーマット設定

端末のオンライン・パフォーマンスの向上

MFS はまた、オンライン処理用に設計された制御ブロックを使用することで、端末向け IMS のオンライン・パフォーマンスを向上させています。 MFS 制御ブロックは、オフラインのとき、つまり IMS Transaction Manager システムが実行されていないときに、ソース言語の定義からコンパイルされます。MFS では、ソース言語定義の妥当性検査や多くの決定をオフラインで行うことにより、オンライン処理を軽減させています。さらに、MFS は、オンライン処理中に MFS 制御ブロックの検索バッファリングを使用して、CPU とチャネルの入出力アクティビティーによる負担を軽減します。

MFS 制御ブロックは再入可能で、複数のアプリケーションで使用できるため、必要となるオンライン・ストレージ要件が緩和されます。オプションの実記憶域索引付けおよび制御ブロックの先行取り出しを使用することによって、応答時間も短縮す

ることができます。IMS が z/OS 用に生成されていれば、MFS 形式ライブラリーから形式ブロックをロードするときに複数の入出力操作を並行して実行することができるので、さらにパフォーマンスが向上します。

また、MFS は z/OS ページング・サービスを使用するので、IMS 制御領域タスクのページ不在を減らすのに役立ちます。

MFS を使用すると、データを圧縮して必要なデータのみを送るので、通信回線の使用を減らすことができます。これにより、回線負荷が削減され、応答時間と装置パフォーマンスの両方が改善されます。

MFS 制御ブロック

アプリケーション・プログラムおよび端末やリモート・プログラムに対して入出力の形式を指定する MFS 制御ブロックには、以下の 4 種類があります。

4 つの種類は以下のとおりです。

メッセージ出力記述子 **(MOD)**
: MFS がアプリケーション・プログラムから受け取るメッセージのレイアウトを定義します。

装置出力形式 **(DOF)**
: プログラムが通信する各装置あてのメッセージの MFS によるフォーマット設定の仕方を定義します。

装置入力形式 **(DIF)**
: プログラムの通信相手の各装置から MFS が受け取るメッセージの形式を記述します。

メッセージ入力記述子 **(MID)**
: アプリケーション・プログラムがメッセージを処理できるように、MFS がさらに行う形式設定の仕方を記述します。

本書では、「メッセージ記述子」という場合は、MID と MOD の両方を指します。「装置形式」という場合は、DIF と DOF の両方を指します。

MOD、DOF、DIF、および MID はそれぞれ特定のメッセージを扱います。プログラムが送信する固有のメッセージごとに MOD と DOF が、プログラムが受信する固有のメッセージごとに DIF と MID が必要です。

MFS の例

MFS 制御ブロックの間の関係を理解する方法の 1 つとして、端末使用者がメッセージを入力した時点から、アプリケーション・プログラムがそれを処理して端末へ応答を送信する時点までの間のメッセージを観察する方法があります。MFS は表示端末でもプリンターでも使用することができますが、分かりやすいように、この例では表示端末を使用しています。

次の図は、MFS 制御ブロック間の関係を示しています。

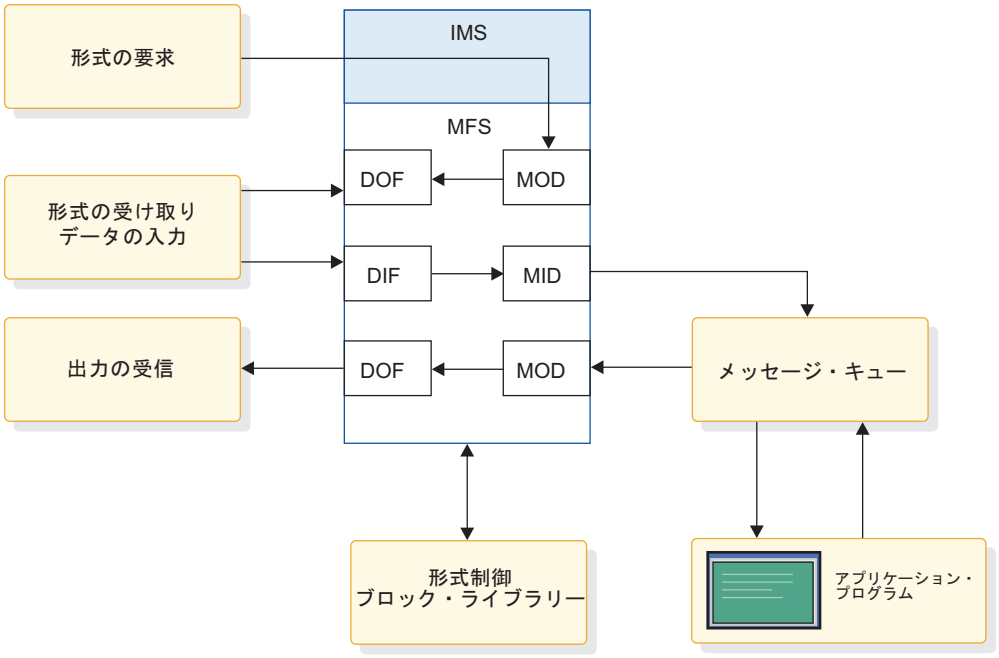


図 *83. MFS* 制御ブロックの関係

給与計算レコードの検討

インストール・システムに、従業員の給与計算レコードを表示するのに使用されるメッセージ処理プログラムがあるものとします。表示端末から IMS の形式設定コマンド (/FORMAT) と MOD 名を入力します。MFS プログラマーが作成した MOD によって定義されているとおりに、画面が形式設定されます。 MOD 名を入力した時点では、画面に表示されるのはリテラルだけで、アプリケーション・プログラムからの出力データは表示されません。この段階では、アプリケーション・プログラムはまだ関与していません。(/FORMAT の詳細については、「*IMS V15* コマンド 第 *1* 巻*: IMS* コマンド *A-M*」を参照してください。)

この例では、アプリケーション用の画面を形式設定する MOD の名前は、PAYDAY であるとします。以下のコマンドを入力します。

```
/FORMAT PAYDAY
```

IMS は PAYDAY という名の MFS MOD 制御ブロックを見つけて、DOF で定義した形式に画面を構成します。

次の図は、画面の表示内容を示しています。

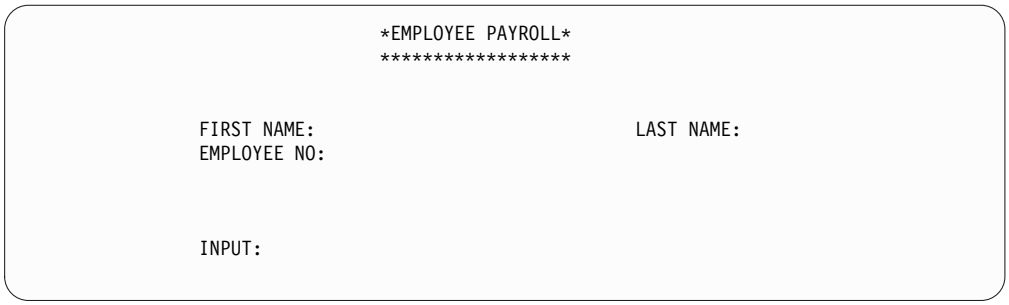


図 *84. PAYDAY* の画面 *(DOF* による形式設定*)*

DOF には、従業員の名前と従業員番号を端末から入力するよう要求する形式が定義されています。PAYUP は、この情報を処理するアプリケーションと関連しているトランザクション・コードです。MOD 名を入力すると、トランザクション・コードは、最初に表示される画面フォーマットの中に組み込まれます。つまり、端末使用者は、データを処理するプログラムの名前を知らなくても、画面を形式設定する MOD の名前さえ知っていればよいわけです。

画面がこの形式で表示された後で、情報を入力することができます。プログラムにメッセージを送信してその応答を受信するまでには、以下の 4 つの操作段階があります。

1. 端末から情報を入力します。この例では、プロンプトで要求された情報を入力します。

   次の図は、情報入力後のこの画面を示しています。

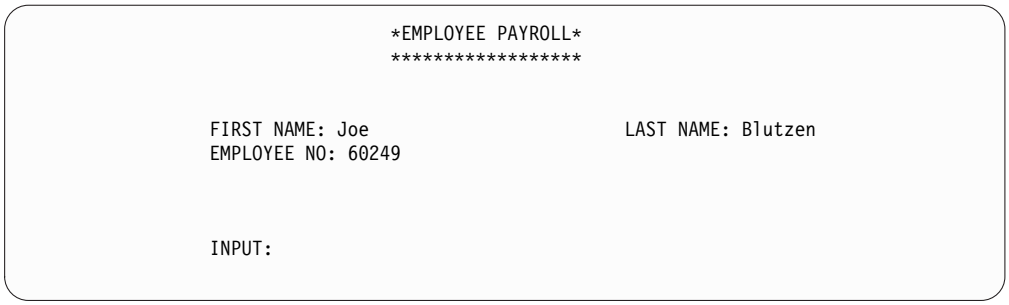


図 *85. PAYDAY* 画面 *(*各フィールドにデータ入力済み*)*

2. IMS がこのデータを受け取ると、MFS は DIF および MID 制御ブロックを使用して、データを、端末画面に入力された形式からアプリケーション・プログラムが期待している受け取り形式に変換します。DIF 制御ブロックは、端末からどのような形でデータが入ってくるかを、 MFS に知らせます。MID 制御ブロックは、アプリケーション・プログラムがどのような形でデータを受け取ることを要求しているかを、MFS に知らせます。アプリケーション・プログラムがメッセージ呼び出しを発行すると、IMS はプログラムの入出力域に「変換後」のメッセージを入れます。

   アプリケーション・プログラムが入出力域で受け取るメッセージは、以下のようになります。

   ```
   PAYUP JOE BLUTZEN 60249
   ```

「PAYUP」はトランザクション・コードです。論理端末の名前は、メッセージそのものには含まれていません。論理端末名は、IMS によって I/O PCB の最初のフィールドに入れられます。

3. アプリケーション・プログラムは、必要があればデータベース・アクセスも行ってメッセージを処理し、入出力域に出力メッセージを作成します。プログラムは、データベースから社会保障番号と給与額の情報をリトリーブしてから、従業員用の出力メッセージ・セグメントを作成します。アプリケーション・プログラムの入出力域には、以下のものが入ります。

   ```
   LLZZJOE BLUTZEN 60249532596381150.00
   ```

   LL は、MFS メッセージ内の 2 バイト・フィールドで、フィールドの長さを示します。LL フィールドの定義方法は、アプリケーション・プログラムを作成するために使用するプログラミング言語によって異なります。
   AIBTDLI、ASMTDLI、
   CEETDLI、または PASTDLI インターフェースの場合は、LL フィールドは 2 進数のハーフワードとして定義しなければなりません。PLITDLI インターフェースの場合は、LL フィールドは、2 進数のフルワードとして定義する必要があります。PLITDLI インターフェースに与えられる値は、実際のセグメント長から 2 バイト引いた長さでなければなりません。

   ZZ は、MFS メッセージ内の 2 バイト・フィールドで、ここには、アプリケーション・プログラムとの間で送受されるメッセージをフォーマット設定するときに使用される MFS フォーマット設定オプションが入ります。MFS オプションについて詳しくは、「*IMS V15* アプリケーション・プログラミング *API*」のトピック『入力メッセージ・フォーマット設定オプション』で説明しています。

4. アプリケーション・プログラムが端末にメッセージを送り返すとき、MFS は再びメッセージを変換します。今度は、アプリケーション・プログラムが送った形式から端末が要求しているデータ形式に変換します。

   MOD は、アプリケーション・プログラムの入出力域から送られてくるメッセージの形式を、MFS に知らせます。DOF は、端末画面にメッセージをどのような形式で表示したらよいかを、MFS に知らせます。MFS がメッセージを変換すると、IMS は変換されたメッセージを端末画面に表示します。

   次の図は、画面の表示内容を示しています。

```
                        *EMPLOYEE PAYROLL*
                        ******************


FIRST NAME: Joe                         LAST NAME: Blutzen
EMPLOYEE NO: 60249
SOC SEC NO: 532-59-6381
RATE OF PAY: $150.00


INPUT:
```

図 *86. PAYDAY* 画面 *(DOF* による形式設定の出力の表示*)*

従業員のサブセットのリスト作成

MPP で以下の要求に応えるものとします。

> スキル・レベル「3」の「ENGINEER」というスキルを持っている従業員のリストを作成する。その中から勤続 4 年以上の者のみをリストする。

表示端末からこの要求を入力するには、形式設定コマンド (/FORMAT) および MOD 名を出します。ユーザーが作成した MOD によって定義したとおりに、画面が形式設定されます。MOD 名を入力した時点では、画面に表示されるのはリテラルだけで、アプリケーション・プログラムからの出力データは表示されません。この段階では、MPP は関係していません。 この要求に対して画面を形式設定する MOD の名前を LE とします (「従業員を見つけだす (locate employee)」の略)。以下を入力します。

```
/FORMAT LE
```

IMS は LE という名の MFS MOD 制御ブロックを見つけて、DOF で定義した形式に画面を構成します。画面は以下のようになります。

```
SKILL
LEVEL
YEARS
      LOCEMP
```

DOF が定義する端末の形式は、必要とする従業員のスキル、レベル、および勤続年数を与えることによって、従業員に対する要求を限定するようにユーザーに要求するものです。LOCEMP は、この要求を処理できる MPP に関連するトランザクション・コードです。 MOD 名を入力すると、トランザクション・コードは、最初に表示される画面フォーマットの中に組み込まれます。つまり、端末使用者は、要求を処理するプログラムの名前を知らなくても、画面を形式設定する MOD の名前さえ知っていればよいわけです。

画面がこの形式で表示された後で、要求を入力することができます。プログラムにメッセージを送信して応答を受信するまでには、以下の 4 つの操作段階があります。

1. 端末から情報を入力します。この例では、IMS が画面に表示している資格の値、つまり、スキルは「ENG」(エンジニア)、スキル・レベルは「3」、勤続年数は「4」を入力します。

   要求に応えて入力すると、画面にはこのデータが表示されます。

   ```
   SKILL ENG
   LEVEL 3
   YEARS 4
         LOCEMP
   ```

2. IMS がこのデータを受け取ると、MFS は DIF および MID 制御ブロックを使用して、データを、端末画面に入力した形式からアプリケーション・プログラムが要求している形式に変換します。DIF 制御ブロックは、端末からどのような形でデータが入ってくるかを、MFS に知らせます。MID 制御ブロックは、アプリケーション・プログラムがどのような形でデータを受け取ることを要求しているかを、MFS に知らせます。アプリケーション・プログラムが I/O PCB に GU 呼び出しを発行すると、IMS はプログラムの入出力域に「変換後」のメッセージを入れます。

MPP が入出力域にこのメッセージを受信すると、メッセージは以下のようになります。

LOCEMP ENG0304

「LOCEMP」はトランザクション・コードです。論理端末の名前は、メッセージそのものには含まれていません。論理端末名は、IMS によって I/O PCB の最初のフィールドに入れられます。

3. MPP は、必要があればデータベース・アクセスも行ってメッセージを処理し、MPP の入出力域に出力メッセージを作成します。

   複数の従業員がこの資格を満たしているとします。 MPP は、従業員 1 人につき 1 つのメッセージ・セグメントを使用することができます。プログラムは、データベースから情報をリトリーブしてから、1 人目の従業員用の出力メッセージ・セグメントを作成します。プログラムの入出力域には、以下のものが入ります。

   ```
   LLZZJONES,CE   3294
   ```

   プログラムが 2 番目のセグメントを送ると、入出力域には以下のものが入ります。

   ```
   LLZZBAKER,KT   4105
   ```

4. アプリケーション・プログラムが端末にメッセージを送り返すとき、MFS は再びメッセージを変換します。今度は、アプリケーション・プログラムが送った形式から端末が要求しているデータ形式に変換します。

   MOD は、アプリケーション・プログラムの入出力域から送られてくるメッセージの形式を、MFS に知らせます。DOF は、端末画面にメッセージをどのような形式で表示したらよいかを、MFS に知らせます。MFS がメッセージを変換すると、IMS は変換されたメッセージを端末画面に表示します。画面には以下のようなデータが表示されます。

   ```
   SKILL    ENG
   NAME     NO
   JONES,CE 3294
   BAKER,KT 4105
   ```

関連概念:

『MFS 制御ブロックと画面フォーマットの関係』

MFS 制御ブロックと画面フォーマットの関係

MFS ソース言語で制御ブロックを使用して、装置での表示形式を定義することができます。

エンド・ユーザーやオペレーターが画面の初期形式を受け取るには、MOD の名前を指定して /FORMAT コマンドを出すことにより初期形式を要求するというのが標準的な方法です。次のサンプル・コードでは、MOD のラベルは PAYDAY になっています。この MOD には、同じラベルの装置出力形式 (DOF) を指しているパラメーターの SOR=PAYF が入っています。

最初の DOF も装置の入力形式となります。つまり、DOF で DIV TYPE=INOUT と指定すると、装置入力形式 (DIF) も生成されます。サンプル・コードの PAYF

は、次の入力の形式も記述しているため、DOF と DIF の両方を兼ねています。最終の出力メッセージは、出力だけのために指定された形式で表示され、DIF は生成されません。

MOD と MID はともに同じ DOF を指しており、装置関連の制御ブロックとメッセージ関連の制御ブロックとを関連させています。

出力の場合、MFS は、MOD で定義されているフィールドを、DOF で定義されている画面上のフィールドに移します。MOD でフィールド定義 (MFLD) をコーディングするときに、ラベルを指定します。その同じラベルを、DOF で装置フィールド (DFLD) をコーディングするときに使用し、フィールドが表示される画面上の位置を定義します。

MFS は、データ・フィールドを出力メッセージから画面フィールドへ移動させます。これをマッピング といいます。入力では、MFS は DIF と MID の中の同じラベルが付いたフィールドをマッピングさせることによって、画面上で変更されたフィールドをプログラムへの入力メッセージのデータ・フィールドへ移動させます。

これらの制御ブロックの指定方法の詳細については、「*IMS V15* データベース・ユーティリティー」を参照してください。

MFS 制御ブロックは、MFS 言語ユーティリティーの実行中に、次のサンプル・コードで示すようなソース・ステートメントから生成されます。制御ブロックは各種の MFS ライブラリーに保管されます。

サンプル・コードは 3270 表示装置用に設計されています。

MFS 制御ブロックのサンプル・コード

DOF/DIF

```
PAYF     FMT
         DEV       TYPE=(3270,2),FEAT=IGNORE,DSCA=X'00A0'
         DIV       TYPE=INOUT
         DPAGE     CURSOR=((5,15))
         DFLD      '**********************',POS=(1,21)
         DFLD      '*  EMPLOYEE PAYROLL  *',POS=(2,21)
         DFLD      '**********************',POS=(3,21)
         DFLD      'FIRST NAME:',POS=(5,2)
FNAME    DFLD      POS=(5,15),LTH=16
         DFLD      'LAST NAME:',POS=(5,36)
LNAME    DFLD      POS=(5,48),LTH=16
         DFLD      'EMPLOYEE NO:',POS=(7,2)
EMPNO    DFLD      POS=(7,16),LTH=6
         DFLD      'SOC SEC NO:',POS=(9,2)
SSN      DFLD      POS=(9,15),LTH=11
         DFLD      'RATE OF PAY: $',POS=(11,2)
RATE     DFLD      POS=(11,17),LTH=9
         DFLD      'INPUT:',POS=(16,2)
INPUT    DFLD      POS=(16,10),LTH=30
         FMTEND
```

MID

```
PAYIN    MSG       TYPE:INPUT,SOR=(PAYF,IGNORE)
         SEG
         MFLD      'PAYUP '     SUPPLIES TRANCODE
         MFLD      LNAME,LTH=16
         MFLD      FNAME,LTH=16
```

```
        MFLD      EMPNO,LTH=6
        MFLD      SSN,LTH=11
        MFLD      RATE,LTH=9
        MFLD      INPUT,LTH=30,JUST=R,FILL=C'0'
        MSGEND
```

MOD

```
PAYDAY  MSG       TYPE:OUTPUT,SOR=(PAYF,IGNORE)
        SEG
        MFLD      LNAME,LTH=16
        MFLD      FNAME,LTH=16
        MFLD      EMPNO,LTH=6
        MFLD      SSN,LTH=11
        MFLD      RATE,LTH=9
        MFLD      INPUT,LTH=30,JUST=R,FILL=C'0'
        MSGEND
```

関連資料:

551 ページの『MFS の例』

MFS のコンポーネントの概要

IMS メッセージ形式サービス (MFS) のコンポーネントには、メッセージ・エディターと 2 つのプール・マネージャーの、3 つのユーティリティーが含まれています。

MFS ユーティリティー

MFS ユーティリティーは、以下のような複数のサービスおよび生成を目的として使用できます。

- MFS 装置特性テーブル・ユーティリティー (DFSUTB00): IMS システム定義を完了せずに、IMS.PROCLIB ライブラリーの記述子メンバーに新規画面サイズを定義します。
- MFS 言語ユーティリティー (DFSUPAA0): MFS 制御ブロックの作成および保管を行います。
- MFS サービス・ユーティリティー (DFSUTSA0): MFS 中間テキスト・ブロックと制御ブロックの処理および保管が MFS 言語ユーティリティー (DFSUPAA0) によって完了した後で、これらのブロックの制御および保守を行います。

MFS ユーティリティーを使用して MFS ライブラリーを更新できるほか、IMS オンライン変更機能も使用できます。IMS 制御領域の実行中は、制御ブロック・ライブラリーを変更することができます。

MFS メッセージ・エディター

MFS メッセージ・エディターを使用して、制御ブロックの仕様に従ってメッセージを形式設定します。この制御ブロック仕様は、ユーザーが入力した制御ステートメント定義から MFS 言語ユーティリティーが生成したものです。

MFS プール・マネージャー

以下の MFS プール・マネージャーの機能をカスタマイズできます。

- MFS プール・マネージャー: MFS は、参照されたブロックをストレージ内に残しておくことにより、形式ライブラリーへの入出力を最小限にしようとします。

このストレージは、MFS プール・マネージャーで管理されます。 MFS サービス・ユーティリティーの INDEX 機能を使用すると、指定された形式ブロック用のディレクトリー・アドレスのリストを作成することによって、この機能をカスタマイズすることができます。このリストにより、IMS は、ブロックを取り出す前にデータ・セット・ディレクトリーを読み取る必要がなくなります。

- MFSTEST プール・マネージャー: MFSTEST 機能を使用すると、MFS 制御ブロックは MFSTEST プール・マネージャーによって管理されます。MFS テスト用の通信回線バッファー・プール・スペースは、システム定義時に指定しますが、IMS 制御領域の初期設定時にスペースを変更することもできます。このスペース値は、ある一時点で MFSTEST ブロック用として使える最大量を表しています。スペース値はプールの予約済み部分ではありません。

関連概念:

- オンライン変更機能 (システム管理)
- MFS コンポーネント (コミュニケーションおよびコネクション)
- メッセージ形式バッファー・プールの使用 (システム定義)

関連資料:

- MFS 言語ユーティリティー (DFSUPAA0) (システム・ユーティリティー)
- MFS サービス・ユーティリティー (DFSUTSA0) (システム・ユーティリティー)
- MFS 装置特性テーブル・ユーティリティー (DFSUTB00) (システム・ユーティリティー)

MFS で作動する装置および論理装置

MFS は、3270 装置の他に、3600 および 4700 金融機関通信システム (FIN)、3770 データ通信システム、3790 通信システム、2 次論理装置 (SLU) タイプ 1、2、6、および P で作動します。ネットワーク端末オプション (NTO) 装置は、2 次論理装置のタイプ 1 コンソールとしてサポートされています。

次の表は、IMS システムで MFS 操作用に定義するときに、装置番号 (例えば 3270 など) で定義できる装置または論理装置、および割り当てられている論理装置のタイプ (例えば SLU 1 など) で定義できる装置または論理装置を示しています。

3600 装置は FIN シリーズに含まれていますが、36xx という名称で指定することもできます。ユーザーがどの名称を指定しても、MFS メッセージでは FIxx という名称が使用されます。しかし一般的には、この情報を読まれるアプリケーション設計者およびプログラマーは、インストール・システムの IMS システムに対して装置がどのように定義されているかを知っていれば、その装置の制御ブロックを定義するのには十分です。

表 *85. MFS* で作動する端末装置：

装置	番号で定義する装置[1]	NTO 装置[2]	SLU 1	SLU 2	SLU P	LU 6.1
3180	X[3]			X[3]		

表 *85. MFS* で作動する端末装置 *(*続き*)*:

装置	番号で定義する装置[1]	NTO 装置[2]	SLU 1	SLU 2	SLU P	LU 6.1
3270	X[3]			X[3]		
3290	X[3]			X[3]		
5550	X[3]			タイプ: 3270-An 3270-Ann		
3270 プリンター: 5553、5557	X[3]		COMPT*n*= MFS-SCS1			
3730					X	
3767			COMPT*n*= MFS-SCS1			
3770 コンソール、プリンター、印刷データ・セット			COMPT*n*= MFS-SCS1		X	
3770 読取装置、穿孔装置、送信データ・セット			COMPT*n*= MFS-SCS2		X	
3790 印刷データ・セット (大量)			COMPT*n*= MFS-SCS1		COMPT*n*= MFS-SCS1 DPM-An	
3790 送信データ・セット			COMPT*n*= MFS-SCS2			
3790 接続 3270				X[3]		
6670						
8100					X	
8100 接続 3270			X	X[3]		
8100 接続シリーズ /1					X	
8100 接続 S/32			X			
8100 接続 S/34					X	
8100 接続 S/38			X			
金融機関	X				COMPT*n*= MFS-SCS1 DPM-An	
TTY		X				
3101		X				
その他のシステム (IMS から IMS、IMS からその他)						COMPT*n*= DPM=Bn

注**:**

1. options= (...,MFS,...) により定義します。 (TERMINAL または TYPE マクロ。)
2. TYPE マクロの UNITYPE= および TERMINAL マクロの PU= を用いて定義します。

3. デフォルトでは、MFS で作動するように定義されます。

論理装置は、論理装置タイプのみ、TERMINAL マクロの COMPT*n*= または TYPE= と論理装置タイプ、または ETO ログオン記述子で定義します。 LU 6.1 定義は、ISC サブシステムを指しています。

SLU 1 についての定義では、SNA 文字ストリング (SCS) 1 または 2 を用いて MFS 操作を指定することができます。 SCS1 は、メッセージをプリンターまたは印刷データ・セットに送るか、あるいはプログラマブル 3770 または 3790 制御装置ディスク記憶装置のキーボードからメッセージを受信することを指定します。 SCS2 は、カード入出力または送信データ・セットとの間でメッセージを送受信するという指定です。

SLU 2 として定義される端末は、3270 と類似の特性を持っており、3270 同様 MFS と共に作動するように定義しておくことができます。3290 端末は 3270 端末とおおむね同じように作動するので、この情報の 3270 端末についての記述は 3290 端末についても適用されます。ただし、3290 の分割化およびスクロールのサポートを受けられるのは、IMS に SLU 2 として定義された 3290 装置のみです。

3180 端末と 5550 端末は 3270 端末とおおむね同じように作動するので、3270 端末についての記述はこれらの装置についても適用されます。同様に、5553 プリンターと 5557 プリンターは 3270P と同じように作動します。

制約事項**:** 5550 漢字のサポート対象は、SLU 2 として定義された 5550 端末と、SCS1 プリンターとして定義された 5553 および 5557 のみです。

IMS が 3790 または FIN 制御装置でユーザー作成リモート・プログラムと通信する場合、装置を SLU P として定義する必要があります。SLU P の定義では、MFS-SCS1 または DPM-An として MFS 操作を指定しなければなりません。ここで、DPM は分散表示管理を、 An はユーザーが割り当てた番号 (A1 から A15) をそれぞれ表します。

現在他の装置で使用できる MFS フォーマット設定機能のほとんどは、特定の装置フォーマット設定についての例外はありますが、ユーザー作成プログラムでも使用することができます。ユーザーが管理を行えば、これらのフォーマット設定機能 (ページングなど) は、MFS とリモート・プログラムとに分散させることができます。

分散表示管理 (DPM) の使用

分散表示管理 (DPM) を行うと、MFS が通常実行するフォーマット設定機能が、MFS と SLU P 装置または ISC ノードで使うユーザー作成プログラムの間で分散されます。3790 または FIN 制御装置が、装置番号であらかじめ IMS に定義されている場合は、DPM に移行するために多少の変更を加えなければなりません。

DPM では、2 次論理装置の物理端末特性を MFS に定義する必要がありません。MFS がフォーマット設定する必要があるのは、リモート・コントローラーまたは ISC ノードにあるユーザー・プログラムに送信するメッセージだけです。MFS では、装置フォーマット設定を完成させる責任があり、また必要に応じて、選択した物理デバイスへデータを送ります。

DPM を使用するリモート・プログラムでは、MFS とリモート・プログラムの間でのデータ・ストリームの受け渡しを装置から独立させることができます。IMS アプリケーション・プログラムから出すメッセージには、いくつかの装置制御文字を入れることができます。そのようにした場合は、IMS アプリケーション・プログラムと、リモート・プログラムへのデータ・ストリームとは、装置からの独立性を失うことになります。

IMS と通信を行う他のサブシステム (IMS, CICS またはユーザー作成のサブシステム) は、ISC サブシステムとして定義しておかなければなりません。ISC の定義は、必ず以下のようになります。

- MFS 操作を DPM-Bn と指定する。ここで Bn は、ユーザーが割り当てた番号 (B1 から B15) です。
- システム定義時に、TERMINAL マクロで TYPE:LUTYPE6 を定義する。

ISC を使用した場合、DPM は以下のようなサービスを提供します。

- 要求時出力ページング。これにより、IMS と他のシステムとにページングを分散させることができます。
- 自動ページ化単位出力。これにより、MFS ページを、ページング要求を介さずに別のシステムに送信することができます。
- トランザクション経路指定。これにより、アプリケーション・プログラムは、入力メッセージに経路指定情報が入っているときには、その情報を見ることができます。

第 28 章 サービスまたはデータへのコールアウト要求

IMS アプリケーションは、サービスまたはデータに対してコールアウト要求を出すことができ、オプションで、IMS Connect および OTMA を経由して、同じトランザクションまたは別のトランザクションで応答を受信できます。このサービスまたはデータに対する要求をコールアウト要求 と呼びます。

要求を出した後に IMS アプリケーションが従属領域で応答を待つ場合、その要求は同期コールアウト要求 になります。要求が出された後に IMS アプリケーションが終了し、従属領域で応答を待たない場合、その要求は非同期コールアウト要求 になります。IMS アプリケーションは、同期プログラム間通信を使用して、IMS トランザクションに対する同期コールアウト要求を発行することもできます。

各タイプの要求は、以下のように処理されます。

同期コールアウト要求
: IMS 従属領域で稼働する IMS アプリケーション・プログラムは DL/I ICAL 呼び出しを実行し、応答を処理するためにその従属領域で待機します。アプリケーション・プログラムは、DL/I ICAL 呼び出しのオプションの制御データ域 を使用して、ルーティング・データ、セキュリティー・データ、またはその他のデータを IMS Connect とそのクライアントに渡すことができます。DL/I ICAL 呼び出しが出されると、IMS は同期コールアウト要求に対して相関トークンを生成します。この相関トークンはコールアウト要求に含まれています。要求側 IMS アプリケーション・プログラムに応答を返すよう経路指定するために、相関トークンは応答とともに IMS に返される必要があります。

非同期コールアウト要求
: IMS 従属領域で稼働する IMS アプリケーション・プログラムはコールアウト要求を ALTPCB キューに挿入 (ISRT ALTPCB 呼び出し) した後に終了して、従属領域を解放します。IMS は、非同期コールアウト要求については相関トークンを生成しません。コールアウト要求への応答が必要な場合、そのコールアウト要求への応答の相関は、IMS アプリケーション・プログラムが管理する必要があります。IMS は、非同期コールアウト要求への応答を受信すると、その応答を新しいトランザクションとして処理します。

同期プログラム間通信要求
: 同期プログラム間通信では ICAL 呼び出しを使用しますが、要求は外部サーバーではなく IMS トランザクションあてに送付されます。同期プログラム間通信要求では、IMS が応答を待機中のアプリケーション・プログラムに自動的に関連付けるため、相関トークンを使用しません。

以下の表は、同期コールアウト要求、非同期コールアウト要求、および同期プログラム間通信要求の違いを要約したものです。

表 86. 同期コールアウト要求と非同期コールアウト要求の比較

コールアウト処理	同期コールアウト要求	非同期コールアウト要求	同期プログラム間通信要求
OTMA 保留キューへの要求の配置	要求側の IMS アプリケーションは、制御データの有無に関係なく、ICAL 呼び出しを出します。	要求側の IMS アプリケーションは ISRT ALTPCB 呼び出しを出します。	要求側の IMS アプリケーションは ICAL 呼び出しを出します。
要求発行後の IMS アプリケーションの状況	アプリケーションは従属領域で応答を待機します。従属領域はブロックされます。	アプリケーションは終了します。	アプリケーションは従属領域で応答を待機します。従属領域はブロックされます。
メッセージ処理の取り扱い	メッセージ処理は IMS OTMA によって行われます。	メッセージ処理は IMS メッセージ・キューによって行われます。	メッセージ処理は IMS メッセージ・キューによって行われます。
応答の取り扱い	応答は、相関トークンに基づいて、同じ作業単位の間に返されて要求側の IMS アプリケーションに相関します。	応答がある場合、要求側か別の IMS アプリケーションが、別のトランザクションで返される応答を処理するようコーディングされている必要があります。トランザクションの作業単位は、非同期出力のフローのためにコミットする必要があります。	応答は、同じ作業単位中に要求側の IMS アプリケーションに返されます。ただし、ターゲット・トランザクションは、別の作業単位で実行されるため、2 フェーズ・コミットには適格ではなく、元のアプリケーションの RRS コミットの有効範囲の一部ではありません。

関連資料:

ICAL 呼び出し (アプリケーション・プログラミング API)

コールアウト要求の方法

IMS Enterprise Suite SOAP Gateway、IMS TM Resource Adapter、 IBM MQ、ユーザー提供の独自の IMS Connect クライアント・アプリケーション、または他の IMS アプリケーション・プログラムに、コールアウト要求を発行できます。

同期コールアウト要求の場合、オプションで、制御データ をコールアウト・メッセージに組み込むことができます。制御データを使用すると、ポートの URL、UUID、ユーザー・トークン、セキュリティー情報、またはその他の情報を IMS Connect とそのクライアントに渡すことができます。

SOAP Gatewayの使用

SOAP Gatewayは、IMS アプリケーションから一般の Web サービスに対してコールアウト要求を発行するために使用します。

SOAP Gatewayにより、IMS アプリケーションは、Web サービス・プロバイダーまたは Web サービス・コンシューマーのいずれかとして機能できます。 SOAP Gatewayは、Web サービス・コンシューマーとしての IMS アプリケーションに対して、非同期コールアウトと同期コールアウトの両方の方法をサポートしています。SOAP Gatewayのツール・サポートは、IBM Developer for System z® で使用

可能です。これを使用すると、IMS Connect と通信するための接続情報と対話情報、および、IMS アプリケーションの言語構造に基づいて、必要な Web サービス成果物を生成できます。また、SOAP Gatewayは、デプロイメント・ユーティリティーを提供し、Web サービスのプロバイダーまたはコンシューマーとしての IMS アプリケーションの配置をサポートします。

SOAP Gateway メッセージの場合、タグ <DFSCNVTR>*CONVERTER_NAME*</DFSCNVTR> を使用して、制御データに XML コンバーター名を指定することができます。コンバーター名とタグは大文字の EBCDIC である必要があります。コンバーター名が存在する場合、IMS Connect がメッセージの処理に使用する可能性のあった現行コンバーター名をオーバーライドします。コンバーター名が抽出されるとすぐに、そのメッセージについてそれ以上のタグのスキャンは発生しなくなります。 XML コンバーター名が含まれているタグは、その制御データから除去されることはなく、制御データ内にある他のタグと一緒に SOAP Gateway に送信されます。

以下の例は、2 つの制御データ項目を持つ制御データ域を示しています。

```
Total Length = 4 + 10 + 8 + 11 + 4 + 9 + 25 + 10 = 81 = X'51'

AIBOPLEN = X'00000051'

Control Data = X'00000021' <DFSCNVTR>CONVERT1</DFSCNVTR> X'00000030'
<USERTAG>USER DATA CAN BE ANYTHING</USERTAG>
```

SOAP Gateway を使用した IMS アプリケーション・コールアウトの使用可能化について詳しくは、Web サービス・コンシューマーとしての IMS アプリケーションの使用可能化を参照してください。

IMS TM Resource Adapterの使用

IMS アプリケーションから同期または非同期のコールアウト要求を発行するには、IMS TM Resource Adapterのバージョン 10 (またはそれ以降) を使用します。この要求の宛先は、メッセージ駆動型 Bean (MDB)、Enterprise JavaBeans (EJB) コンポーネント、Java EE (以前の J2EE) アプリケーション、または Web サービスです。

IMS TM Resource Adapterを使用すると、Java EE アプリケーションは、インターネット上で IMS トランザクションにアクセスできるだけでなく、IMS 従属領域で実行される IMS アプリケーションから外部の Java EE アプリケーションにコールアウト要求を発行することもできます。 IMS TM Resource Adapterには、WebSphere Application Server 用のランタイム・コンポーネントが含まれています。 IMS TM Resource Adapterのツール・サポートは、IBM Rational® Application Developer for WebSphere Software で使用可能です。さらに、J2EE コネクター (J2C) ウィザードが組み込まれている Rational および WebSphere のさまざまな統合開発環境 (IDE) でも使用可能です。

IMS TM Resource Adapterでのコールアウト・サポートについて詳しくは、コールアウト・プログラミング・モデルを参照してください。

IBM MQの使用

IBM MQ 経由でその他のアプリケーションに非同期コールアウト要求を行うアプリケーションを作成できます。IBM MQ を IMS ブリッジに構成する必要があります。

また、TYPE=MQSERIES を指定して IBM MQ の宛先記述子を作成する必要があります。記述子は、IMS PROCLIB データ・セットの DFSYDT*x* メンバーに追加するか、CREATE OTMADESC コマンドを使用して追加することができます。 OTMA ルーティング出口 (DFSYPRX0 および DFSYDRU0) は必要ありません。

IBM MQ の宛先記述子タイプには、MQMD データ構造のフィールドのカスタマイズに使用可能なパラメーターが含まれています。MQMD 構造では、IBM MQ が記述子を使用するメッセージを処理する方法を制御します。

ユーザー作成 IMS Connect TCP/IP アプリケーションの使用

ユーザーは独自の IMS Connect TCP/IP アプリケーションを作成することも、コールアウト要求をリトリーブするために TCP/IP および IMS Connect プロトコルを使用するベンダー提供ソリューションを使用することもできます。ユーザーのカスタム IMS Connect クライアント・アプリケーションは、OTMA 宛先記述子で定義されている OTMA ルーティング宛先 (トランザクション・パイプ (TPIPE) とも呼ばれる) への RESUME TPIPE 呼び出しを出す必要があります。この TPIPE は、コールアウト要求を保留します。ユーザーのカスタム IMS Connect TCP/IP アプリケーションは、TPIPE をポーリングしてコールアウト要求をリトリーブしなければなりません。

IMS 同期プログラム間通信要求の使用

ICAL 呼び出しを使用すると、TYPE=IMSTRAN を指定して宛先記述子を作成することで、別の IMS アプリケーションに要求を送信できます。このタイプのコールアウト要求が、同期プログラム間通信です。

IMSTRAN 記述子タイプでは、宛先トランザクションを指定します。このタイプは、遅延応答メッセージ・キューの作成に使用することもできます。記述子は、IMS PROCLIB データ・セットの DFSYDT*x* メンバーに追加するか、CREATE OTMADESC コマンドを使用して追加することができます。宛先アプリケーションで同期プログラム間通信に ICAL 呼び出しも発行する場合は、複数の同期プログラム間通信要求を一緒にチェーニングすることができます。

同期プログラム間通信に ICAL 呼び出しを使用する場合に、OTMA は必要ありません。宛先トランザクションの OTMA 宛先記述子を定義する必要があります。ただし IMS は OTMA がアクティブであるかどうかに関わらずトランザクションをスケジュールに入れます。

制御データは、同期プログラム間通信要求についてはサポートされません。

Java 従属領域リソース・アダプターの使用

Java 従属領域リソース・アダプターは、Java Message Service (JMS) インターフェースをサポートします。Java アプリケーションはこのインターフェースを使用して、Java 従属領域で同じ機能を実行することができます。

IMS™ は、IMS Java 従属領域で稼働する Java アプリケーションを開発するための、IMS Java 従属領域リソース・アダプターと呼ばれる一連の Java™ API 呼び出しを提供します。

IMS Java 従属領域リソース・アダプターは、JMP または JBP 領域で稼働する Java アプリケーション・プログラムに、メッセージ処理プログラム (MPP) および非メッセージ・ドリブン BMP 領域に提供されているものと同様の、以下のような DL/I 機能を提供します。

- メッセージの読み取りおよび書き込みのための、IMS メッセージ・キューへのアクセス
- プログラム間通信の実行
- コミットおよびロールバック処理
- GSAM データベースへのアクセス
- データベース・リカバリー (CHKP/XRST)

IMS トランザクションが Java で作成されていて、IMS Universal Java 従属領域 (JDR) リソース・アダプターを有効に活用すると、制御データを含む ICAL 呼び出しを発行できます。JDR リソース・アダプターは、Java Native Interface (JNI) 呼び出しを使用して Universal Driver の C ライブラリー (DFSCLIBU) を呼び出し、ICAL 情報を持つ AIBTDLI インターフェースに対して C から呼び出しを発行します。

IMS Java 従属領域リソース・アダプターをタイプ 2 IMS Universal JDBC ドライバーまたはタイプ 2 IMS Universal DL/I ドライバーと一緒に使用して、GSAM データベース・アクセスなどのデータベース操作を実行します。

関連概念:

IMS アプリケーション・プログラムからのコールアウト要求 (コミュニケーションおよびコネクション)

OTMA 宛先記述子 (コミュニケーションおよびコネクション)

関連資料:

IMS PROCLIB データ・セットの DFSYDTx メンバー (システム定義)

ICAL 呼び出し (アプリケーション・プログラミング API)

RESUME TPIPE プロトコル

RESUME TPIPE プロトコルは、IMS から非同期コールアウト・メッセージおよび同期コールアウト・メッセージをリトリーブします。

IMS Connect クライアントは、IRM_TIMER フィールドで IRM タイムアウト値を指定することにより、IMS からの出力を待機する時間を通知します。IRM タイムア

ウト値は、IMS Connect に送信される RESUME TPIPE 呼び出しと、IMS Connect に送信される ACK 応答メッセージまたは NAK 応答メッセージに影響を与えます。

IMS TM Resource Adapterまたは IMS Enterprise Suite SOAP Gatewayを使用して IMS アプリケーションからのコールアウト要求を処理する際、IMS Connect との通信が処理されます。

IMS TM Resource Adapterと SOAP Gatewayの両方は、連続して RESUME TPIPE 呼び出しを IMS Connect に対して実行することによって、同期コールアウト要求を listen します。コールアウト要求メッセージが TPIPE キューにある場合、OTMA はそのコールアウト要求を IMS Connect に送信します。IMS Connect はそのメッセージを処理して、必要に応じて XML へ変換します (SOAP Gateway の場合で、IMS Connect XML アダプター機能を使用しているとき)。そして、そのメッセージを IMS TM Resource Adapterまたは SOAP Gateway に送信します。

カスタム IMS Connect クライアントを使用する場合、コールアウト・メッセージをリトリーブするために RESUME TPIPE 呼び出しを出すように、そのクライアントをコーディングする必要があります。

RESUME TPIPE セキュリティー

リソース・アクセス管理機能 (RACF) または OTMA Resume TPIPE セキュリティーのユーザー出口 (OTMARTUX)、あるいはその両方を使用して、RESUME TPIPE 呼び出しの無許可の使用からコールアウト・メッセージを保護することができます。

セキュリティーが有効なときには、OTMA クライアントにメッセージが送信される前に、RESUME TPIPE 呼び出しを出すユーザー ID には、RESUME TPIPE 呼び出しメッセージに入っている TPIPE 名にアクセスする権限が与えられなければなりません。

RACF が行うセキュリティー検査および OTMARTUX ユーザー出口が行うセキュリティー検査は任意です。RACF と OTMARTUX の両方を使用する場合は、OTMARTUX ユーザー出口に制御が与えられる前に、最初に RACF が呼び出されます。この場合、OTMARTUX ユーザー出口は、必要に応じて RACF をオーバーライドすることができます。

同期コールアウト機能の実装

使用している IMS アプリケーションから同期コールアウト要求を出すには、ICAL 呼び出しを出して OTMA 記述子名を指定します。

ICAL 呼び出しは、REXXTDLI 呼び出しによっても出すことができ、また、JMP 領域または JBP 領域で実行される Java アプリケーションから出すこともできます。オプションで、タイムアウト値 (応答が戻るまで待機する最大時間) を指定することもできます。

IMS からの入出力メッセージは、同期コールアウト要求では、セグメントごとに 32 KB 以上になる場合があります。ただし、同期プログラム間通信要求の場合、最大セグメント・サイズは 32 KB です。

次の図に、同期コールアウト機能のメッセージ・フローを示します。要求は、ICAL 呼び出しを出す IMS アプリケーションで開始します。応答は、要求側の IMS アプリケーションに返されます。

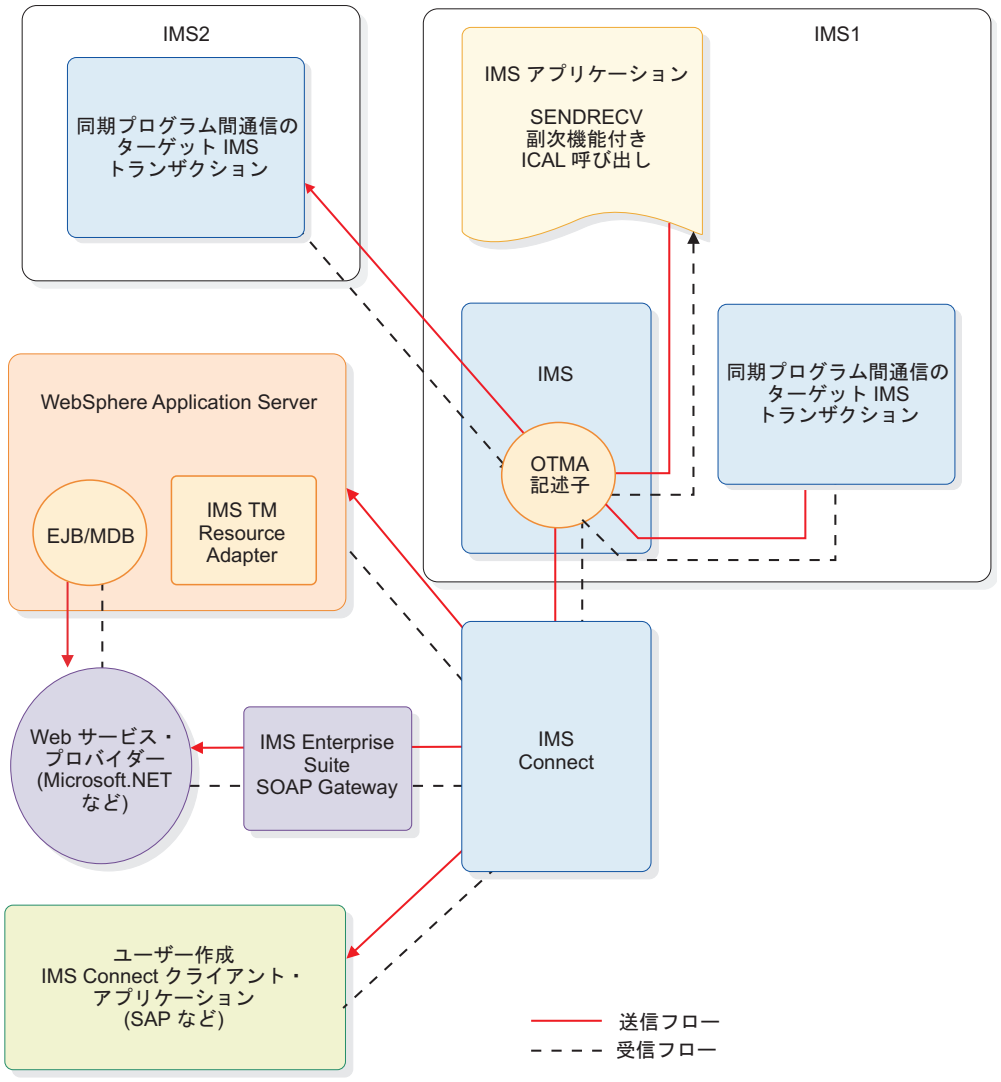


図 87. 同期コールアウト機能のメッセージ・フロー

IMS アプリケーションから、同じまたは別の IMS 接続の宛先に経路指定された ICAL 呼び出しを同時に出すことができます。 RESUME TPIPE 呼び出しおよび送信専用プロトコルを使用することで、同期コールアウト機能は別の接続やスレッドでの要求の送信や応答の受信を可能にします。コールアウト要求のリトリーブ用に 1 つの接続を保持し、別の接続を使用して応答メッセージを同時に返すことができます。

相関トークンは、応答メッセージを正しい IMS トランザクション・インスタンスに返して相関させるために、IMS によって作成されます。コールアウト要求をプルするためおよび応答メッセージを返すために異なるスレッドや接続を保持できることで、最大の並行性が得られます。

同期プログラム間通信要求では、相関トークンは使用されません。

アプリケーションは、RECEIVE サブ関数コードを指定した ICAL 呼び出しを実行することで、以前に SENDRECV サブ関数コードを指定して不完全に実行された ICAL 呼び出しの完全な応答データを取得することができます。ただし、続けて SENDRECV サブ関数を指定した呼び出しが発行された場合、その後でアプリケーションが RECEIVE サブ関数呼び出しを指定しても完全な応答データをリトリーブすることはできません。続けて SENDRECV サブ関数を指定した ICAL 呼び出しが発行されると、IMS 制御領域内にある前回の ICAL 呼び出しの ICAL 応答バッファーは消去されます。

以下の概略的な手順では、同期コールアウトのアプリケーションおよび機能の、実装とデプロイメントの概要を示しています。

1. ICAL 呼び出し用の IMS アプリケーションを作成または変更します。
2. 以下のいずれかのコールアウト要求アプローチ用に、OTMA 宛先記述子を定義します。
   - IMS Enterprise SuiteIMS TM Resource Adapter経由でコールアウト要求を外部アプリケーションに送信する。
   - IMS Enterprise Suite SOAP Gateway経由でコールアウト要求を外部アプリケーションに送信する。
   - ユーザーが作成した IMS Connect クライアント・アプリケーションにコールアウト要求を送信する。
   - 別の IMS アプリケーションに同期プログラム間通信要求を送信する。

   宛先記述子は、IMS.PROCLIB データ・セットの DFSYDTx メンバーで定義するか、CREATE OTMADESC コマンドを使用して定義することができます。
3. 新しく定義された OTMA 記述子のために IMS を再始動します。 CREATE OTMADESC コマンドを使用して記述子が動的に追加された場合は、このステップは必須ではありません。
4. ステップ 1 で作成または変更した IMS アプリケーションを実行して、同期コールアウト要求を発行します。

IMS アプリケーションは、同期コールアウト・ターゲットから応答メッセージを受け取ります。応答メッセージが割り振られた応答域に収まらない場合、アプリケーションは、使用可能な応答データ域を拡張して、RECEIVE サブ関数を指定した ICAL 呼び出しを発行することで、完全な応答を取得することができます。

同期コールアウト機能の COBOL プログラムの実装例

COBOL プログラムで ICAL 呼び出しを出すには、CALL ステートメントを使用します。

```
CALL 'AIBTDLI' USING ICAL, AIB, CA-REQUEST, SCA-RESPONSE.
```

次の例は、COBOL プログラムにおける ICAL 呼び出しの必須 AIB フィールド宣言を示しています。完全な COBOL の例 (パーツ名 DFSSSCBL を持つ) は、SDFSSMPL サンプル・ライブラリーのコールアウト IVP サンプルで提供されています。

```
01 AIB.
   02 AIBRID PIC x(8)  VALUE 'DFSAIB  '.
   02 AIBRLEN PIC 9(9) USAGE BINARY.
   02 AIBSFUNC PIC x(8) VALUE 'SENDRECV'.
```

```
   02 AIBRSNM1 PIC x(8) VALUE 'OTMDEST1'.
   02 AIBRSNM2 PIC x(8).
   02 AIBRESV1 PIC x(8).
   02 AIBOALEN PIC 9(9) USAGE BINARY VALUE 28.
   02 AIBOAUSE PIC 9(9) USAGE BINARY VALUE 30.
   02 AIBRSFLD PIC 9(9) USAGE BINARY VALUE 5000.
   02 AIBRESV2 PIC x(8).
   02 AIBRETRN PIC 9(9) USAGE BINARY.
   02 AIBREASN PIC 9(9) USAGE BINARY.
   02 AIBERRXT PIC 9(9) USAGE BINARY.
 ...
```

次の例は、COBOL プログラムにおける CA-REQUEST 宣言および SCA-RESPONSE 宣言を示します。

```
* ICAL Request Area
 01  CA-REQUEST.
     02  CA-MESSAGE   PIC  X(45) VALUE SPACES.

* ICAL Response Area
 01  SCA-RESPONSE.
     02  SCA-MESSAGE  PIC  X(100) VALUE SPACES.
```

関連概念:

- OTMA 宛先記述子 (コミュニケーションおよびコネクション)

関連タスク:

- コールアウト要求用の IMS アプリケーションの変更

846 ページの『Java 従属領域からの同期コールアウト要求の実行』

関連資料:

- ICAL 呼び出し (アプリケーション・プログラミング API)
- DL/I 呼び出し機能の例 (アプリケーション・プログラミング API)
- コールアウト・プログラミング・モデル

同期コールアウト要求の制御データ

DL/I ICAL 呼び出しを同期コールアウト要求に使用する場合、IMS アプリケーション・プログラムは、実行時に呼び出しを発行する際に、ICAL 呼び出しの制御データ域にコールアウト・メッセージのエンドポイント情報やその他の経路指定情報を指定できます。

IMS アプリケーション・プログラムは、ICAL DL/I 呼び出しを使用して同期コールアウト要求を作成することができます。ICAL 呼び出しは IMS Connect クライアント・アプリケーションにコールアウト要求を送信して、応答を受け取ります。コールアウト・メッセージの経路の指定は、メッセージのアプリケーション・インターフェース・ブロック (AIB) および OTMA 宛先記述子で定義されます。最大 4095 個の宛先記述子を IMS PROCLIB データ・セットの DFSYDTx メンバー内で定義できます。メッセージのコールアウト・エンドポイントが数千に及ぶ場合、限られた数の OTMA 宛先記述子項目を経路の指定に使用することは難しい問題になります。また、コールアウト・メッセージに、複数の Universally Unique Identifier (UUID)、SOAP ヘッダー、セキュリティー・トークン、あるいはユーザー提供の経路指定情報ですら容易に含めることができません。

この問題を解決するために、ICAL 呼び出し形式では、オプションの制御データ を使用できます。これは、ポートの URL、UUID、ユーザー・トークン、セキュリティー情報、またはその他のどのような情報でもかまいません。制御データのフィールドを使用して、IMS アプリケーション・プログラムは実行時に ICAL 呼び出しを発行する際に、経路指定情報やその他の制御データを指定できます。

制御データは、1 対多の制御データ項目から構成することができるため、同じ同期コールアウト呼び出しで多数のサービスや操作を指定できます。それぞれの制御データ項目は、4 バイトの長さフィールドから始まり、タグ、データ、および終了タグが続きます。

ICAL 制御データは、1 対多の制御データ項目から構成することができるため、同じ ICAL 呼び出しで多数のサービスや操作を指定できます。それぞれの制御データ項目は、4 バイトの長さフィールドから始まり、タグ、データ、および終了タグが続きます。タグの長さは任意です。開始タグは、「より小」記号 (<)、タグ名、および「より大」記号 (>) からなります。終了タグは、「より小」記号 (<)、スラッシュ (/)、開始タグ名と一致するタグ名、および「より大」記号 (>) からなります。

ICAL 制御データの制御データ項目のフォーマットは、次のとおりです。

```
LLLL  <tag1> data1 </tag1>  { LLLL <tag2> data2 </tag2> ... }
```

IBM が提供する制御データ項目は、タグ内の DFS で始まります。例えば、<DFSCNVTR> などです。

タグ名とデータ内容はバイナリー・データとして扱われ、ターゲット・クライアントに「現状のまま」渡されます。<、/、および > の各記号と、IBM が提供する制御データ・タグ名 (DFS で始まるもの) は、EBCDIC であることが必要です。

OTMA は制御データの「整形式」検査を行い、制御データが、長さの正しいサポートされている形式に従っているかどうかを確認します。制御データ項目を持つ制御データの合計長は、AIB 内の AIBOPLEN フィールドに指定されている必要があります。OTMA はアプリケーション・データの前に制御データを置き、コールアウト・メッセージのアプリケーション・データ・セクション内の制御データのセグメント数を示すために OTMA 接頭部を更新します。OTMA の Resume Tpipe プロトコル・コマンドは、制御データ・オプションを持つコールアウトをサポートします。

IMS Connect ユーザー・メッセージ出口は、クライアントが制御データをサポートするかどうかを示す、IRM 内のフラグの有無を検査します。フラグがセットされている場合は、Resume Tpipe の OTMA ヘッダーの状態データ・セクションに適切なフラグがセットされます。IMS から制御データが入っているメッセージを受信すると、ユーザー・メッセージ出口はメッセージのアプリケーション・データ・セクションから制御データを抽出し、コールアウト相関トークン・セグメントによく似たセグメントを作成します。これは、IMS Connect 用の IMS TM Resource Adapter メッセージ以外のメッセージについて行われます。その後、ユーザー・メッセージ出口からの結果としてのメッセージは、IMS Connect によってクライアントへ送信されます。

SOAP Gateway を使用している場合は、制御データを使用して、要求の処理に使用したい XML コンバーターの名前を指定できます。制御データ内では、<DFSCNVTR> タグを使用して XML コンバーター名を指定します。

IMS Java 従属領域 (JDR) サポートは、ICAL 呼び出しの制御データ用の API を備えています。

制御データは、アウトバウンド・メッセージのコールアウト要求用に設計されています。応答メッセージ内の制御データはサポートされていません。

非同期コールアウト機能の実装

IMS アプリケーションから非同期コールアウト要求を出すには、ISRT ALTPCB 呼び出しを出して OTMA 宛先記述子名、または DFSYPRX0 および DFSYDRU0 経路指定出口ルーチンを指定します。

コールアウト要求に対して IMS に返されるすべての応答は、新しい着信トランザクションとして処理されます。応答がある場合、要求側アプリケーションか別の IMS アプリケーションが、別のトランザクションで返される応答を処理するようコーディングされている必要があります。

同期コールアウト要求と異なり、非同期コールアウト要求では、要求を出す IMS アプリケーション・プログラムが従属領域で応答を待つ必要はありません。アプリケーション・プログラムは、非同期コールアウト要求を出した後、終了して従属領域を解放することができます。コールアウト要求に対して IMS に返されるすべての応答は、新しい着信トランザクションとして扱われ、IMS はそれを処理するための新しいアプリケーション・プログラム・インスタンスをスケジュールします。

しかし、非同期コールアウト要求が応答を生成する場合、従属領域を解放して得られる利点は、その応答を管理する複雑さが加わることで相殺されることがあります。非同期コールアウト応答では、ご使用のシステムが、応答を元の要求に相関させる方法を作成する責任を負います。同期コールアウト要求では、IMS がこの相関を管理します。

以下の概略的な手順では、非同期コールアウトのアプリケーションおよび機能の、実装とデプロイメントの概要を示しています。

1. 非同期コールアウト応答の相関に関する計画を立てます。
2. 非同期コールアウト要求のための ISRT ALTPCB 呼び出しを出すために、IMS アプリケーションを作成または変更します。
3. コールアウト・ルーティング情報を定義します。必要な情報を定義する場合、以下の 2 つのオプションがあります。
   - OTMA ルーティング記述子を定義する。
   - DFSYPRX0 および DFSYDRU0 出口ルーチンをコーディングする。

   ルーティング記述子、出口ルーチン、またはその両方の組み合わせを使用して、コールアウト要求をどのようにルーティングするかを指定することができます。
4. オプション: 新しく定義された OTMA 記述子のために IMS を再始動します。再始動は、IMS.PROCLIB データ・セットの DFSYDTx メンバー内で OTMA

ルーティング記述子を作成または変更した場合にのみ必要です。 CREATE OTMADESC または UPDATE OTMADESC コマンドを使用した場合は、IMS を再始動する必要はありません。

5. コールアウト要求を出す IMS アプリケーションを実行します。 IMS アプリケーションは、通常、端末などの開始側クライアントを経由して、あるいは、IMS Connect クライアントか OTMA クライアントを経由して起動されます。

関連概念:

非同期コールアウト要求 (コミュニケーションおよびコネクション)

OTMA 宛先記述子 (コミュニケーションおよびコネクション)

関連資料:

OTMA ユーザー・データ・フォーマット設定出口ルーチン (DFSYDRU0) (出口ルーチン)

OTMA 宛先解決ユーザー出口 (DFSYPRX0 およびその他の OTMAYPRX タイプの出口) (出口ルーチン)

第 4 部 EXEC DLI によるアプリケーション・プログラミング

IMS は、EXEC DLI を使用して IMS リソースにアクセスするアプリケーションの作成をサポートしています。

第 29 章 EXEC DLI によるアプリケーション・プログラムの作成

EXEC DLI コマンドを実行して IMS にアクセスするプログラムは、アセンブラー言語、COBOL、PL/I、C、および C++ で作成できます。

プログラミングのガイドライン

効率的でエラーのない EXEC DL/I プログラムを作成するには、以下のガイドラインに従ってください。

プログラムが出す DL/I 要求の数、タイプ、および順序は、プログラムの能率に影響します。 プログラムの設計が悪くても、正しくコーディングされているかぎり、プログラムは動作します。 下記の推奨事項は、アプリケーション・プログラムに対する最も効率の良い設計開発を行うのに役立ちます。効率の悪い設計のプログラムはパフォーマンスに悪影響を及ぼし、またそのプログラムの変更も難しくなります。特定のコマンドまたは呼び出しの組み合わせがパフォーマンスにどのように影響するかを理解することによって、そのような問題の回避や、より効率の良いプログラムの設計に役立ちます。

プログラムについて一般的な呼び出し順序を計画したら、以下の指針を使用して、計画した順序をさらに改善してください。通常、要求順序が効率の良いものであれば、内部 DL/I 処理も効率良く行われます。

- 最も単純な呼び出しを使用してください。 要求を修飾して DL/I の検索範囲を狭めますが、必要以上に修飾しないでください。
- 必要なセグメントまでの最短パスを DL/I に指定する要求または一連の要求を使用してください。
- プログラムでは最小限の要求しか使用しないでください。プログラムが DL/I 要求を出すたびに、システム時刻とリソースが使用されます。不必要な呼び出しは、次の処置を行うことにより削除できます。
  - 同じパス内の 2 つ以上のセグメントの置き換え、リトリーブ、または挿入を行う場合には、パス要求を使用してください。複数の要求を使用してこのような操作を行うと、不必要な要求を出すことになります。
  - プログラムが個別の入出力域にセグメントを保管し、次にこのセグメントが必要になる時にはこの入出力域からセグメントを取り出せるように、順序を変更してください。 プログラムの実行中に、プログラムが同じセグメントを複数回リトリーブしている場合は、不必要な要求を出していることになります。
  - GB、GE、および II 状況コードが戻されるような不必要で非生産的な要求をあらかじめ取り除いておきます。例えば、特定のセグメント・タイプに対して GN を出す際に、そのセグメント・タイプのオカレンスの数が分かっている場合は、GN は出さないでください。GE 状況コードが発生する原因となります。 プログラムがリトリーブしたオカレンスの数を記録しておくことができるため、そのセグメント・タイプのすべてのオカレンスがリトリーブされたことが分かった時点で別の処理を継続することができます。

  - 各親セグメントの存在を確認するには、読み取り要求を出すのではなく、親セグメントについて修飾した挿入要求を出してください。 セグメントを挿入する場合、親セグメントが存在しないと従属セグメントを挿入することはできません。 DL/I が GE 状況コードを戻すときは、親セグメントが 1 つも存在していないということです。
- プログラム論理の主要セクションは一緒にしておいてください。 例えば、エラー・ルーチンおよび印刷ルーチンなどの条件付きルーチンへは、正常処理を続けるためにプログラムの別の部分において分岐し、それらの周囲で分岐しないでください。
- 呼び出しは、データの物理的な配置を効果的に利用できる順序で出してください。セグメントは、できるかぎり階層順でアクセスしてください。階層を逆方向に移動しないでください。
- データベース・レコードの処理は、ルート・セグメントのキー・フィールドの順序で行ってください。 (HDAM データベースの場合、この順序は使用されるランダム化ルーチンによって異なります。この情報については、DBA にお尋ねください。)
- プログラムの論理、およびコマンドまたは呼び出しの構造を、データベースの構造に大きく依存した方法で構築しないでください。階層の現行構造によっては、プログラムの柔軟性が損なわれます。

アセンブラー言語でのプログラム・コーディング

次のアセンブラー言語プログラムによるサンプル・プログラムは、コマンド・レベル・プログラムのさまざまな部分が整合して機能する様子、および CICS オンライン・プログラムにおける EXEC DLI コマンドのコーディング方法を示しています。

一部のコマンドを除き、このプログラムは、バッチ、BMP、CICS プログラムに適用されます。違いはいずれも、サンプル・アセンブラー・コードの注で強調して示してあります。サンプル・コードの右側の番号は、このような注を指しています。

```
*ASM XOPTS(CICS,DLI)
*                                                          1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R11      EQU   11
R12      EQU   12
R13      EQU   13
DFHEISTG DSECT
SEGKEYA  DS    CL4
SEGKEYB  DS    CL4                                         2
SEGKEYC  DS    CL4
SEGKEY1  DS    CL4
SEGKEY2  DS    CL4
CONKEYB  DS    CL8
SEGNAME  DS    CL8
SEGLEN   DS    H
PCBNUM   DS    H
AREAA    DS    CL80
AREAB    DS    CL80                                        3
AREAC    DS    CL80
AREAG    DS    CL250
AREASTAT DS    CL360
*        COPY  MAPSET
*
```

```
*****************************************************************
*  INITIALIZATION
*  HANDLE ERROR CONDITIONS IN ERROR ROUTINE                          4
*  HANDLE ABENDS (DLI ERROR STATUS CODES) IN ABEND ROUTINE
*  RECEIVE INPUT MESSAGE
*****************************************************************
*
SAMPLE   DFHEIENT CODEREG=(R2,R3),DATAREG=(R13,R12),EIBREG=R11     5
*
         EXEC CICS HANDLE CONDITION ERROR(ERRORS)                  6
*
         EXEC CICS HANDLE ABEND LABEL(ABENDS)                      6
*
         EXEC CICS RECEIVE MAP ('SAMPMAP') MAPSET('MAPSET')        6
*        ANALYZE INPUT MESSAGE AND PERFORM NON-DLI PROCESSING
*
*****************************************************************
*  SCHEDULE PSB NAMED 'SAMPLE1'
*****************************************************************
*
         EXEC DLI SCHD PSB(SAMPLE1)                                7
         BAL   R4,TESTDIB          CHECK STATUS
*
*****************************************************************
*  RETRIEVE ROOT SEGMENT AND ALL ITS DEPENDENTS
*****************************************************************
*
         MVC   SEGKEYA,=C'A300'                                    8
         EXEC DLI GU USING PCB(1) SEGMENT(SEGA) INTO(AREAA)          X
               SEGLENGTH(80) WHERE(KEYA=SEGKEYA) FIELDLENGTH(4)
         BAL   R4,TESTDIB          CHECK STATUS
GNPLOOP  EQU   *
         EXEC DLI GNP USING PCB(1) INTO(AREAG) SEGLENGTH(250)
         CLC   DIBSTAT,=C'GE'      LOOK FOR END                    9
         BE    LOOPDONE            DONE AT 'GE'
         BAL   R4,TESTDIB          CHECK STATUS
         B     GNPLOOP
LOOPDONE EQU   *
*
*****************************************************************
*  INSERT NEW ROOT SEGMENT
*****************************************************************
*
         MVC   AREAA,=CL80'DATA FOR NEW SEGMENT INCLUDING KEY'
         EXEC DLI ISRT USING PCB(1) SEGMENT(SEGA) FROM(AREAA)        X
               SEGLENGTH(80)
         BAL   R4,TESTDIB          CHECK STATUS
*
*****************************************************************
*  RETRIEVE 3 SEGMENTS IN PATH AND REPLACE THEM
*****************************************************************
*
         MVC   SEGKEYA,=C'A200'
         MVC   SEGKEYB,=C'B240'
         MVC   SEGKEYC,=C'C241'
         EXEC DLI GU USING PCB(1)                                    X
               SEGMENT(SEGA) WHERE(KEYA=SEGKEYA)                     X 10
               FIELDLENGTH(4)                                        X
               INTO(AREAA)                                           X
               SEGLENGTH(80)                                         X
               SEGMENT(SEGB) WHERE(KEYB=SEGKEYB) FIELDLENGTH(4)      X
               INTO(AREAB)                                           X
               SEGLENGTH(80)                                         X
               SEGMENT(SEGC) WHERE(KEYC=SEGKEYC) FIELDLENGTH(4)      X
               INTO(AREAC)                                           X
               SEGLENGTH(80)
         BAL   R4,TESTDIB
```

```
*        UPDATE FIELDS IN THE 3 SEGMENTS
         EXEC DLI REPL USING PCB(1)                                    X
               SEGMENT(SEGA) FROM(AREAA) SEGLENGTH(80)                 X
               SEGMENT(SEGB) FROM(AREAB) SEGLENGTH(80)                 X
               SEGMENT(SEGC) FROM(AREAC) SEGLENGTH(80)
         BAL   R4,TESTDIB          CHECK STATUS
*
*****************************************************************
*  INSERT NEW SEGMENT USING CONCATENATED KEY TO QUALIFY PARENT
*****************************************************************
*
         MVC   AREAC,=CL80'DATA FOR NEW SEGMENT INCLUDING KEY'
         MVC   CONKEYB,=C'A200B240'
         EXEC DLI ISRT USING PCB(1)                                    X
               SEGMENT(SEGB) KEYS(CONKEYB) KEYLENGTH(8)                X
               SEGMENT(SEGC) FROM(AREAC) SEGLENGTH(80)
         BAL   R4,TESTDIB          CHECK STATUS
*
*****************************************************************
*  RETRIEVE SEGMENT DIRECTLY USING CONCATENATED KEY
*  AND THEN DELETE IT AND ITS DEPENDENTS
*****************************************************************
*
         MVC   CONKEYB,=C'A200B230'
         EXEC DLI GU USING PCB(1)                                      X
               SEGMENT(SEGB)                                           X
               KEYS(CONKEYB) KEYLENGTH(8)                              X
               INTO(AREAB) SEGLENGTH(80)
         BAL   R4,TESTDIB          CHECK STATUS
         EXEC DLI DLET USING PCB(1)                                    X
               SEGMENT(SEGB) SEGLENGTH(80) FROM(AREAB)
         BAL   R4,TESTDIB          CHECK STATUS
*
*****************************************************************
*  RETRIEVE SEGMENT BY QUALIFYING PARENT WITH CONCATENATED KEY,
*  OBJECT SEGMENT WITH WHERE OPTION USING A LITERAL,
*  AND THEN SET PARENTAGE
*
*  USE VARIABLES FOR PCB INDEX, SEGMENT NAME, AND SEGMENT LENGTH
*****************************************************************
*
         MVC   CONKEYB,=C'A200B230'
         MVC   SEGNAME,=CL8'SEGA'
         MVC   SEGLEN,=H'80'
         MVC   PCBNUM,=H'1'
         EXEC DLI GU USING PCB(PCBNUM)                                 X
               SEGMENT((SEGNAME))                                      X
               KEYS(CONKEYB) KEYLENGTH(8) SETPARENT                    X
               SEGMENT(SEGC) INTO(AREAC) SEGLENGTH(SEGLEN)             X
               WHERE(KEYC='C520')
         BAL   R4,TESTDIB          CHECK STATUS
*
*****************************************************************
*  RETRIEVE DATABASE STATISTICS
*****************************************************************
*
         EXEC DLI STAT USING PCB(1) INTO(AREASTAT)                     X
               VSAM FORMATTED LENGTH(360)
         BAL   R4,TESTDIB          CHECK STATUS
*
*****************************************************************
*  RETRIEVE ROOT SEGMENT USING BOOLEAN OPERATORS
*****************************************************************
*
         MVC   SEGKEY1,=C'A050'
         MVC   SEGKEY2,=C'A150'
         EXEC DLI GU USING PCB(1) SEGMENT(SEGA) INTO(AREAA)            X
```

```
              SEGLENGTH(80) FIELDLENGTH(4,4,4,4)                    X
              WHERE(KEYA > SEGKEY1 AND KEYA < SEGKEY2
              KEYA > 'A275' AND  KEYA < 'A350')
         BAL   R4,TESTDIB          CHECK STATUS
*
*****************************************************************
*  TERMINATE PSB WHEN DLI PROCESSING IS COMPLETED
*****************************************************************
*
         EXEC DLI TERM                                              11
*
*****************************************************************
*  SEND OUTPUT MESSAGE
*****************************************************************
*
         EXEC CICS SEND MAP('SAMPMAP') MAPSET('MAPSET')              6
         EXEC CICS WAIT TERMINAL
*
*****************************************************************
*  COMPLETE TRANSACTION AND RETURN TO CICS
*****************************************************************
*
         EXEC CICS RETURN                                           12
*
*****************************************************************
*  CHECK STATUS IN DIB
*****************************************************************
*
TESTDIB  EQU   *
         CLC   DIBSTAT,=C'  '      IS STATUS BLANK                  13
         BER   R4                  YES - RETURN
*        HANDLE DLI STATUS CODES REPRESENTING EXCEPTIONAL CONDITIONS
*
         BR    R4                  RETURN
ERRORS   EQU   *
*        HANDLE ERROR CONDITIONS
*
ABENDS   EQU   *
*        HANDLE ABENDS INCLUDING DLI ERROR STATUS CODES
*
         END
```

サンプル・アセンブラー・コードに対する注

1 EXEC DLI コマンドを含む CICS オンライン・プログラムの場合、DLI オプションと CICS オプションを指定しなければなりません。EXEC DLI コマンドを含むバッチ・プログラムまたは BMP プログラムの場合は、 DLI オプションのみを指定しなければなりません。

2 再入可能にするため、DFHEISTG に、プログラムが使用する各区域 (すなわち、入出力域、キー・フィードバック域、およびセグメント名域) を定義します。

3 (単一のコマンドで) リトリーブ、追加、または置き換えを行うセグメントごとに入出力域を定義します。

4 EXEC DLI を含むバッチまたは BMP プログラムの場合、z/OS のレジスター保管規則に従って、入り口でレジスターを保管し、出口でレジスターを復元しなければなりません。

5 バッチまたは BMP プログラムでは、オプションの DFHEIENT を指定した DFHEIRET が入り口でレジスターを保管します。バッチ・プログラムでは、EIBREG パラメーターを指定しないでください。

6 バッチ・プログラムまたは BMP プログラムでは、EXEC CICS コマンドをコーディングしないでください。

7 CICS オンライン・プログラムでは、SCHD PSB コマンドを使用して、プログラムで使用する PSB を取得します。バッチ・プログラムまたは BMP プログラムでは PSB をスケジュールしてはなりません。

8 この GU コマンドは、キー値が A300 の SEGA が最初に現れる位置をリトリーブします。アセンブラー言語プログラムでは、KEYLENGTH または SEGLENGTH オプションを指定する必要はありません。

9 この GNP コマンドは、セグメント SEGA の従属セグメントをすべてリトリーブします。 GE 状況コードは、従属セグメントがそれ以上ないことを示します。

10 この GU コマンドは、パス・コマンドの例です。 セグメントごとに、別々の入出力域を使用してください。

11 CICS オンライン・プログラムでは、先にスケジュールされた PSB が、TERM コマンドにより終了します。バッチ・プログラムまたは BMP プログラムでは、PSB を終了させてはなりません。

12 バッチまたは BMP プログラムの場合、EXEC CICS RETURN の代わりに、RCREG パラメーターをコーディングします。 RCREG パラメーターは、戻りコードが入るレジスターを識別します。

13 各コマンドを出した後、DIB の状況コードを検査する必要があります。

COBOL でのプログラム・コーディング

次の COBOL によるサンプル・プログラムは、コマンド・レベル・プログラムのさまざまな部分が整合して機能する様子、および CICS オンライン・プログラムにおける EXEC DLI コマンドのコーディング方法を示しています。

一部のコマンドを除き、このプログラムは、バッチ、BMP、CICS プログラムに適用されます。違いはいずれも、サンプル COBOL コードの注で強調して示してあります。サンプル・コードの右側の番号は、このような注を指しています。

```
CBL LIB,APOST,XOPTS(CICS,DLI)        IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE.                                          1
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
.*     SOURCE-COMPUTER. IBM-370.
.*     OBJECT-COMPUTER. IBM-370.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77  SEGKEYA               PIC X(4).
       77  SEGKEYB               PIC X(4).                          2
       77  SEGKEYC               PIC X(4).
       77  SEGKEY1               PIC X(4).
       77  SEGKEY2               PIC X(4).
       77  SEGKEY3               PIC X(4).
       77  SEGKEY4               PIC X(4).
       77  CONKEYB               PIC X(8).
       77  SEGNAME               PIC X(8).
       77  SEGLEN                COMP PIC S9(4).
       77  PCBNUM                COMP PIC S9(4).
       01  AREAA                 PIC X(80).
      *    DEFINE SEGMENT I/O AREA
       01  AREAB                 PIC X(80).
       01  AREAC                 PIC X(80).                         3
       01  AREAG                 PIC X(250).
```

```
 01  AREASTAT              PIC X(360).
*    COPY MAPSET.
 PROCEDURE DIVISION.
*
* ***************************************************************
*  INITIALIZATION
*  HANDLE ERROR CONDITIONS IN ERROR ROUTINE
*  HANDLE ABENDS (DLI ERROR STATUS CODES) IN ABEND ROUTINE
*  RECEIVE INPUT MESSAGE
* ***************************************************************
*
     EXEC CICS HANDLE CONDITION ERROR(ERRORS) END-EXEC.           4
*
     EXEC CICS HANDLE ABEND LABEL(ABENDS) END-EXEC.               4
*
     EXEC CICS RECEIVE MAP ('SAMPMAP') MAPSET('MAPSET') END-EXEC. 4
*    ANALYZE INPUT MESSAGE AND PERFORM NON-DLI PROCESSING
*
* ***************************************************************
*  SCHEDULE PSB NAMED 'SAMPLE1'
* ***************************************************************
*
     EXEC DLI SCHD PSB(SAMPLE1) END-EXEC.
     PERFORM TEST-DIB THRU OK.                                    5
*
* ***************************************************************
*  RETRIEVE ROOT SEGMENT AND ALL ITS DEPENDENTS
* ***************************************************************
*
     MOVE 'A300' TO SEGKEYA.
     EXEC DLI GU USING PCB(1) SEGMENT(SEGA) INTO(AREAA)
          SEGLENGTH(80) WHERE(KEYA=SEGKEYA)                       6
          FIELDLENGTH(4)
     END-EXEC.
     PERFORM TEST-DIB THRU OK.
 GNPLOOP.
     EXEC DLI GNP USING PCB(1) INTO(AREAG) SEGLENGTH(250)
     END-EXEC.
     IF DIBSTAT EQUAL TO 'GE' THEN GO TO LOOPDONE.
     PERFORM TEST-DIB THRU OK.
     GO TO GNPLOOP.
 LOOPDONE.
*
* ***************************************************************
*  INSERT NEW ROOT SEGMENT
* ***************************************************************
*
     MOVE 'DATA FOR NEW SEGMENT INCLUDING KEY' TO AREAA.
     EXEC DLI ISRT USING PCB(1) SEGMENT(SEGA) FROM(AREAA)
          SEGLENGTH(80) END-EXEC.
     PERFORM TEST-DIB THRU OK.
*
* ***************************************************************
* RETRIEVE 3 SEGMENTS IN PATH AND REPLACE THEM
* ***************************************************************
*
     MOVE 'A200' TO SEGKEYA.
     MOVE 'B240' TO SEGKEYB.
     MOVE 'C241' TO SEGKEYC.
     EXEC DLI GU USING PCB(1)
       SEGMENT(SEGA) WHERE(KEYA=SEGKEYA) FIELDLENGTH(4)           7
          INTO(AREAA)
          SEGLENGTH(80)
       SEGMENT(SEGB) WHERE(KEYB=SEGKEYB) FIELDLENGTH(4)
          INTO(AREAB)
          SEGLENGTH(80)
       SEGMENT(SEGC) WHERE(KEYC=SEGKEYC) FIELDLENGTH(4)
```

```
                INTO(AREAC)
                SEGLENGTH(80)
           END-EXEC.
           PERFORM TEST-DIB THRU OK.
      *    UPDATE FIELDS IN THE 3 SEGMENTS
           EXEC DLI REPL USING PCB(1)
             SEGMENT(SEGA) FROM(AREAA) SEGLENGTH(80)
             SEGMENT(SEGB) FROM(AREAB) SEGLENGTH(80)
             SEGMENT(SEGC) FROM(AREAC) SEGLENGTH(80)
           END-EXEC.
           PERFORM TEST-DIB THRU OK.
      *
      * ***************************************************************
      *  INSERT NEW SEGMENT USING CONCATENATED KEY TO QUALIFY PARENT
      * ***************************************************************
      *
           MOVE 'DATA FOR NEW SEGMENT INCLUDING KEY' TO AREAC.
           MOVE 'A200B240' TO CONKEYB.
           EXEC DLI ISRT USING PCB(1)
             SEGMENT(SEGB) KEYS(CONKEYB) KEYLENGTH(8)
             SEGMENT(SEGC) FROM(AREAC) SEGLENGTH(80)
           END-EXEC.
           PERFORM TEST-DIB THRU OK.
      *
      * ***************************************************************
      *  RETRIEVE SEGMENT DIRECTLY USING CONCATENATED KEY
      *  AND THEN DELETE IT AND ITS DEPENDENTS
      * ***************************************************************
      *
           MOVE 'A200B230' TO CONKEYB.
           EXEC DLI GU USING PCB(1)
             SEGMENT(SEGB)
                KEYS(CONKEYB) KEYLENGTH(8)
                INTO(AREAB) SEGLENGTH(80)
           END-EXEC.
           PERFORM TEST-DIB THRU OK.
           EXEC DLI DLET USING PCB(1)
             SEGMENT(SEGB) SEGLENGTH(80) FROM(AREAB) END-EXEC.
           PERFORM TEST-DIB THRU OK.
      *
      * ***************************************************************
           *  RETRIEVE SEGMENT BY QUALIFYING PARENT WITH CONCATENATED KEY,
           *  OBJECT SEGMENT WITH WHERE OPTION,
           *  AND THEN SET PARENTAGE
           *
           *  USE VARIABLES FOR PCB INDEX, SEGMENT NAME, AND SEGMENT LENGTH
           * ***************************************************************
           *
                MOVE 'A200B230' TO CONKEYB.
                MOVE 'C520' TO SEGKEYC.
                MOVE 'SEGA' TO SEGNAME.
                MOVE 80 TO SEGLEN.
                MOVE 1 TO PCBNUM.
                EXEC DLI GU USING PCB(PCBNUM)
                  SEGMENT((SEGNAME))
                     KEYS(CONKEYB) KEYLENGTH(8) SETPARENT
                  SEGMENT(SEGC) INTO(AREAC) SEGLENGTH(SEGLEN)
                     WHERE(KEYC=SEGKEYC) FIELDLENGTH(4) END-EXEC.
                PERFORM TEST-DIB THRU OK.
           *
           * ***************************************************************
           *  RETRIEVE DATABASE STATISTICS
           * ***************************************************************
           *
                EXEC DLI STAT USING PCB(1) INTO(AREASTAT)
                     VSAM FORMATTED LENGTH(360) END-EXEC.
                PERFORM TEST-DIB THRU OK.
```

```
      *
      * ************************************************************
      *  RETRIEVE ROOT SEGMENT USING BOOLEAN OPERATORS
      * ************************************************************
      *
           MOVE 'A050' TO SEGKEY1.
           MOVE 'A150' TO SEGKEY2.
           MOVE 'A275' TO SEGKEY3.
           MOVE 'A350' TO SEGKEY4.
           EXEC DLI GU USING PCB(1) SEGMENT(SEGA) INTO(AREAA)
                SEGLENGTH(80) FIELDLENGTH(4,4,4,4)
                WHERE(KEYA > SEGKEY1 AND KEYA < SEGKEY2 OR
                      KEYA > SEGKEY3 AND KEYA < SEGKEY4)
           END-EXEC.
           PERFORM TEST-DIB THRU OK.
      *
     * ************************************************************
      *  TERMINATE PSB WHEN DLI PROCESSING IS COMPLETED
      * ************************************************************
      *
             EXEC DLI TERM END-EXEC.                                  8
      *
      * ************************************************************
      * ************************************************************
      *  SEND OUTPUT MESSAGE
      * ************************************************************
      *
           EXEC CICS SEND MAP('SAMPMAP') MAPSET('MAPSET') END-EXEC.
           EXEC CICS WAIT TERMINAL END-EXEC.
      *
      * ************************************************************
      *  COMPLETE TRANSACTION AND RETURN TO CICS
      * ************************************************************
      *
           EXEC CICS RETURN END-EXEC.
      *
      * ************************************************************
      *  CHECK STATUS IN DIB
      * ************************************************************
      *
       TEST-DIB.
           IF DIBSTAT EQUAL TO '  ' THEN GO TO OK.
       OK.                                                            9
       ERRORS.
      *    HANDLE ERROR CONDITIONS
       ABENDS.
      *    HANDLE ABENDS INCLUDING DLI ERROR STATUS CODES
```

サンプル **COBOL** コードに対する注:

1 EXEC DLI コマンドを含む CICS オンライン・プログラムの場合、DLI オプションと CICS オプションを指定しなければなりません。EXEC DLI コマンドを含むバッチ・プログラムまたは BMP プログラムの場合は、 DLI オプションのみを指定しなければなりません。

2 プログラムが使用する区域、すなわち入出力域、キー・フィードバック域、およびセグメント名域のそれぞれを、レベル 77 または 01 の作業用ストレージ項目として定義します。

3 (単一のコマンドで) リトリーブ、追加、または置き換えを行うセグメントごとに入出力域を定義します。

4 バッチ・プログラムまたは BMP プログラムでは、EXEC CICS コマンドをコーディングしないでください。

5 CICS オンライン・プログラムでは、SCHD PSB コマンドを使用して PSB を取得します。バッチ・プログラムまたは BMP プログラムでは、PSB をスケジュールしてはなりません。

6 この GU コマンドは、キー値が A300 の SEGA が最初に現れる位置をリトリーブします。 IBM COBOL for z/OS & VM (および VS COBOL II) では、KEYLENGTH と SEGLENGTH の指定はオプションです。COBOL V4 および OS/VS COBOL では、KEYLENGTH と SEGLENGTH の指定は必須です。

7 この GU コマンドは、パス・コマンドの例です。リトリーブするセグメントごとに、別々の入出力域を使用しなければなりません。

8 CICS オンライン・プログラムでは、先にスケジュールされた PSB が、TERM コマンドにより終了します。バッチ・プログラムまたは BMP プログラムでは、PSB を終了させてはなりません。

9 各コマンドを出した後、DIB の状況コードを検査する必要があります。

PL/I でのプログラム・コーディング

次の PL/I によるサンプル・プログラムは、コマンド・レベル・プログラムのさまざまな部分が整合して機能する様子、および CICS オンライン・プログラムにおける EXEC DLI コマンドのコーディング方法を示しています。

一部のコマンドを除き、このプログラムは、バッチ、BMP、CICS プログラムに適用されます。違いはいずれも、サンプル PL/I コードの注で強調して示してあります。サンプル・コードの右側の番号は、このような注を指しています。

```
*PROCESS INCLUDE,GN,XOPTS(CICS,DLI);                                    1
 SAMPLE: PROCEDURE OPTIONS(MAIN);
 DCL      SEGKEYA                 CHAR (4);
 DCL      SEGKEYB                 CHAR (4);                              2
 DCL      SEGKEYC                 CHAR (4);
 DCL      SEGKEY1                 CHAR (4);
 DCL      SEGKEY2                 CHAR (4);
 DCL      SEGKEY3                 CHAR (4);
 DCL      SEGKEY4                 CHAR (4);
 DCL      CONKEYB                 CHAR (8);
 DCL      SEGNAME                 CHAR (8);
 DCL      PCBNUM                  FIXED BIN (15);
 DCL      AREAA                   CHAR (80);
     /*   DEFINE SEGMENT I/O AREA                                */
 DCL      AREAB                   CHAR (80);
 DCL      AREAC                   CHAR (80);                             3
 DCL      AREAG                   CHAR (250);
 DCL      AREASTAT                CHAR (360);
     %INCLUDE MAPSET
     /*                                                          */
     /*                                                          */
     /* ********************************************************** */
     /*  INITIALIZATION                                          */
     /*  HANDLE ERROR CONDITIONS IN ERROR ROUTINE                */
     /*  HANDLE ABENDS (DLI ERROR STATUS CODES) IN ABEND PROGRAM */
     /*  RECEIVE INPUT MESSAGE                                   */
     /* ********************************************************** */
     /*                                                          */
     EXEC CICS HANDLE CONDITION ERROR(ERRORS);                          4
     /*                                                          */
     EXEC CICS HANDLE ABEND PROGRAM('ABENDS');                          4
     /*                                                          */
     EXEC CICS RECEIVE MAP ('SAMPMAP') MAPSET('MAPSET');                4
     /* ANALYZE INPUT MESSAGE AND PERFORM NON-DLI PROCESSING     */
     /*                                                          */
```

```
     /* ************************************************************  */
     /*  SCHEDULE PSB NAMED 'SAMPLE1'                                 */
     /* ************************************************************  */
     /*                                                               */
     EXEC DLI SCHD PSB(SAMPLE1);
     CALL TEST_DIB;                                                          5

     /* ************************************************************ */
     /*  RETRIEVE ROOT SEGMENT AND ALL ITS DEPENDENTS                */
     /* ************************************************************ */
     /*                                                               */
     SEGKEYA = 'A300';
     EXEC DLI GU USING PCB(1) SEGMENT(SEGA) INTO(AREAA)
     WHERE(KEYA=SEGKEYA);                                                    6
     CALL TEST_DIB;
GNPLOOP:
     EXEC DLI GNP USING PCB(1) INTO(AREAG);                                  7
     IF DIBSTAT = 'GE' THEN GO TO LOOPDONE;
     CALL TEST_DIB;
     GO TO GNPLOOP;
 LOOPDONE:
     /*                                                               */
     /* ************************************************************  */
     /*  INSERT NEW ROOT SEGMENT                                      */
     /* ************************************************************  */
     /*                                                               */
     AREAA = 'DATA FOR NEW SEGMENT INCLUDING KEY';
     EXEC DLI ISRT USING PCB(1) SEGMENT(SEGA) FROM(AREAA);
     CALL TEST_DIB;
     /*                                                               */
     /* ************************************************************ */
     /*  RETRIEVE 3 SEGMENTS IN PATH AND REPLACE THEM                */
     /* ************************************************************ */
     /*                                                               */
     SEGKEYA = 'A200';
     SEGKEYB = 'B240';
     SEGKEYC = 'C241';
     EXEC DLI GU USING PCB(1)
       SEGMENT(SEGA) WHERE(KEYA=SEGKEYA)                                     8
          INTO(AREAA)
       SEGMENT(SEGB) WHERE(KEYB=SEGKEYB)
          INTO(AREAB)
       SEGMENT(SEGC) WHERE(KEYC=SEGKEYC)
          INTO(AREAC);
     CALL TEST_DIB;
     /* UPDATE FIELDS IN THE 3 SEGMENTS                               */
     EXEC DLI REPL USING PCB(1)
       SEGMENT(SEGA) FROM(AREAA)
       SEGMENT(SEGB) FROM(AREAB)
       SEGMENT(SEGC) FROM(AREAC);
     CALL TEST_DIB;
     /*                                                               */
     /* ************************************************************ */
     /*  INSERT NEW SEGMENT USING CONCATENATED KEY TO QUALIFY PARENT  */
     /* ************************************************************ */
     /*                                                               */
     AREAC = 'DATA FOR NEW SEGMENT INCLUDING KEY';
     CONKEYB = 'A200B240';
     EXEC DLI ISRT USING PCB(1)
       SEGMENT(SEGB) KEYS(CONKEYB)
       SEGMENT(SEGC) FROM(AREAC);
     CALL TEST_DIB;
     /*                                                               */
     /* ************************************************************  */
     /*  RETRIEVE SEGMENT DIRECTLY USING CONCATENATED KEY             */
     /*  AND THEN DELETE IT AND ITS DEPENDENTS                        */
     /* ************************************************************  */
```

```
    /*                                                               */
    CONKEYB = 'A200B230';
    EXEC DLI GU USING PCB(1)
      SEGMENT(SEGB)
         KEYS(CONKEYB)
         INTO(AREAB);
    CALL TEST_DIB;
    EXEC DLI DLET USING PCB(1)
      SEGMENT(SEGB) FROM(AREAB);
    CALL TEST_DIB;
    /*                                                               */
/* ************************************************************* */
    /*  RETRIEVE SEGMENT BY QUALIFYING PARENT WITH CONCATENATED KEY, */
    /*  OBJECT SEGMENT WITH WHERE OPTION                             */
    /*  AND THEN SET PARENTAGE                                       */
    /*                                                               */
    /*  USE VARIABLES FOR PCB INDEX, SEGMENT NAME                    */
    /* ************************************************************* */
    /*                                                               */
    CONKEYB = 'A200B230';
    SEGNAME = 'SEGA';
    SEGKEYC = 'C520';
    PCBNUM = 1;
    EXEC DLI GU USING PCB(PCBNUM)
      SEGMENT((SEGNAME))
         KEYS(CONKEYB) SETPARENT
      SEGMENT(SEGC) INTO(AREAC)
         WHERE(KEYC=SEGKEYC);
    CALL TEST_DIB;
    /*                                                               */
    /* ************************************************************* */
    /*  RETRIEVE DATABASE STATISTICS                                 */
    /* ************************************************************* */
    /*                                                               */
    EXEC DLI STAT USING PCB(1) INTO(AREASTAT) VSAM FORMATTED;
    CALL TEST_DIB;
    /*                                                               */
    /* ************************************************************  */
    /*  RETRIEVE ROOT SEGMENT USING BOOLEAN OPERATORS                */
    /* ************************************************************  */
    /*                                                               */
    SEGKEY1 = 'A050';
    SEGKEY2 = 'A150';
    SEGKEY3 = 'A275';
    SEGKEY4 = 'A350';
    EXEC DLI GU USING PCB(1) SEGMENT(SEGA) INTO(AREAA)
         WHERE(KEYA &Ar; SEGKEY1 AND KEYA &Al; SEGKEY2 OR
               KEYA &Ar; SEGKEY3 AND KEYA &Al; SEGKEY4);
    CALL TEST_DIB;
    /*                                                               */
    /* ************************************************************* */
    /*  TERMINATE PSB WHEN DLI PROCESSING IS COMPLETED               */
    /* ************************************************************* */
    /*                                                               */

    EXEC DLI TERM;                                                   9

    /*                                                               */
    /* ************************************************************* */
    /*  SEND OUTPUT MESSAGE                                          */
    /* ************************************************************* */
    /*                                                               */
    EXEC CICS SEND MAP('SAMPMAP') MAPSET('MAPSET');                  4
    EXEC CICS WAIT TERMINAL;
    /*                                                               */
    /* ************************************************************* */
```

```
    /*  COMPLETE TRANSACTION AND RETURN TO CICS                     */
    /* ************************************************************ */
    /*                                                              */
    EXEC CICS RETURN;                                                  4
    /*                                                              */
    /* ************************************************************ */
    /*  CHECK STATUS IN DIB                                         */
    /* ************************************************************ */
    /*                                                              */
 TEST_DIB: PROCEDURE;
    IF DIBSTAT = '  ' RETURN;                                         10

    /* HANDLE DLI STATUS CODES REPRESENTING EXCEPTIONAL CONDITIONS   */
    /*                                                              */
 OK:
 END TEST_DB;
 ERRORS:
    /* HANDLE ERROR CONDITIONS                                     */
    /*                                                              */
 END SAMPLE;
```

サンプル **PL/I** コードに対する注**:**

1 EXEC DLI コマンドを含む CICS オンライン・プログラムの場合、DLI オプションと CICS オプションを指定しなければなりません。EXEC DLI コマンドを含むバッチ・プログラムまたは BMP プログラムの場合は、 DLI オプションのみを指定しなければなりません。

2 自動ストレージに、各領域、すなわち入出力域、キー・フィードバック域、およびセグメント名域を定義します。

3 (単一のコマンドで) リトリーブ、追加、または置き換えを行うセグメントごとに入出力域を定義します。

4 バッチ・プログラムまたは BMP プログラムでは、EXEC CICS コマンドをコーディングしないでください。

5 CICS オンライン・プログラムでは、SCHD PSB コマンドを使用して PSB を取得します。バッチ・プログラムまたは BMP プログラムでは、PSB をスケジュールしてはなりません。

6 この GU コマンドは、キー値が A300 の SEGA が最初に現れる位置をリトリーブします。 KEYLENGTH および SEGLENGTH オプションを指定する必要がないことに注意してください。

7 この GNP コマンドは、セグメント SEGA の従属セグメントをすべてリトリーブします。 GE 状況コードは、従属セグメントがそれ以上ないことを意味します。

8 この GU コマンドは、パス・コマンドの例です。リトリーブするセグメントごとに、別々の入出力域を使用しなければなりません。

9 CICS オンライン・プログラムでは、先にスケジュールされた PSB が、TERM コマンドにより終了します。バッチ・プログラムまたは BMP プログラムでは、PSB を終了させてはなりません。

10 各コマンドを出した後、DIB の状況コードを検査する必要があります。

C でのプログラム・コーディング

次のサンプルの C プログラムは、コマンド・レベル・プログラムのさまざまな部分が整合して機能する様子、および CICS オンライン・プログラムにおける EXEC DLI コマンドのコーディング方法を示しています。

一部のコマンドを除き、このプログラムは、バッチ、BMP、CICS プログラムに適用されます。違いはいずれも、サンプル C コードの注で強調して示してあります。サンプル・コードの右側の番号は、このような注を指しています。

```
#include < string.h>                                                    1
#include < stdio.h >                                                    2

   char  DIVIDER[120] = "----------------------------------------¥
 ------------------------------------------------------------------";
   char  BLANK[120]   = "                                            ¥
                                                                     ¥0";
   char  BLAN2[110]   = "                                            ¥
                                                  ¥0";
   char  SCHED[120] = "Schedule PSB(PC3COCHD)         "                    3
   char  GN1[120] = "GN using PCB(2) Segment(SE2ORDER) check dibstat ¥
is blank";
   char   GNP1[120] = "GNP using PCB(2) check dibstat = GK or blank ¥
(or GE for last GNP)";
   char   GU1[120]  = "GU using PCB(2) Segment(SE2ORDER) where(¥
FE2OGREF=000000'') check dibstat blank";
   char   GU2[120]  = "GU using PCB(2) Segment(SE2ORDER) where(¥
FE2OGREF=000999'') check dibstat blank";
   char  REP1[120]  = "REPLACE using PCB(2) Segment(SE2ORDER) check ¥
dibstat is blank";
   char  DEL1[120]   = "DELETE using PCB(2) Segment(SE2ORDER) check ¥
dibstat is blank";
   char  INS1[120]   = "INSERT using PCB(2) Segment(SE2ORDER) where¥
(FE2OGREF=''000999'') check dibstat is blank";
   char  TERM[120]   = "TERM - check dibstat is blank";
   char  STAT[120]   = "STAT USING PCB(2) VSAM FORMATTED";
   char  DATAB[6]    = "000999";
   char  DATAC[114]  = " REGRUN TEST INSERT NO1.";
   char  START[120]  = "PROGXIV STARTING";
   char  OKMSG[120]  = "PROGXIV COMPLETE";
   int   TLINE = 120;
   int   L11 = 11;
   int   L360 = 11;
   struct {
         char NEWSEGB[6];
         char NEWSEGC[54];
   } NEWSEG;
   char  OUTLINE[120];                                                   4
   struct {
         char OUTLINA[9];
         char OUTLINB[111];
   } OUTLIN2;
   struct {
         char  OUTLINX[9];
         char  OUTLINY[6];
         char  OUTLINZ[105];
   } OUTLIN3;
   char  GUIOA[60];
   char  GNIOA[60];
   struct {
         char ISRT1[6];
         char ISRT2[54];
   } ISRTIOA;
   struct {
         char REPLIO1[6];
```

```
        char REPLIO2[54];
   } REPLIOA;
   struct {
        char DLET1[6];
        char DLET2[54];
   } DLETIOA;
   struct {
        char STATA1[120];
        char STATA2[120];
        char STATA3[120];
   } STATAREA;
   struct {
        char DHPART[2];
        char RETCODE[2]
   } DHABCODE;

main()
{
        EXEC CICS ADDRESS EIB(dfheiptr);                          5
        strcpy(OUTLINE,DIVIDER);
        SENDLINE();
        strcpy(OUTLINE,START);
        SENDLINE();
    /*                                                          */
    /* SCHEDULE PSB                                             */
    /*                                                          */
        strcpy(OUTLINE,SCHED);
        SENDLINE();
        EXEC DLI SCHEDULE PSB(PC3COCHD);                          6
        SENDSTAT();
        TESTDIB();
    /*                                                         */
    /* ISSUE GU REQUEST                                        */
    /*                                                         */
        strcpy(OUTLINE,GU1);
        SENDLINE();
        EXEC DLI GET UNIQUE USING PCB(2)                          7
             SEGMENT(SE2ORDER)
             WHERE(FE2OGREF>="000000")
             INTO(&GUIOA) SEGLENGTH(60);
        strcpy(OUTLIN2.OUTLINA,"SE2ORDER=");
        strcpy(OUTLIN2.OUTLINB,GUIOA);
        SENDLIN2();
        SENDSTAT();
        TESTDIB();
    /*                                                    */
    /* ISSUE GNP REQUEST                                  */
    /*                                                    */
 do {
        strcpy(OUTLINE,GNP1);
        SENDLINE();
        EXEC DLI GET NEXT IN PARENT USING PCB(2)                  8
             INTO(&GNIOA) SEGLENGTH(60);
        strcpy(OUTLIN2.OUTLINA,"SEGMENT=");
        strcpy(OUTLIN2.OUTLINB,GNIOA);
        SENDLIN2();
        SENDSTAT();
        if (strncmp(dibptr->dibstat,"GE",2) != 0)              9
           TESTDIB();
  } while (strncmp(dibptr->dibstat,"GE",2) != 0);
    /*                                                         */
    /* ISSUE GN REQUEST                                        */
    /*                                                         */
        strcpy(OUTLINE,GN1);
        SENDLINE();
        EXEC DLI GET NEXT USING PCB(2)
             SEGMENT(SE2ORDER)                                   10
```

```
           INTO(&GNIOA) SEGLENGTH(60);
     strcpy(OUTLIN2.OUTLINA,"SE2ORDER=");
     strcpy(OUTLIN2.OUTLINB,GNIOA);
     SENDLIN2();
     SENDSTAT();
     TESTDIB();
/*                                                  */
/* INSERT SEGMENT                                   */
/*                                                  */
     strcpy(OUTLINE,INS1);
     SENDLINE();
     strcpy(NEWSEG.NEWSEGB,DATAB);                                  11
     strcpy(NEWSEG.NEWSEGC,DATAC);
     strcpy(ISRTIOA.ISRT1,NEWSEG.NEWSEGB);
     strcpy(ISRTIOA.ISRT2,NEWSEG.NEWSEGC);
     strcpy(OUTLIN3.OUTLINX,"ISRT SEG=");
     strcpy(OUTLIN3.OUTLINY,ISRTIOA.ISRT1);
     strcpy(OUTLIN3.OUTLINZ,ISRTIOA.ISRT2);
     SENDLIN3();
     EXEC DLI ISRT USING PCB(2)
           SEGMENT(SE2ORDER)
           FROM(&ISRTIOA) SEGLENGTH(60);
     SENDSTAT();
     if (strncmp(dibptr->dibstat,"II",2) == 0)
           strncpy(dibptr->dibstat,"  ",2);
     TESTDIB();
/*                                                  */
/* ISSUE GN REQUEST                                 */
/*                                                  */
     strcpy(OUTLINE,GN1);
     SENDLINE();
     EXEC DLI GET NEXT USING PCB(2)                                 12
           SEGMENT(SE2ORDER)
           INTO(&GNIOA) SEGLENGTH(60);
     strcpy(OUTLIN2.OUTLINA,"SE2ORDER=");
     strcpy(OUTLIN2.OUTLINB,GNIOA);
     SENDLIN2();
     SENDSTAT();
     TESTDIB();
/*                                              */
/* GET INSERTED SEGMENT TO BE REPLACED          */
/*                                              */
     strcpy(OUTLINE,GU2);
     SENDLINE();
     EXEC DLI GET UNIQUE USING PCB(2)                               13
           SEGMENT(SE2ORDER)
           WHERE(FE2OGREF="000999")
           INTO(&ISRTIOA) SEGLENGTH(60);
     strcpy(OUTLIN3.OUTLINX,"ISRT SEG=");
     strcpy(OUTLIN3.OUTLINY,ISRTIOA.ISRT1);
     strcpy(OUTLIN3.OUTLINZ,ISRTIOA.ISRT2);
     SENDLIN3();
     SENDSTAT();
     TESTDIB();
/*                                                */
/* REPLACE SEGMENT                                */
/*                                                */
     strcpy(OUTLINE,REP1);
     SENDLINE();
     strcpy(REPLIOA.REPLIO1,DATAB);                                 14
     strcpy(REPLIOA.REPLIO2,"REGRUN REPLACED SEGMENT NO1.");
     strcpy(OUTLIN3.OUTLINX,"REPL SEG=");
     strcpy(OUTLIN3.OUTLINY,REPLIOA.REPLIO1);
     strcpy(OUTLIN3.OUTLINZ,REPLIOA.REPLIO2);
     SENDLIN3();
     EXEC DLI REPLACE USING PCB(2)
           SEGMENT(SE2ORDER)
```

```
          FROM(&REPLIOA) SEGLENGTH(60);
     SENDSTAT();
     TESTDIB();
/*                                                  */
/* ISSUE GN REQUEST                                 */
/*                                                  */
     strcpy(OUTLINE,GN1);
     SENDLINE();
     EXEC DLI GET NEXT USING PCB(2)                                  15
          SEGMENT(SE2ORDER)
          INTO(&GNIOA) SEGLENGTH(60);
     strcpy(OUTLIN2.OUTLINA,"SE2ORDER=");
     strcpy(OUTLIN2.OUTLINB,GNIOA);
     SENDLIN2();
     SENDSTAT();
     TESTDIB();
/*                                            */
/* GET REPLACED SEGMENT                       */
/*                                            */
     strcpy(OUTLINE,GU2);
     SENDLINE();
     EXEC DLI GET UNIQUE USING PCB(2)                                16
          SEGMENT(SE2ORDER)
          WHERE(FE2OGREF="000999")
          INTO(&REPLIOA) SEGLENGTH(60);
     strcpy(OUTLIN3.OUTLINX,"REPL SEG=");
     strcpy(OUTLIN3.OUTLINY,REPLIOA.REPLIO1);
     strcpy(OUTLIN3.OUTLINZ,REPLIOA.REPLIO2);
     SENDLIN3();
     SENDSTAT();
     TESTDIB();
/*                                                        */
/* ISSUE DELETE REQUEST                                   */
/*                                                        */
     strcpy(OUTLINE,DEL1);
     SENDLINE();
     strcpy(DLETIOA.DLET1,REPLIOA.REPLIO1);                          17
     strcpy(DLETIOA.DLET2,REPLIOA.REPLIO2);
     strcpy(OUTLIN3.OUTLINX,"DLET SEG=");
     strcpy(OUTLIN3.OUTLINY,DLETIOA.DLET1);
     strcpy(OUTLIN3.OUTLINZ,DLETIOA.DLET2);
     SENDLIN3();
     EXEC DLI DELETE USING PCB(2)
          SEGMENT(SE2ORDER)
          FROM(&DLETIOA) SEGLENGTH(60);
     SENDSTAT();
     TESTDIB();
/*                                                        */
/* ISSUE STAT REQUEST                                     */
/*                                                        */
     strcpy(OUTLINE,STAT);
     SENDLINE();
     EXEC DLI STAT USING PCB(2)                                      18
          VSAM FORMATTED
          INTO(&STATAREA);
     SENDSTT2();
     TESTDIB();
/*                                                        */
/* ISSUE TERM REQUEST                                     */
/*                                                        */
     strcpy(OUTLINE,TERM);
     SENDLINE();
     EXEC DLI TERM;                                                  19
     SENDSTAT();
     TESTDIB();
     strcpy(OUTLINE,DIVIDER);
     SENDLINE();
```

```
        SENDOK();
    /*                                                          */
    /* RETURN TO CICS                                           */
    /*                                                          */
        EXEC CICS RETURN;
}
    /*                                                          */
    /*                                                          */
    /*                                                          */
 SENDLINE()
 {
        EXEC CICS SEND FROM(OUTLINE) LENGTH(120);                                  20
        EXEC CICS WRITEQ TD QUEUE("PRIM") FROM(OUTLINE) LENGTH(TLINE);
        strcpy(OUTLINE,BLANK);
        return;
 }

 SENDLIN2()
{
        EXEC CICS SEND FROM(OUTLIN2) LENGTH(120);
        EXEC CICS WRITEQ TD QUEUE("PRIM") FROM(OUTLIN2) LENGTH(TLINE);
        strcpy(OUTLIN2.OUTLINA,BLANK,9);
        strcpy(OUTLIN2.OUTLINB,BLANK,111);
        return;
 }

 SENDLIN3()
{
        EXEC CICS SEND FROM(OUTLIN3) LENGTH(120);
        EXEC CICS WRITEQ TD QUEUE("PRIM") FROM(OUTLIN3) LENGTH(TLINE);
        strcpy(OUTLIN3.OUTLINX,BLANK,9);
        strcpy(OUTLIN3.OUTLINY,BLANK,6);
        strcpy(OUTLIN3.OUTLINZ,BLANK,105);
        return;
 }

SENDSTAT()
{
        strncpy(OUTLIN2.OUTLINA,BLANK,9);
        strncpy(OUTLIN2.OUTLINB,BLAN2,110);
        strcpy(OUTLIN2.OUTLINA," DIBSTAT=");
        strcpy(OUTLIN2.OUTLINB,dibptr->dibstat);
        EXEC CICS SEND FROM(OUTLIN2) LENGTH(11);
        EXEC CICS WRITEQ TD QUEUE("PRIM") FROM(OUTLIN2) LENGTH(L11);
        strcpy(OUTLINE,DIVIDER);
        SENDLINE();
        return;
 }

SENDSTT2()
{
        strncpy(OUTLIN2.OUTLINA,BLANK,9);
        strncpy(OUTLIN2.OUTLINB,BLAN2,110);
        strcpy(OUTLIN2.OUTLINA," DIBSTAT=");
        strcpy(OUTLIN2.OUTLINB,dibptr->dibstat);
        EXEC CICS SEND FROM(STATAREA) LENGTH(360);
        EXEC CICS WRITEQ TD QUEUE("PRIM") FROM(STATAREA)
                  LENGTH(L360);
        return;
 }

SENDOK()
{
        EXEC CICS SEND FROM(OKMSG) LENGTH(120);
        EXEC CICS WRITEQ TD QUEUE("PRIM") FROM(OKMSG) LENGTH(TLINE);
        return;
 }
```

```
TESTDIB()                                                              21
{
    if (strncmp(dibptr->dibstat,"  ",2) == 0)
       return;
    else if (strncmp(dibptr->dibstat,"GK",2) == 0)
       return;
    else if (strncmp(dibptr->dibstat,"GB",2) == 0)
       return;
    else if (strncmp(dibptr->dibstat,"GE",2) == 0)
       return;
    else
       {
            EXEC CICS ABEND ABCODE("PETE");                            22
            EXEC CICS RETURN;
       }
         return;
 }
```

サンプル C コードに対する注:

1 ストリング処理機能にアクセスするには、標準ヘッダー・ファイル string.h を組み込まなければなりません。

2 標準入出力ライブラリーにアクセスするには、標準ヘッダー・ファイル stdio.h を組み込まなければなりません。

3 DL/I メッセージを定義します。

4 入出力域を定義します。

5 プログラムを始動します。

6 PSB PC3COCHD を定義します。

7 最初のコマンドを出します。 SE2ORDER セグメントの最初のオカレンスをリトリーブして、配列 OUTLIN2 に書き込みます。

8 次のセグメントを読み取って配列 OUTLIN2 に書き込むために GNP コマンドを出します。

9 GE 状況コードは読み取るセグメントがないことを示します。

10 次のセグメント SE2ORDER を読み取って、配列 OUTLIN2 に書き込みます。

11 セグメントを配列 OUTLIN3 に挿入します。

12 次のセグメントをリトリーブして配列 OUTLIN2 に書き込むために、GN を出します。

13 置き換える次のセグメントを読み取って OUTLIN3 に書き込みます。

14 セグメントを置き換え、配列 OUTLIN3 に書き込みます。

15 次のセグメントを読み取って配列 OUTLIN2 に書き込みます。

16 置換されたセグメントを読み取り、配列 OUTLIN3 に書き込みます。

17 セグメントの内容を配列 OUTLIN3 に書き込んだ後、DELETE コマンドを出します。

18 STAT REQUEST コマンドを発行します。

19 TERM コマンドを発行します。

20 出力処理を行います。

21 戻りコードを検査します。

22 バッチ・プログラムまたは BMP プログラムでは、EXEC CICS コマンドをコーディングしないでください。

EXEC DLI プログラムの実行準備

EXEC DLI プログラムを実行する前に、それを変換、コンパイル、およびバインドする必要があります。

プログラムを変換、コンパイル、およびバインドするために、CICS 提供プロシージャーを使用することができます。使用するプロシージャーは、プログラムのタイプ (バッチ、BMP、または CICS オンライン) およびどの言語で作成されたか (COBOL、PL/I、またはアセンブラー言語) によって異なります。

次の手順に従ってプログラムの実行準備を行います。

1. EXEC DLI コマンドと EXEC CICS コマンドを変換するために、CICS コマンド言語変換プログラムを実行します。 COBOL、PL/I、アセンブラー言語の各プログラムには、それぞれ個別の変換プログラムがあります。
2. プログラムをコンパイルします。
3. バインド:
   - 適切な CICS インターフェース・モジュールをもつオンライン・プログラム
   - IMS インターフェース・モジュールをもつバッチまたは BMP プログラム

EXEC DLI に必要な変換プログラム、コンパイラー、およびバインド・プログラムのオプション

EXEC DLI プログラムを実行するには、変換プログラム、コンパイル、およびバインド・プログラムのオプションを設定する必要があります。

EXEC DLI に必要な変換プログラム・オプション

CICS 提供プロシージャーを使用してプログラムを準備する場合でも、特定の変換プログラム・オプションを指定しなければなりません。

EXEC DLI コマンドを含む CICS オンライン・プログラムの場合、DLI オプションと CICS オプションを指定しなければなりません。EXEC DLI コマンドを含むバッチ・プログラムまたは BMP プログラムの場合は、 DLI オプションを指定しなければなりません。

変換プログラムを呼び出す EXEC ジョブ制御ステートメントにおいても、これらのオプションを指定できます。両方のメソッドを使用した場合、 CBL および *PROCESS ステートメントは、EXEC ステートメントで指定されたオプションを無効にします。変換プログラム・オプションの詳細については、「*CICS Transaction Server for z/OS CICS* アプリケーション・プログラミング・ガイド」を参照してください。

COBOL プログラムで使用する変換プログラム・オプションが COBOL コンパイラー・オプションと矛盾しないようにしてください。

EXEC DLI に必要なコンパイラー・オプション

バッチ COBOL プログラムをコンパイルするには、選択した COBOL コンパイラーに応じて、異なるコンパイラー・オプションを使用することが必要になる場合があります。 CICS プログラムにどのコンパイラー・オプションを使用するべきかについて詳しくは、「*CICS Transaction Server for z/OS CICS* アプリケーション・プログラミング・ガイド」を参照してください。

EXEC DLI に必要なバインダー・オプション

使用しているコンパイラーがサポートしている場合には、EXEC コマンドを使用して作成されたプログラムを、AMODE(31) RMODE(ANY) としてリンクすることができます。

第 30 章 アプリケーション・プログラム・エレメントの定義

アプリケーション・プログラム・エレメントを定義するには、アプリケーション・インターフェース・ブロック (AIB) および DL/I インターフェース・ブロック (DIB) を指定した EXEC DLI コマンドを使用し、フィードバック域および入出力域を定義します。

アプリケーション・インターフェース・ブロック (AIB) の指定

EXEC DLI コマンドは AIB インターフェースを使用できます。

例えば、AIB インターフェースを使用すると、GU コマンドの形式は PCB 形式を使用した EXEC DLI GU USING PCB(n) ではなく、EXEC DLI GU AIB(aib) になります。

IBM CICS Transaction Server for z/OS では、EXEC DLI コマンドが AIB 形式 (および PCB 形式) でサポートされています。EXEC DLI インターフェースの使用により、AIB 専用コマンド ICMD、RCMD、および GMSG がサポートされています。

CICS EDF (実行診断機能) デバッグ・トランザクションは、PCB タイプの要求を処理する場合と同様に、AIB EXEC DLI 要求もサポートしています。

AIB マスク (AIB mask)

AIB マスクはアプリケーションによって指定され、EXEC 呼び出しで PCB 番号の代わりに参照されなければなりません (例えば、EXEC DLI GU AIB(aib))。

AIBRETRN = X'00000000' または X'00000900' の場合、DIBSTAT フィールドには有効な STATUS コードが設定されています。アプリケーションは他の値について AIBRETRN をテストし、応答する必要があります。

AIB サポートに関する CICS の制約事項

機能シップのための制約事項は、次のとおりです。

- AIBLEN フィールドは 128 バイトから 256 バイトでなければなりません。 推奨値は 128 バイトです。
- PSB 内の PCB に LIST=NO が指定されていてはなりません。

関連資料:

607 ページの『第 31 章 アプリケーション・プログラム用の EXEC DLI コマンド』

DL/I インターフェース・ブロック (DIB) の指定

プログラムが DL/I コマンドを実行するたびに、DL/I は状況コードとその他の情報を DL/I インターフェース・ブロック (DIB) を介してプログラムに戻します。DIB は IMS PCB のサブセットです。プログラムは、状況コードを検査して、コマンドが正しく実行されたかどうか確認する必要があります。

各プログラムの作業ストレージには独自の DIB があります。 DIB の内容は、そのプログラムの中で実行された最後の DL/I コマンドの状況を表します。 プログラムの DIB に入っている情報が、トランザクションで使用している別のプログラムに必要な場合、この情報をそのプログラムに渡さなければなりません。

DIB 内のフィールドにアクセスするときは、変換プログラムによってプログラム内に自動的に生成されるラベルを使用します。

制約事項: それらのラベルは予約済みであり、再定義することはできません。

COBOL、PL/I、アセンブラー言語、C などのプログラムでは、一部の変数名が必須となります。

COBOL プログラム

```
DIBVER    PICTURE X(2)
DIBSTAT    PICTURE X(2)
DIBSEGM    PICTURE X(8)
DIBSEGLV    PICTURE X(2)
DIBKFBL    PICTURE S9(4) COMPUTATIONAL
DIBDBDNM    PICTURE X(8)
DIBDBORG    PICTURE X(8)
```

```
DIBVER    CHAR(2)
DIBSTAT    CHAR(2)
DIBSEGM    CHAR(8)
DIBSEGLV    CHAR(2)
DIBKFBL    FIXED BINARY (15,0)
DIBDBDNM    CHAR(8)
DIBDBORG    CHAR(8)
```

アセンブラー言語プログラム

```
DIBVER    CL2
DIBSTAT    CL2
DIBSEGM    CL8
DIBSEGLV    CL2
DIBKFBL    H
DIBDBDNM    CL8
DIBDBORG    CL8
```

C プログラム

```
unsigned char     dibver    {2}  ;
unsigned char     dibstat   {2}  ;
unsigned char     dibsegm   {8}  ;
unsigned char     dibfic01  ;
unsigned char     dibfic02  ;
unsigned char     dibseglv  {2}  ;
signed short int  dibkfbl   ;
unsigned char     dibdbdnm  {8}  ;
unsigned char     dibdborg  {8}  ;
unsigned char     dibfic03  {6}  ;
```

以下の注では、各変数名の内容について説明します。括弧で囲まれた名前は、変数名に入っている情報にアクセスする時に使用されるラベルです。

1. 変換プログラムのバージョン **(DIBVER)**

   これは、プログラムが使用している DIB 形式のバージョンです。 (DIBVER は文書化と問題判別に使用されます。)

2. 状況コード **(DIBSTAT)**

DL/I は、各 DL/I コマンドを実行後、このフィールドに 2 文字の状況コードを入れます。 このコードは、コマンドの結果を記述します。

DL/I コマンドの処理後、DL/I は、このコマンドに続く次の順次命令で制御をプログラムに戻します。各コマンドの実行後、プログラムはまず最初に状況コード・フィールドをテストし、適切な処置を行う必要があります。 このコマンドが完全に正常に実行されたときは、このフィールドにブランクが入ります。

このフィールドに戻される状況コードは次のとおりです (プログラムに戻される状況コードは、以下のコードだけです)。

bb (ブランク) コマンドは完全に成功しました。

BA GU、GN、GNP、DLET、REPL、ISRT コマンドの実行後に戻されます。 データがありませんでした。

BC DLET、REPL、ISRT コマンドの実行後に戻されます。デッドロックが検出されました。

FH GU、GN、GNP、DLET、REPL、ISRT、POS、CHKP、SYMCHKP コマンドの実行後に戻されます。 DEDB にアクセスできませんでした。

FW GU、GN、GNP、DLET、REPL、ISRT、POS コマンドの実行後に戻されます。 通常の許容バッファー・スペースよりも多くのスペースが必要です。

GA 修飾されていない GN および GNP コマンドの実行後に戻されます。 DL/I はセグメントを戻しましたが、このセグメントは、最後に戻されたセグメントよりも階層レベルが高くなっています。

GB GN コマンドの実行後に戻されます。 DL/I は GN コマンドを正しく実行しようとしてデータベースの終わりに達したため、プログラムの入出力域にセグメントを戻しませんでした。

GD ISRT コマンドの実行後に戻されます。 プログラムが出した ISRT コマンドに、挿入されるセグメントより高いすべてのレベルに対して SEGMENT オプションが指定されていませんでした。

GE GU、GN、GNP、ISRT、STAT コマンドの実行後に戻されます。 DL/I は、ユーザーが要求したセグメント、あるいはユーザーが挿入しようとしているセグメントの親を 1 つも見つけることができませんでした。

GG 読み取りコマンドに対するものです。 処理オプション GOT または GON を指定しているプログラムがリトリーブしようとしているセグメントに無効ポインターが含まれている場合、DL/I は、このプログラムに GG 状況コードを戻します。

GK 修飾されていない GN および GNP コマンドの実行後に戻されます。 DL/I は、修飾されていない GN または GNP 要求を正しく実行するセグメントを戻しますが、このセグメントのタイプが最後に戻されたセグメントのタイプと異なります (ただし、レベルは同じです)。

II ISRT コマンドの実行後に戻されます。 ユーザーが挿入しようとしているセグメントが既にデータベースにあります。 セグメントを挿入しようとする前にこのセグメントのパスを設定しておかなかった場合にも、このコードが戻される場合があります。 ユーザーが挿入しようとして

いるセグメントが、別の階層またはデータベース・レコードの中の同一キーをもつセグメントと一致している場合があります。

LB LOAD コマンドを出した後にだけロード・プログラムが戻されます。 ロードしようとしているセグメントが既にデータベースにあります。 DL/I は、キー・フィールドをもつセグメントに対してのみこの状況コードを戻します。

NI ISRT と REPL コマンドの実行後に戻されます。 挿入または置き換えようとしているセグメントは、重複項目を禁止している副次索引に重複項目の挿入を要求しています。この状況コードは、ログ・レコードを直接アクセス・ストレージに書き込むバッチ・プログラムに対して戻されます。ディスクにログを記録しない CICS プログラムがこの状況を検出すると、プログラム (トランザクション) が異常終了します。

TG TERM コマンドの実行後に戻されます。 PSB がスケジュールされていないのに、プログラムが PSB を終了させようとしました。

リストされた状況コード (DIBSTAT) は例外条件を示すものであり、プログラムに戻されるのはこれらの状況コードだけです。その他の状況コードはすべてエラー条件を示し、トランザクションまたはバッチ・プログラムを異常終了させます。CICS プログラムからエラー・ルーチンに制御を渡す場合には、CICS HANDLE ABEND コマンドを出すことができます。異常終了コードの最後の 2 バイトが、異常終了の原因となった IMS 状況コードです。HANDLE ABEND コマンドの詳細については、ご使用のバージョンの CICS のアプリケーション・プログラミング解説書を参照してください。バッチ BMP プログラムは、異常終了 1041 で異常終了します。

3. セグメント名 **(DIBSEGM)**

   これは、正常にアクセスされた最も低いレベルのセグメントの名前です。リトリーブが成功すると、このフィールドにリトリーブされたセグメントの名前が入ります。リトリーブが失敗すると、このフィールドには、要求されたセグメントに通じるパスに沿う、このコマンドを正しく実行した最後のセグメントが入ります。

   XRST コマンドを出すと、このフィールドにはブランク (正常に始動したことを示す) か、チェックポイント ID (プログラムが再始動したポイントのチェックポイント ID を示す) が設定されます。

   以下のいずれかのコマンドを出した後で、このフィールドをテストします。

   - GN
   - GNP
   - GU
   - ISRT
   - LOAD
   - RETRIEVE
   - XRST

4. セグメント・レベル番号 **(DIBSEGLV)**

これは、リトリーブされた最低レベル・セグメントの階層レベルです。 IMS DB は、要求されたセグメントをリトリーブするときに、セグメントのレベル番号を文字形式でこのフィールドに入れます。ユーザーがパス・コマンドを出すと、IMS DB はリトリーブされた最低レベルのセグメントの番号を格納します。 IMS DB は、要求されたセグメントを見つけられなかった場合には、IMS DB が検出した、コマンドを正しく実行する最後のセグメントのレベル番号を戻します。これは、要求されたセグメントの検索時に IMS DB が検出した最後のパスの最低レベル・セグメントです。

以下のいずれかのコマンドを出した後で、このフィールドをテストします。

- GN
- GNP
- GU
- ISRT
- LOAD
- RETRIEVE

5. キー・フィードバック長 **(DIBKFBL)**

   これはハーフワードのフィールドであり、このフィールドには、読み取りコマンドで KEYFEEDBACK オプションを使用したとき、連結キーの長さが入ります。 ユーザーが指定したキー・フィードバック域の長さでは連結キーが入りきらない場合、キーは切り捨てられ、この区域は完全な連結キーの実際の長さを示します。

6. データベース記述名 **(DIBDBDNM)**

   これは、フルワードのフィールドで、DBD の名前が入ります。 DBD は、データベースを記述するときに使用されるすべての情報を格納する DL/I 制御ブロックです。 DIBDBDNM フィールドは、QUERY コマンドによってのみ戻されます。

7. データベース編成 **(DIBDBORG)**

   これは、フルワードのフィールドで、データベース編成のタイプ (HDAM、HIDAM、HISAM、HSAM、GSAM、SHSAM、INDEX、または DEDB) を指定します。そのタイプの右側には、ブランクが埋め込まれています。 DIBDBORG フィールドは、QUERY コマンドによってのみ戻されます。

キー・フィードバック域の定義

セグメントの連結キーをリトリーブするためには、キーを入れる区域を定義しなければなりません。

戻される連結キーは、リトリーブされた最低レベル・セグメントのキーです。(リトリーブされたセグメントは、DIB の中の DIBSEGM フィールドおよび DIBSEGLV フィールドに表示されます。)

読み取り (GET) コマンドで KEYFEEDBACK オプションを指定して、キー・フィードバック域名を指定します。

連結キーは、セグメントのキーと、その親のすべてに対するキーとで構成されます。 例えば、患者番号が 05142 で日付が 1988 年 1 月 2 日の ILLNESS セグメントの連結キーを要求したものとします。**0514219880102** が、キー・フィードバック・フィールドに戻されます。この数は、PATIENT セグメントのキー・フィールド (PATNO) に、ILLNESS セグメントのキー・フィールド (ILLDATE) が連結されたものです。

ユーザーが定義したキー・フィードバック域が短すぎて連結キー全体が入らないと、このキーは切り捨てられます。

入出力域の定義

入出力域は、プログラムとデータベースとの間でセグメントをやりとりするために使用します。

入出力域の内容は、実行されるコマンドの種類によって異なります。

- セグメントをリトリーブするとき、DL/I は要求されたセグメントを入出力域に入れます。
- 新しいセグメントを追加する場合は、ISRT コマンドを出す前に入出力域でこの新しいセグメントを作成しておきます。
- セグメントを変更する前に、まず入出力域にセグメントをリトリーブし、次に DLET または REPL コマンドを発行します。

制約事項: 入出力域の長さは、データベースからリトリーブするまたはデータベースに追加する最長セグメントが入るサイズでなければなりません。 (この長さがないと、ストレージのオーバーラップが発生します。)1 つのコマンドで複数のセグメントをリトリーブ、追加、または置換する場合は、セグメントごとに 1 つの入出力域を定義してください。

セグメントが入出力域でどのように見えるかを示す例として、1988 年 3 月 3 日に診療所に来た Robert James の ILLNESS セグメントをリトリーブしたとします。彼は、咽頭炎のため治療を受けました。入出力域に戻されたデータは、次のとおりです。

```
19880303STREPTHROA
```

COBOL 入出力域

COBOL プログラムでは、入出力域を 01 レベルの作業ストレージ項目として定義してください。 この区域を 02 項目でさらに定義することができます。

```
IDENTIFICATION DIVISION.
⋮
DATA DIVISION.
WORKING-STORAGE SECTION.
01    INPUT-AREA.
    02 KEY PICTURE X(6).
    02 FIELD PICTURE X(84).
```

PL/I 入出力域

PL/I では、DL/I 呼び出しに使用される入出力域の名前は、固定長文字ストリング、大構造、接続配列、または調整可能文字ストリングの名前でなければなりません。

制約事項: PL/I 入出力域は、小構造の名前、または属性 VARYING が指定された文字ストリングの名前にすることはできません。入出力域を小構造として定義したい場合は、小構造を指すポインターをパラメーターとして使用することができます。

プログラムは、入出力域を固定長文字ストリングとして定義してから、その文字ストリングの名前を渡すか、あるいは前述の方法の 1 つで入出力域を定義してから、その定義を指すポインター変数を渡さなければなりません。サブストラクチャーまたは配列エレメントを使用したい場合は、DEFINED または BASED 属性を使用してください。

```
DECLARE    1 INPUT_AREA,
           2 KEY CHAR(6),
           2 FIELD CHAR(84);
```

アセンブラー言語入出力域

アセンブラー言語プログラムの入出力域は、次のようにフォーマット設定されます。

```
IOAREA   DS   0CL90
KEY      DS   CL6
FIELD    DS   CL84
```

第 31 章 アプリケーション・プログラム用の EXEC DLI コマンド

EXEC DLI コマンドは、アプリケーション・プログラム内で、プログラム仕様ブロック (PSB)、およびさまざまな種類のプログラム連絡ブロック (PCB) と一緒に使用されます。

関連資料:

599 ページの『アプリケーション・インターフェース・ブロック (AIB) の指定』

PCB および PSB

DBCTL 環境で使用されるプログラム仕様ブロック (PSB) には、I/O PCB、代替 PCB、データベース PCB (DB PCB)、または GSAM PCB を含めることができます。

I/O PCB

DBCTL 環境では、DBCTL サービス要求を出すためには I/O PCB が必要です。他のタイプの PCB とは異なり、I/O PCB は PSB の生成時に定義されませんが、アプリケーション・プログラムが I/O PCB を使用している場合は、その旨を PSB スケジューリング要求の中で示さなければなりません。

代替 PCB

代替 PCB は論理端末を定義するので、応答を端末に送る必要がある場合には、I/O PCB の代わりに代替 PCB を使用することができます。代替 PCB は、CICS-DBCTL 環境で使用される PSB に含まれますが、IMS DC 環境でのみ使用されます。DBCTL を使用する CICS アプリケーションでは、代替 PCB、MSDB PCB、または GSAM PCB を指定するコマンドを出しても成功しません。しかし、このようなタイプの PCB を含む PSB を CICS-DBCTL 環境でスケジュールすることができます。

代替 PCB は、呼び出しレベルのアプリケーション・プログラムに戻される PCB アドレス・リストに含まれます。 EXEC DLI アプリケーション・プログラムでは、PSB の中に代替 PCB があると、PCB キーワードで使用される PCB 数に影響します。

DB PCB

DB PCB は、アプリケーション・プログラムの、データベースとのインターフェースを定義する PCB です。アプリケーション・プログラムによって使用されるデータベース視点ごとに、1 つの DB PCB が必要です。 それは、全機能 PCB、DEDB PCB、または MSDB PCB のいずれかです。

GSAM PCB

GSAM PCB は、アプリケーション・プログラムの GSAM 操作用インターフェースを定義します。

DBCTL を使用すると、デフォルトで CICS プログラムは、スケジュール後にプログラムに渡されるパラメーター・リストの最初の PCB として DB PCB を受け取ります。ただし、アプリケーション・プログラムが I/O PCB を処理できる場合は、ユーザーが SCHD コマンドに SYSSERVE キーワードを指定して、このことを示します。 したがって、I/O PCB が、アプリケーション・プログラムに戻されるパラメーター・アドレス・リスト内の最初の PCB になります。

さまざまなタイプのアプリケーション・プログラムにおける I/O PCB および代替 PCB

DB バッチ・プログラム

ユーザーが CMPAT=Y を指定したかどうかにかかわらず、DL/I によってプログラムに提供される PCB のリストには、必ず代替 PCB が含まれます。I/O PCB は、CMPAT オプションの指定に従って戻されます。

CMPAT=Y を指定すると、PCB リストには、I/O PCB のアドレス、代替 PCB のアドレス、および DB PCB のアドレスがこの順序で含まれます。

CMPAT=Y を指定しないと、PCB リストには、代替 PCB のアドレス、その後に DB PCB のアドレスが含まれます。

BMP プログラム、***MPP***、および ***IFP***

I/O PCB と代替 PCB は、常に BMP プログラムに渡されます。また、I/O PCB と代替 PCB は、常に MPP アプリケーション・プログラムと IFP アプリケーション・プログラムにも渡されます。

PCB リストには、I/O PCB のアドレス、代替 PCB のアドレス、および DB PCB のアドレスがこの順序で含まれます。

DBCTL を使用する ***CICS*** プログラム

最初の PCB は、SYSSERVE キーワードの指定の有無にかかわらず、常に最初の DB PCB を参照します。

次の表では、I/O PCB と代替 PCB の情報を要約しています。最初の列は各種の DB 環境を示しています。2 番目の列と 3 番目の列は、指定された環境で I/O PCB と代替 PCB がそれぞれ有効かどうかを示しています。

表 *87. PCB* 情報の要約

環境	**EXEC DLI: PCB(n)** に含まれる **I/O PCB** カウント	**EXEC DLI: PCB(n)** に含まれる代替 **PCB** カウント
CICS DBCTL[1]	なし	なし
CICS DBCTL[2]	なし	なし
BMP	あり	あり

表 *87. PCB* 情報の要約 (続き)

環境	**EXEC DLI: PCB(n)** に含まれる **I/O PCB** カウント	**EXEC DLI: PCB(n)** に含まれる代替 **PCB** カウント
Batch[3]	なし	あり
Batch[4]	あり	あり

注**:**

1. SYSSERVE オプションなしの SCHD コマンドが出された場合
2. SYSSERVE オプションを指定した SCHD コマンドが、CICS の DBCTL コマンドに対して、または DBCTL を使用するリモート CICS システムにより実行される機能シップ済みコマンドに対して出された場合
3. PSBGEN ステートメント上で CMPAT=N が指定された場合
4. PSBGEN ステートメント上で CMPAT=Y が指定された場合

PSB の形式

PSB のフォーマットは次のとおりです。

```
[IOPCB]
[Alternate PCB ... Alternate PCB]
[DBPCB ... DBPCB]
[GSAMPCB ... GSAMPCB]
```

各 PSB には、少なくとも 1 つの PCB がなければなりません。I/O PCB は、システム・サービス・コマンドを実行するためにアドレス可能でなければなりません。代替 PCB は、IMS オンライン・プログラムに対してのみ使用されますが、このタイプのプログラムは、トランザクション・マネージャーと一緒にしか実行できません。プログラムをトランザクション・マネージャーの下で実行しない場合でも、代替 PCB を使用できることに注意してください。DB PCB は、全機能 PCB、DEDB PCB、または MSDB PCB のいずれであってもかまいません。

第 32 章 データベースのリカバリーとデータベース保全性の維持

これらのコマンドを発行して、プログラムにアクセスされるデータをリカバリーし、データ保全性を維持することができます。

- 基本チェックポイント・コマンド CHKP。このコマンドは、バッチ・プログラムまたは BMP プログラムからチェックポイントを出すときに使用します。
- シンボリック・チェックポイント・コマンド SYMCHKP。このコマンドは、バッチ・プログラムまたは BMP プログラムからチェックポイントを出す場合、およびプログラムの再始動時に復元できるデータ域を指定する場合に使用します。
- 拡張再始動コマンド XRST。このコマンドをシンボリック・チェックポイントとともに使用することにより、バッチ・プログラムまたは BMP プログラムを開始または再始動することができます。
- ロールバック・コマンド ROLL と ROLB。この 2 つのコマンドは、バッチ・プログラムまたは BMP プログラムからデータベースの変更内容を動的にバックアウトするときに使用します。
- バックアウト・ポイント管理コマンド SETS と ROLS。この 2 つのコマンドは複数のバックアウト・ポイントを設定して、後でこれらのポイントまで戻るために使用します。
- デキュー・コマンド DEQ。これは、前に予約されたセグメントを解放します。

上記のコマンドを使用する場合は、前もってプログラムの I/O PCB を定義しておく必要があります。ただし、 DEDB PCB に対して出される DEDB DEQ 呼び出しは除きます。

バッチ・プログラムまたは BMP プログラムでのチェックポイントの出し方

チェックポイントを指定する場合に使用できるコマンドは、基本チェックポイント・コマンドの CHKP と、シンボリック・チェックポイント・コマンドの SYMCHKP です。

バッチ・プログラムでは、シンボリック・チェックポイント・コマンドまたは基本チェックポイント・コマンドのどちらかを使用することができます。

どちらのチェックポイント・コマンドを使用しても、プログラムの変更内容をデータベースにコミットし、異常終了した場合にバッチ・プログラムまたは BMP プログラムが再始動する位置を設定できるようになります。

要件: IMS のチェックポイントを取るために、DD ステートメントに CHKPT=EOV パラメーターを指定してはなりません。

どちらのチェックポイント・コマンドでも、コマンドが出された時点でデータベースの位置が失われてしまうため、GU コマンドまたはその他の方法を使用して、位置を再設定してください。

非ユニーク・キーをもつセグメントまたはキーなしセグメントでは、位置の再設定はできません。

CHKP コマンドの出し方

CHKP コマンドを出す場合には、プログラム再始動のためのコードとチェックポイントの ID を指定してください。 この場合、ID が入っているプログラム内のデータ域の名前を指定するか、または実際の ID を単一引用符で囲んで指定することができます。例えば、次のコマンドはいずれも有効です。

```
EXEC DLI CHKP ID(chkpid);
EXEC DLI CHKP ID('CHKP0007');
```

SYMCHKP コマンドの出し方

バッチ・プログラムおよび BMP プログラムでは、SYMCHKP コマンドは次の操作を行います。

- プログラムが異常終了した場合、拡張再始動 (XRST) コマンドを使用してプログラムを再始動します。
- プログラムの再始動時に復元される、プログラムのデータ域を 7 つまで保管できます。 これには、変数、カウンター、状況情報を保管することができます。

SYMCHKP コマンドの指定法の例については、「*IMS V15* アプリケーション・プログラミング *API*」のトピック『SYMCHKP コマンド』を参照してください。

プログラムの再始動および位置の検査

シンボリック・チェックポイント・コマンドを出すプログラムは、拡張再始動 (XRST) コマンドも出さなければなりません。 プログラムの最初のコマンドとして、XRST を 1 回出してください。 XRST コマンドを使用して、プログラムを正常に開始させたり、あるいは異常終了した場合に再始動させることができます。

次のいずれかから、プログラムを再始動することができます。

- 特定のチェックポイント ID
- 時刻/日付スタンプ

XRST コマンドがデータベースの位置変更を行おうとするため、プログラムはその位置が正しいかどうかも調べる必要があります。

データベース更新の動的バックアウト: ROLL および ROLB コマンド

バッチ・プログラムで、一部の処理が無効であることが検出された場合、 ROLL コマンドと ROLB コマンドを使用すると不正な処理の影響が除去されます。

バッチ・プログラムでは ROLL と ROLB の両方を使用することができます。 システム・ログが直接アクセス・ストレージに保管されており、JCL のパラメーター・フィールドに BKO=Y を指定している場合は、バッチ・プログラムでは ROLB コマンドだけしか使用できません。

このコマンドのどちらかを出すと、最後のチェックポイントから、あるいはプログラムがチェックポイントを出さなかった場合はプログラムの最初から、プログラムがデータベースに行った変更をすべて DL/I がバックアウトします。

中間バックアウト・ポイントの使用: SETS および ROLS コマンド

SETS コマンドと ROLS コマンドを使用すると、DL/I 全機能データベースの状態を保存するポイントを複数定義し、これらのポイントへ後で戻ることができます。例えば、これらのコマンドを使用すると、PSB のスケジュールの完了時に参照される DL/I データベースが使用できない場合に発生する可能性のある状況を、プログラムが処理できるようになります。

SETS および ROLS コマンドは、DL/I 全機能データベースのみに適用されます。つまり、作業論理単位 (LUW) が全機能データベース以外のリカバリー可能リソース (VSAM ファイルなど) を更新する場合、SETS 要求と ROLS 要求は非 DL/I リソースには影響を及ぼしません。 バックアウト・ポイントは CICS コミット・ポイントではありません。それらは、DBCTL リソースだけに適用される中間バックアウト・ポイントです。ユーザー・プログラムによって、関連リソースのすべてに整合性をもたせるようにしてください。

ある機能を実行する一連の DL/I 要求を開始する前に、SETS コマンドを使用して、アプリケーション内で DL/I データベースの状態を保存するポイントを定義することができます。 アプリケーションは、この機能を完了できない場合に後から ROLS コマンドを出すことができます。特定の SETS 要求を出す前、あるいは最後のコミット・ポイントよりも前のすべての全機能データベースの状態にバックアウトするときに、ROLS コマンドを使用できます。

第 33 章 高速機能データベースの処理

DBCTL のもとで EXEC DLI コマンドを使用すると、CICS プログラムまたはバッチ指向 BMP プログラムは、DEDB にアクセスすることができます。パラメーターを指定すると、サブセット・ポインターなどの DEDB の機能をユーザー・プログラムで使用することができます。

DEDB には、1 つのルート・セグメントと 127 タイプの従属セグメントが入っています。 127 タイプのうちの 1 つは順次従属にすることができますが、他の 126 個は直接従属です。 順次従属セグメントは日時順に保管され、直接従属セグメントは階層的に保管されます。

DEDB は、高データ可用性を提供します。各 DEDB は、複数のエリア に区画分けまたは分割することができます。各エリアには、異なるデータベース・レコードのセットが入っています。 さらに、各エリアのデータ・セットのコピーを 7 つまで作成することができます。 エリアのコピーの 1 つにエラーがあっても、そのエリアのアプリケーション・プログラムは他のコピーを使用してデータにアクセスすることができます。 このことは、アプリケーション・プログラムには影響を与えません。 DEDB 内のデータにエラーが発生しても、IMS は、データベースを停止しません。エラーのあるデータは使用不能になりますが、アプリケーション・プログラムのスケジューリングと処理は続行されます。エラーのあるデータを必要としないプログラムには、影響がありません。

別々の IMS システム内のアプリケーション・プログラム間で、DEDB を共用することができます。DEDB を共用するということは、実質的に全機能データベースを共用することと同じなので、同じ規則のほとんどが適用されます。 複数の IMS システムが、エリア・レベル (全機能データベースにおけるデータベース・レベルではない) で DEDB を共用するか、あるいはブロック・レベルで DEDB を共用することができます。

サブセット・ポインター・オプションを使用した高速機能 DEDB の処理

サブセット・ポインター、およびそれとともに使用するオプションは、長いセグメント・チェーンを処理しなければならないときに、プログラムの効率を大きく向上させる最適化ツールです。

サブセット・ポインターは、同じ親のもとにあるセグメント・オカレンスのチェーンを、複数のグループ (すなわち、サブセット) に分割します。 1 つのセグメント・タイプについて、8 つまでのサブセット・ポインターを定義できます。そのうえで、アプリケーション・プログラム内でそのサブセット・ポインターを定義します。各サブセット・ポインターは、新しいサブセットの開始位置を指します。 例えば、次の図で、最後の 3 つのセグメント・オカレンスを最初の 4 つのセグメント・オカレンスから分ける、1 つのサブセット・ポインターを定義したと仮定します。こうすることで、プログラムは、オプションを使用してそのサブセット・ポインターを参照し、最後の 3 つのセグメント・オカレンスを直接リトリーブすること

ができます。

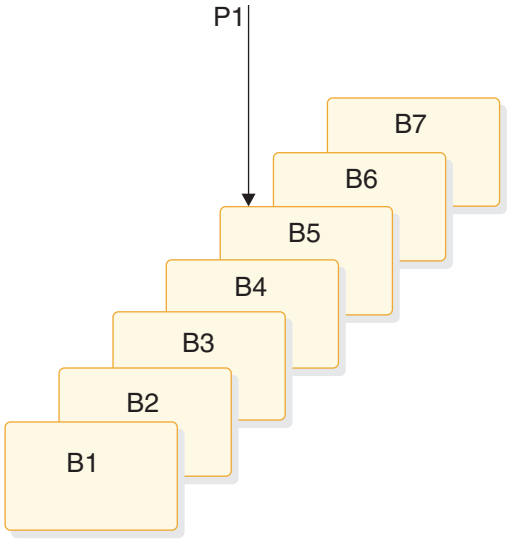


図 *88.* サブセット・ポインターを使用したセグメント・オカレンスの長いチェーンの処理

サブセット・ポインターは、ルート・レベルを除いて、データベース階層のどのレベルでも使用することができます。 ルート・レベルで使用されるサブセット・ポインターは無視されます。

次の 2 つの図では、サブセット・ポインターを設定するいくつかの方法を示しています。サブセット・ポインターは、互いに独立しているので、チェーンの中のどのセグメントに対しても 1 つ以上のポインターを設定することができます。 例えば、次の図に示すように、1 つのセグメントに複数のサブセット・ポインターを設定できます。

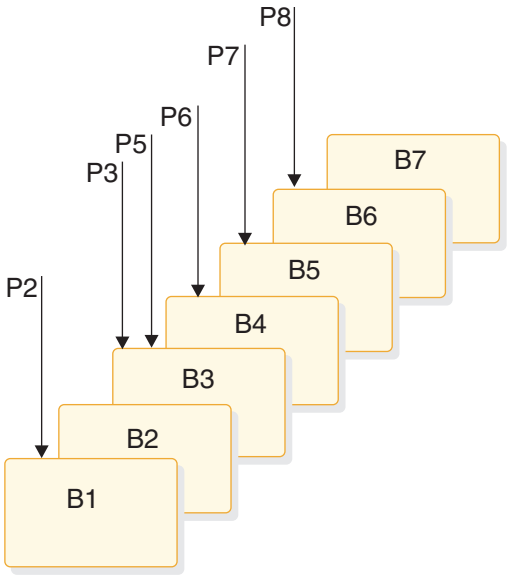


図 *89.* 複数のサブセット・ポインターの設定例

あるいは、図 90 に示すように、ポインターとセグメントとの間に 1 対 1 の関係を定義することもできます。この図ではセグメント・オカレンスごとに 1 つのサブセット・ポインターがあります。

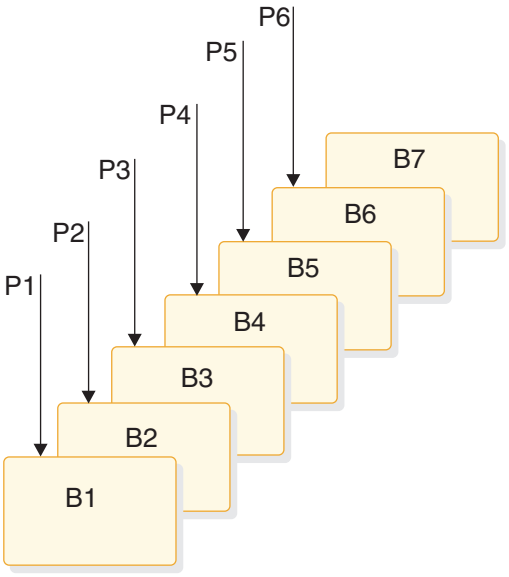


図 *90.* サブセット・ポインター設定の追加例

次の図は、同じ親の下にあるセグメント・オカレンスのチェーンが、サブセット・ポインターの使用によりどのように分けられるかを示しています。各サブセットはチェーン全体の最後のセグメントで終わります。 例えば、サブセット・ポインター 1 によって定義されるサブセットの中の最後のセグメントは B7 です。

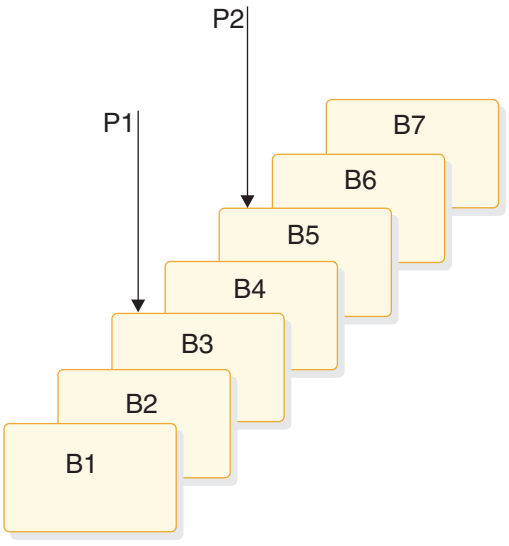


図 *91.* サブセット・ポインターがチェーンをサブセットに分ける方法

サブセット・ポインターの使用の準備

プログラムでサブセット・ポインターを使用するには、これらのポインターを DEDB の DBD、およびプログラムの PSB の中で定義しておく必要があります。

DBD には、1 つのセグメント・チェーンに使用されるポインター数を指定します。どのセグメント・チェーンに対しても最高 8 つまでポインターを指定することができます。

PSB には、プログラムが使用するポインターを指定します。この定義は、 SENSEG ステートメントの中で行います。 (各ポインターは 1 から 8 までの整数で定義します。) また、プログラムが、使用するポインターを設定できるかどうかについても、 SENSEG ステートメントで指定します。 プログラムが読み取り専用センシティビティーである場合は、ポインターを設定できませんが、既に設定されたサブセット・ポインターを使用して、セグメントをリトリーブすることだけはできます。プログラムが更新センシティビティーである場合、SET、SETCOND、MOVENEXT、および SETZERO の各オプションを使用して、サブセット・ポインターを更新することもできます。

DBD と PSB の中でポインターを定義した後、アプリケーション・プログラムは、ポインターをチェーンの中のセグメントに設定することができます。 アプリケーション・プログラムが実行を終了すると、そのプログラムによって使用されたサブセット・ポインターは、設定されたままになり、再設定されません。

サブセット・ポインターの指定

プログラムの中でサブセット・ポインターを使用するためには、PSB の中で定義されたポインターの番号が分かっていなければなりません。

サブセット・ポインターのオプションを使用する場合は、使用する各サブセット・ポインターの番号をオプションの直後に指定します。例えば、サブセット・ポインター 3 によって定義されたサブセット中の最初のセグメント・オカレンスをリトリーブすることを示すときは、P3 を使用します。デフォルトがないため、ユーザーが 1 から 8 までの番号を指定しないと、IMS は修飾ステートメントが無効であると見なし、AJ 状況コードをプログラムに戻します。

サブセット・ポインターのオプション

サブセットを利用するために、アプリケーション・プログラムは 5 つの異なるオプションを使用します。

オプションは次のとおりです。

GETFIRST
サブセットにある最初のセグメントをリトリーブすることができます。

SETZERO
サブセット・ポインターをゼロに設定します。

MOVENEXT
サブセット・ポインターを、現行セグメントの次のセグメントに設定します。 現在位置は現行セグメントです。

SET 無条件で、サブセット・ポインターを現行セグメントに設定します。現在位置は現行セグメントです。

SETCOND
条件付きで、サブセット・ポインターを現行セグメントに設定します。現在位置は現行セグメントです。

銀行用トランザクション・アプリケーションの例

この章での例は、通帳預金の銀行用トランザクションの記録に関するサンプル・アプリケーションに基づいています。トランザクションは、カスタマーの通帳に記帳されたかどうかに基づいて、記帳済みまたは未記帳としてデータベースに書き込まれます。例えば、Bob Emery 氏は、銀行と取引をしていますが、通帳を持参するのを忘れたとします。アプリケーション・プログラムは、トランザクションを未記帳としてデータベースに書き込みます。 アプリケーション・プログラムは、サブセット・ポインターを、最初の未記帳トランザクションに設定するので、後で簡単にアクセスすることができます。次に Bob 氏が通帳を持参したときに、プログラムはトランザクションを記帳します。 プログラムは、前に設定されたサブセット・ポインターを使用して、最初の未記帳トランザクションを直接リトリーブできます。プログラムは、トランザクションを記帳した後で、サブセット・ポインターを 0 に設定します。これによって、それ以降にデータベースを更新するアプリケーション・プログラムは、未記帳のトランザクションがないと判別できるようになります。次の図は、通帳が使用不可である場合に行われる処理の要約を示しています。

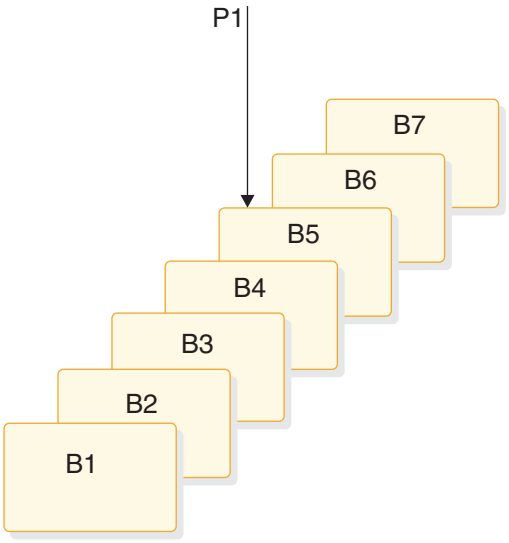


図 92. 通帳に対して実行される処理の例 (通帳が使用不可の場合)

通帳が使用可能である場合、アプリケーション・プログラムは未記帳トランザクションをデータベースに追加し、サブセット・ポインター 1 を最初の未記帳トランザクションに設定します。次の図は、通帳が使用可能である場合に行われる処理の要約を示しています。

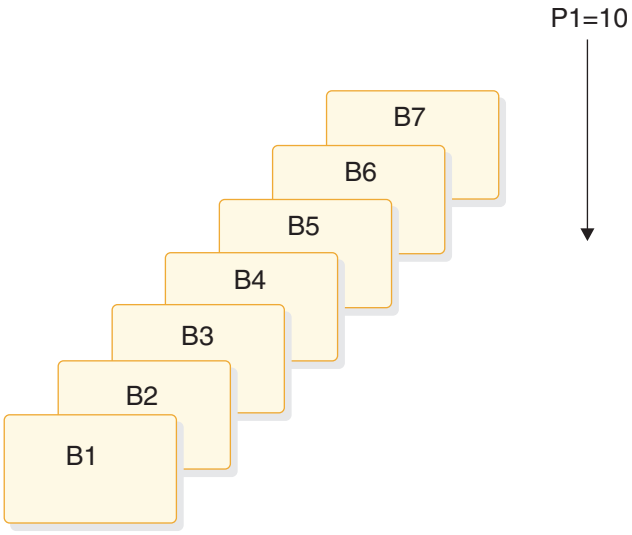


図 93. 通帳に対して実行される処理の例 (通帳が使用可能な場合)

通帳が使用可能な場合、アプリケーション・プログラムは、このプログラムを使用した最初の未記帳トランザクションをリトリーブします。その後、すべての未記帳トランザクションを記帳し、サブセット・ポインター 1 をゼロに設定します。

GETFIRST オプション: サブセットの最初のセグメントのリトリーブ

サブセットの中の最初のセグメント・オカレンスをリトリーブするために、プログラムは、GETFIRST オプションが指定された読み取りコマンドを出します。GETFIRST オプションはポインターを設定したり動かしたりしませんが、IMS に対して、サブセット中の最初のセグメント・オカレンスに位置を設定するように指示します。 GETFIRST オプションは、FIRST オプションに似ていますが、GETFIRST オプションは、セグメント・チェーン全体に適用されるのではなく、サブセットに適用されます。

前の例を使用して、Bob Emery 氏が通帳を持参して銀行に行き、すべての未記帳トランザクションを記帳する場合を想定します。サブセット・ポインター 1 は既に、最初の未記帳トランザクションに設定されているので、プログラムで、以下のコマンドを使用して、そのトランザクションをリトリーブすることができます。

```
EXEC DLI GU SEGMENT(A) WHERE(AKEY = 'A1')
     SEGMENT(B) INTO(BAREA) GETFIRST('1');
```

次の図に示されているように、このコマンドはセグメント B5 をリトリーブします。チェーンの中のセグメントを処理し続けるには、サブセット・ポインターを使用していない場合と同じように、GET NEXT コマンドを出すことができます。

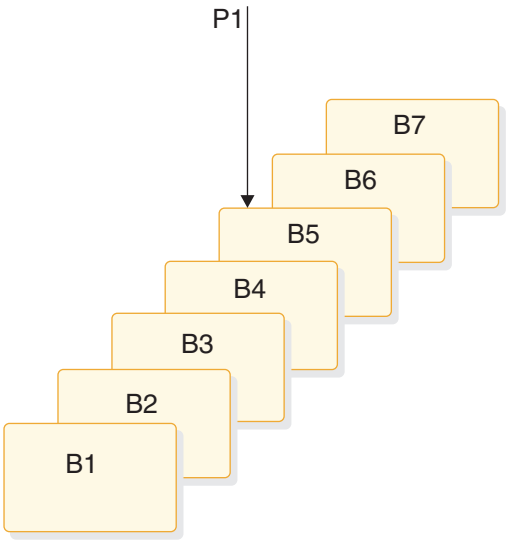


図 *94.* セグメント・チェーン内の最初のセグメントのリトリーブ

サブセットが存在していない (サブセット・ポインター 1 がゼロに設定されている) 場合、IMS は GE 状況コードを戻し、データベース内の位置は、チェーンの最後のセグメントの直後になります。通帳の例では、GE 状況コードは未記帳トランザクションがないことを示しています。

1 つの修飾ステートメントに指定できる GETFIRST オプションは 1 つのみです。修飾ステートメントに複数の GETFIRST を使用すると、IMS は、ユーザーのプログラムに AJ 状況コードを戻します。GETFIRST オプションの使用に関する規則は、以下のとおりです。

1. GETFIRST は次のオプションを除くすべてのオプションとともに使用できます。
   - FIRST
   - LOCKCLASS
   - LOCKED
2. 他のオプションは、GETFIRST オプションが有効になり、サブセット中の最初のセグメントに位置が設定されてから有効になります。
3. GETFIRST を LAST とともに使用した場合、セグメント・チェーンの最後のセグメントがリトリーブされます。
4. GETFIRST に対して指定したサブセット・ポインターが設定されていない場合、IMS は GE 状況コードを戻し、セグメント・チェーンの最後のセグメントを戻しません。
5. GETFIRST は FIRST とともに使用しないでください。 これによって、AJ 状況コードがリトリーブされます。
6. GETFIRST は、LAST を含めすべての挿入規則を指定変更します。

SETZERO、MOVENEXT、SET、および SETCOND オプション: サブセット・ポインターの設定

SETZERO、MOVENEXT、SET、および SETCOND の各オプションを使用すると、サブセット・ポインターを変更して、サブセットを再定義することができま

す。 プログラムでサブセット・ポインターを設定するには、その前にデータベース内の位置を設定しておかなければなりません。コマンドが完全に正しく実行されないと、サブセット・ポインターは設定されません。 ポインターがどのセグメントに設定されるかは、コマンドが完了したときの現在位置によって決まります。 セグメントをリトリーブするためのコマンドが完全に正しく実行されておらず、しかも位置が設定されていないと、サブセット・ポインターは、コマンドが出される前の状態に保たれます。

- サブセット・ポインターのゼロへの設定**: SETZERO**

  SETZERO オプションは、サブセット・ポインターの値をゼロに設定します。プログラムが SETZERO オプションを指定したコマンドを出すと、ポインターはセグメントに設定されなくなります。つまり、そのポインターによって定義されたサブセットは存在しなくなります。(ゼロの値をもつサブセット・ポインターを使用しようとすると、GE 状況コードが IMS からプログラムに戻されます。)

  前の例を使用して、最初の未記帳トランザクションをリトリーブするために、GETFIRST オプションを使用したとします。それからセグメント・チェーンを処理して、トランザクションを記帳します。トランザクションを記帳し、新しいトランザクションをチェーンに挿入した後、次のコマンドで示すように、SETZERO オプションを使用してサブセット・ポインターをゼロに設定します。

  ```
  EXEC DLI ISRT SEGMENT(A) WHERE(AKEY = 'A1')
       SEGMENT(B) FROM(BAREA) SETZERO('1');
  ```

  このコマンドの実行後、サブセット・ポインター 1 は 0 に設定されます。これは、データベースを後で更新するプログラムに、未記帳のトランザクションがないことを示します。

- 現在位置の次のセグメントへのサブセット・ポインターの移動**: MOVENEXT**

  サブセット・ポインターを現在位置の次のセグメントに移動するには、プログラムは MOVENEXT オプションを指定したコマンドを出します。前の例を使用して、いくつかのトランザクションを記帳し (全部ではない)、サブセット・ポインターを最初の未記帳トランザクションに設定したいとします。次のコマンドは、サブセット・ポインター 1 をセグメント B6 に設定します。

  ```
  EXEC DLI GU SEGMENT(A) WHERE(AKEY = 'A1')
       SEGMENT(B) INTO(BAREA) GETFIRST('1') MOVENEXT('1');
  ```

  このコマンドを使用してサブセット・ポインターを移動する処理を次の図に示します。現行セグメントがチェーンの最後のセグメントであって、MOVENEXT オプションを使用した場合、IMS はポインターをゼロに設定します。

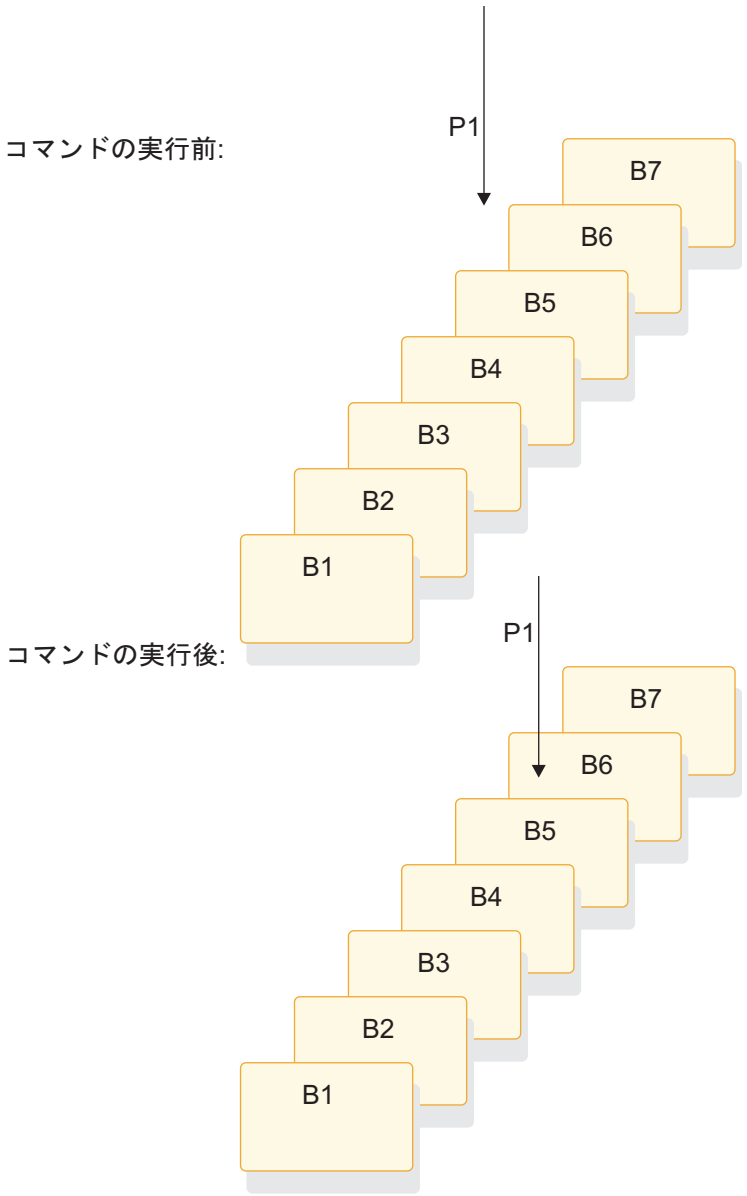


図 *95.* 現在位置の次のセグメントへのサブセット・ポインターの移動

- 無条件でのサブセット・ポインターの設定**: SET**

  SET オプションは、サブセット・ポインターを設定するときに使用します。 SET オプションは、サブセット・ポインターが既に設定されているかどうかにかかわらず、これを無条件で設定します。 プログラムが SET オプションを含むコマンドを出すと、IMS はポインターを現在位置に設定します。

  例えば、サブセット・ポインター 1 によって定義されたサブセットの中の最初の B セグメント・オカレンスをリトリーブし、ポインター 1 を次の B セグメント・オカレンスに再設定したい場合、次のコマンドを出します。

```
EXEC DLI GU SEGMENT(A) WHERE(AKEY = 'A1')
     SEGMENT(B) INTO(BAREA) GETFIRST('1');
EXEC DLI GN SEGMENT(B) INTO(BAREA) SET('1');
```

これらのコマンドを発行した後、サブセット・ポインター 1 はセグメント B5 を指すのではなく、次の図で示すようにセグメント B6 を指します。

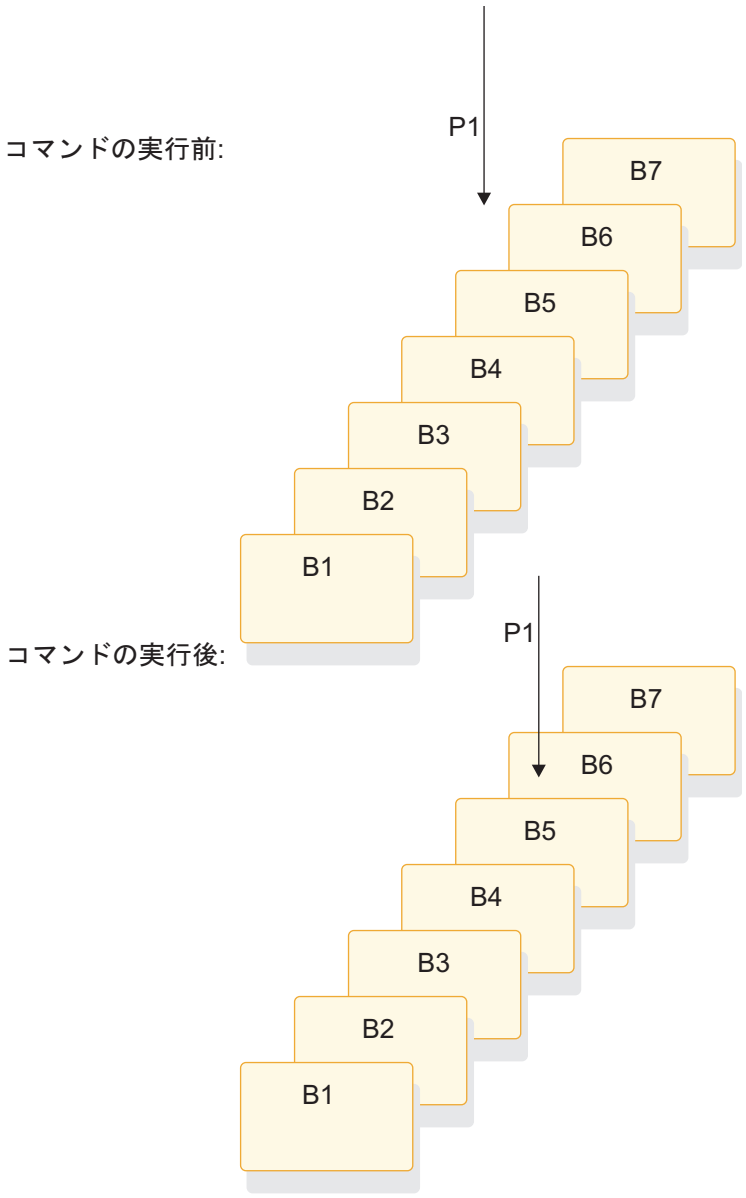


図 *96.* 無条件でのサブセット・ポインターの現行位置への設定

- 条件付きでのサブセット・ポインターの設定**: SETCOND**

  サブセット・ポインターを条件付きで設定するとき、プログラムは SETCOND オプションを使用します。SETCOND オプションは、SET オプションに似ています。唯一の違いは、SETCOND オプションを使用すると、IMS は、サブセット・ポインターがセグメントにまだ設定されていない 場合にのみ、そのポインターを更新することです。

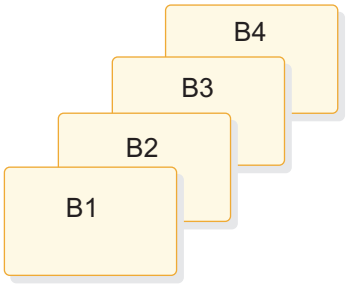
  預金通帳の例において、Bob Emery 氏が銀行に行ったときに、通帳を持参するのを忘れたとします。この場合、データベースに未記帳トランザクションを追加します。 後でトランザクションを記帳するときに、すぐに最初のものにアクセ

スできるように、ポインターを最初の未記帳トランザクションに設定します。次のコマンドは、挿入するトランザクションが最初の未記帳トランザクションである場合に、サブセット・ポインターをこのトランザクションに設定します。

```
EXEC DLI ISRT SEGMENT(A) WHERE(AKEY = 'A1')
     SEGMENT(B) FROM(BAREA) SETCOND('1');
```

次の図に示すように、このコマンドは、サブセット・ポインター 1 をセグメント B5 に設定します。未記帳トランザクションが既に存在している場合、サブセット・ポインターは変更されません。

コマンドの実行前:



コマンドの実行後:

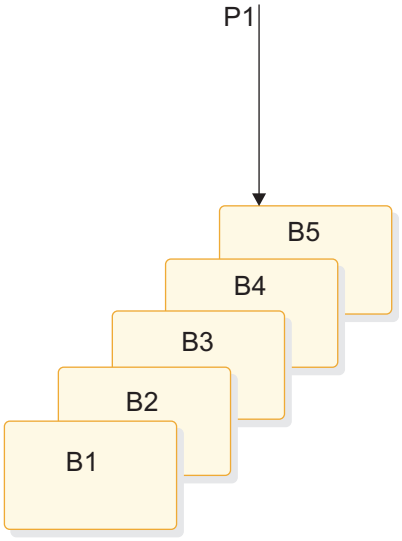


図 97. 条件付きでのサブセット・ポインターの現行位置への設定

サブセットへのセグメントの挿入

キーをもたないセグメントをサブセットに挿入するために GETFIRST オプションを使用すると、この新しいセグメントは、サブセット中の最初のセグメント・オカレ

ンスの前に挿入されます。 しかし、サブセット・ポインターが、新しいセグメント・オカレンスに自動的に設定されるわけではありません。 例えば、次のコマンドは、新しい B セグメント・オカレンスをセグメント B5 の前に挿入しますが、サブセット・ポインター 1 をこの新しい B セグメント・オカレンスを指し示すようには設定しません。

```
EXEC DLI ISRT SEGMENT(A) WHERE(AKEY = 'A1')
     SEGMENT(B) FROM(BAREA) GETFIRST('1');
```

サブセット・ポインター 1 を新しいセグメントに設定するには、次の例に示すように、GETFIRST オプションとともに SET オプションを使用します。

```
EXEC DLI ISRT SEGMENT(A) WHERE(AKEY = 'A1')
     SEGMENT(B) FROM(BAREA) GETFIRST('1') SET ('1');
```

サブセットが存在しない (サブセット・ポインター 1 がゼロに設定された) 場合、セグメントはセグメント・チェーンの最後に追加されます。

サブセット・ポインターに指示されているセグメントの削除

サブセット・ポインターにより指示されているセグメントを削除すると、サブセット・ポインターは、チェーンの中の次のセグメント・オカレンスを指示します。 削除したセグメントがチェーンの最後のセグメントであれば、サブセット・ポインターはゼロに設定されます。

オプションの組み合わせ

オプション間で矛盾しないかぎり、SET、MOVENEXT、SETCOND オプションを他のオプションと組み合わせて使用したり、サブセット・ポインターの複数のオプションを組み合わせて使用することができます。 例えば、GETFIRST と SET は一緒に使用することができますが、 SET と SETZERO は機能が競合するので、一緒に使用することはできません。 競合するオプションを組み合わせると、AJ 状況コードが IMS からプログラムに戻されます。

1 つの修飾ステートメントにつき、1 つの GETFIRST オプションを使用することができ、また各サブセット・ポインターにつき 1 つの更新オプション (SETZERO、MOVENEXT、SET、または SETCOND) を使用することができます。

サブセット・ポインターの状況コード

サブセット・ポインター・オプションを含む修飾ステートメントにエラーがあると、IMS は、次のいずれかの状況コードをプログラムに返します。

AJ DBD でサブセット・ポインターが定義されていないセグメントに対して、GETFIRST、SET、SETZERO、SETCOND、または MOVENEXT オプションを修飾ステートメントが使用しました。

修飾ステートメントに含まれているサブセット・オプションが競合しています。例えば、同じサブセット・ポインターに対する SET オプションと SETZERO オプションが、1 つの修飾ステートメントに指定されている場合、IMS は AJ 状況コードを戻します。 S はポインターを現在位置に設定することを意味し、 Z はポインターをゼロに設定することを意味するので、 これらのオプションを 1 つの修飾ステートメント内で一緒に使用することはできません。

修飾ステートメントに複数の GETFIRST オプションが入っていました。

サブセット・ポインター・オプションの後のポインター番号が無効です。ユーザーが番号を入れなかったか、無効文字が含まれていたかのどちらかです。オプションの次の番号は、1 から 8 までの番号でなければなりませんが、重複してもかまいません。

AM 修飾ステートメントの中で参照されたサブセット・ポインターが、プログラムの PSB で指定されていませんでした。 例えば、プログラムの PSB に、プログラムでサブセット・ポインター 1 と 4 を使用できることが指定されているときに、修飾ステートメントがサブセット・ポインター 5 を参照した場合、IMS は AM 状況コードをプログラムに戻します。

プログラムが、ポインターを更新するオプション (SET、SETCOND、または MOVENEXT) を使用しようとしましたが、プログラムの PSB はポインターが更新センシティビティーであるとは指定していませんでした。

ご使用のプログラムは、IOAREA を指定せずに GSAM データベースを開こうとしました。

POS コマンド

位置 (POS) コマンドを使用すると、次のような操作を行うことができます (DEDB のみ)。

- 特定の順次従属セグメントの位置、または最後に挿入された順次従属セグメントの位置のリトリーブ。
- 各 DEDB エリア内の未使用スペースの容量の通知。 例えば、POS コマンドの実行後に IMS が戻す位置情報を使用して、特定の期間にある順次従属セグメントをスキャンまたは削除することができます。

POS コマンドの構文については、「*IMS V15* アプリケーション・プログラミング *API*」のトピック『POS コマンド』を参照してください。

POS コマンドによって指定されたエリアを使用できない場合は、入出力域は変更されず、状況コード FH が戻されます。

関連資料:

POS コマンド (「アプリケーション・プログラミング API」)

特定の順次従属セグメントの位置指定

特定のルート・セグメント上に位置を設定した場合は、位置情報と、そのルートの特定の順次従属セグメントの区域名をリトリーブできます。

順次従属セグメント上に位置を設定した場合には、検索はその位置から始まります。 IMS は、コマンドを正しく実行させる最初の順次従属セグメントの位置情報を戻します。

この情報をリトリーブする場合は、順次従属セグメントの名前が入っている修飾ステートメントをもつ POS コマンドを出します。このように POS コマンドを実行した後の現在位置は、GNP コマンド実行後の現在位置と同じ位置になります。

POS コマンドが正常に実行されると、入出力域の内容は次のようになります。

LL 入出力域内のデータ全体の長さを示す 2 バイトのフィールド (2 進数)

区域名
AREA ステートメントで指定された DD 名を示す 8 バイトのフィールド

位置 要求されたセグメントに対する位置情報が入っている 8 バイトのフィールド

POS コマンドの目標セグメントである順次従属セグメントが同じ同期間隔で挿入されている場合は、位置情報は戻されません。 バイト 11 から 18 には X'FF'; が入っており、他のフィールドには通常のデータが入っています。

未使用の **CI**
順次従属部分にある未使用の CI 数が入っている 4 バイトのフィールド

未使用の **CI**
独立したオーバーフロー部分にある未使用の CI 数が入っている 4 バイトのフィールド

最後に挿入された順次従属セグメントの位置指定

特定のルート・セグメントの順次従属セグメントの中で、最後に挿入されたセグメントの位置情報をリトリーブすることもできます。

これを行うには、セグメント名としてルート・セグメントが指定されている修飾ステートメントを使用して POS コマンドを発行します。このようにコマンドを実行した後の現在位置は、GU の実行後の位置の場合と同じ規則に従って設定されます。

コマンドが成功した後の入出力域の内容は、次のとおりです。

LL 入出力域内のデータ全体の長さを示す 2 バイトのフィールド (2 進数)

区域名
AREA ステートメントで指定された DD 名を示す 8 バイトのフィールド

位置 最後に挿入された順次従属セグメントに関する位置情報が入っている 8 バイトのフィールド。 このルートに順次従属セグメントがない場合、このフィールドにはゼロが入ります。

未使用の **CI**
順次従属部分にある未使用の CI 数が入っている 4 バイトのフィールド

未使用の **CI**
独立したオーバーフロー部分にある未使用の CI 数が入っている 4 バイトのフィールド

POS コマンドの使用によるフリー・スペースの識別

すべてのオンライン区域から区域名と順次従属セグメント内で使用可能な次の位置をリトリーブする場合は、POS コマンドを修飾せずに出します。このタイプのコマンドは、独立したオーバーフロー部分および順次従属部分にあるフリー・スペースもリトリーブします。

修飾せずに出された POS コマンドが成功すると、入出力域には、データ全体の長さ (LL) と、データベース内の区域と同数の項目が入ります。どの項目にも、次に示すフィールド 2 から 5 までが含まれます。

LL 入出力域内のデータ全体の長さを示す 2 バイトのフィールド (2 進数)。この長さには、 LL フィールドの 2 バイトと、項目ごとに 24 バイトを加えた長さも含まれています。

区域名
AREA ステートメントで指定された DD 名を示す 8 バイトのフィールド

位置 順次従属部分内で次に使用可能な位置を示す 8 バイトのフィールド

未使用の **CI**
順次従属部分にある未使用の CI 数が入っている 4 バイトのフィールド

未使用の **CI**
独立したオーバーフロー部分にある未使用の CI 数が入っている 4 バイトのフィールド

P 処理オプション

プログラムの PCB に P 処理オプションが (PROCOPT パラメーターで) 指定されている場合、セグメントをリトリーブまたは挿入するコマンドによって作業単位 (UOW) の境界を越えると、必ず GC 状況コードがプログラムに戻されます。

UOW 境界を越えることは、プログラムにとっては特に重要ではありませんが、GC 状況コードは CHKP コマンドを出すよいタイミングであることを示します。この操作の利点は次のとおりです。

- データベース内の位置が保持されます。 CHKP を出すと、通常、データベース内の位置が失われるため、アプリケーション・プログラムは処理を再開する前に位置を再設定しなければなりません。
- コミット点が一定の間隔で発生します。

GC 状況コードが戻されると、データはリトリーブも挿入もされません。プログラムでは、次のいずれかを行うことができます。

- CHKP コマンドを出し、GC 状況コードが戻される原因となったコマンドを再度出すことにより、データベースの処理を再開する。
- GC 状況コードを無視し、状況コードが戻される原因となったコマンドを再度出すことにより、データベースの処理を再開する。

第 34 章 コマンド・レベル・プログラムと呼び出しレベル・プログラムの比較

コマンド・レベル・プログラムと呼び出しレベル・プログラムでは、示す動作が異なります。

IMS および CICS の DL/I 呼び出し

次の表に示すのは、バッチ、バッチ指向 BMP、または DBCTL 使用の CICS 環境で DL/I 呼び出しを使用するための参照情報です。

表 *88. IMS* および *CICS* コマンド・レベル・アプリケーション・プログラムで使用可能な *DL/I* 呼び出し：

要求タイプ	バッチ	バッチ指向 **BMP**	**DBCTL** を使用する **CICS**[1]
CHKP 呼び出し (記号)	あり	あり	なし
CHKP 呼び出し (基本)	あり	あり	なし
GSCD 呼び出し[2]	あり	なし	なし
INIT 呼び出し	あり	あり	あり
ISRT 呼び出し (初期ロード)	あり	なし	なし
ISRT 呼び出し	あり	あり	あり
LOG 呼び出し	あり	あり	あり
SCHD 呼び出し	なし	なし	あり
ROLB 呼び出し	あり	あり	なし
ROLL 呼び出し	あり	あり	なし
ROLS 呼び出し (SETS へのロールバック)[3]	あり	あり	あり
ROLS 呼び出し (コミットへのロールバック)	あり	あり	あり
SETS 呼び出し[3]	あり	あり	あり
STAT 呼び出し[4]	あり	あり	あり
TERM 呼び出し	なし	なし	あり
XRST 呼び出し	あり	あり	なし

1. CICS リモート DL/I 環境では、CICS-DBCTL 欄の呼び出しは、DBCTL を使用するリモート CICS への機能シップを行っている場合にサポートされます。
2. GSCD は、プロダクト・センシティブ・プログラミング・インターフェースです。

3. SETS および ROLS 呼び出しは、PSB に DEDB が入っているときには有効でありません。
4. STAT は、プロダクト・センシティブ・プログラミング・インターフェースです。

EXEC DLI コマンドと DL/I 呼び出しの比較

プログラムでは、適切な EXEC DLI コマンドと DL/I 呼び出しを使用します。

次の表では、EXEC DLI コマンドと DL/I 呼び出しとを比較しています。例えば、コマンド・レベル・プログラムでは、データベースを初期ロードするときには ISRT 呼び出しではなく LOAD コマンドを使用します。

表 *89.* 呼び出しレベル・プログラムとコマンド・レベル・プログラムの比較: コマンドと呼び出し :

呼び出しレベル	コマンド・レベル	目的
INIT 呼び出し	ACCEPT コマンド	データ可用性状況コードを初期設定する。
CHKP 呼び出し (基本)	CHKP コマンド	基本チェックポイントを出す。
DEQ 呼び出し	DEQ コマンド	LOCKCLASS オプションまたは Q コマンド・コードを使用してリトリーブされたセグメントを解放する。
DLET 呼び出し	DLET コマンド	データベースからセグメントを削除する。
GU、GN、および GNP 呼び出し	GU、GN、および GNP コマンド[1]	データベースからセグメントをリトリーブする。
GHU、GHN、および GHNP 呼び出し[1]	GU、GN、および GNP コマンド[1]	更新のためにデータベースからセグメントをリトリーブする。
GSCD 呼び出し	GSCD 呼び出し[2]	システム・アドレスをリトリーブする。
ISRT 呼び出し	ISRT コマンド	データベースにセグメントを追加する。
ISRT 呼び出し	LOAD コマンド	データベースを初期ロードする。
LOG 呼び出し	LOG コマンド	システム・ログにメッセージを書き出す。
POS 呼び出し	POS コマンド	DEDB エリアでの位置またはスペース使用状況、あるいはその両方をリトリーブする。
INIT 呼び出し	ACCEPT コマンド	データ可用性状況を初期設定する。
INIT 呼び出し	QUERY コマンド	初期データ可用性情報を入手する。
INIT 呼び出し	REFRESH コマンド	PCB を使用した後の可用性情報
REPL 呼び出し	REPL コマンド	データベースのセグメントを置換する。
XRST 呼び出し	RETRIEVE コマンド	拡張再始動を出す。
ROLL または ROLB 呼び出し	ROLL または ROLB コマンド	変更を動的にバックアウトする。
ROLS 呼び出し	ROLS コマンド	前に設定されたバックアウト・ポイントまでバックアウトする。
PCB 呼び出し	SCHD コマンド	PSB をスケジュールする。
SETS 呼び出し	SETS コマンド	バックアウト・ポイントを設定する。
SETU 呼び出し	SETU コマンド	サポートされていない PCB (DEDB または MSDB など) がある場合にも、バックアウト・ポイントを設定する。
STAT 呼び出し[3]	STAT コマンド	システムおよびバッファー・プール統計を入手する。

表 89. 呼び出しレベル・プログラムとコマンド・レベル・プログラムの比較: コマンドと呼び出し (続き) :

呼び出しレベル	コマンド・レベル	目的
CHKP 呼び出し (拡張)	SYMCHKP コマンド	シンボリック・チェックポイントを出す。
TERM 呼び出し	TERM コマンド	PSB を終了する。
XRST 呼び出し	XRST コマンド	拡張再始動を出す。

注:

1. 読み取りコマンドは読み取り保持呼び出しに相当し、読み取りコマンドと読み取り呼び出しのパフォーマンスは同じです。
2. バッチ・コマンド・レベル・プログラムでは、GSCD 呼び出しを使用することができます。 GSCD は、プロダクト・センシティブ・プログラミング・インターフェースです。
3. STAT は、プロダクト・センシティブ・プログラミング・インターフェースです。

コマンド・コードとオプションの比較

次の表では、EXEC DLI コマンドで使用するオプションと、DL/I 呼び出しで使用するコマンド・コードとを比較しています。例えば、LOCKED オプションは、Q コマンド・コードと同じ機能を果たします。

表 90. 呼び出しレベル・プログラムとコマンド・レベル・プログラムの比較: コマンド・コードとオプション

呼び出しレベル	コマンド・レベル	用途 . . .
C	KEYS オプション	セグメントの連結キーを使用して、セグメントを識別する。
D	INTO または FROM。リトリーブまたは挿入されるセグメント・レベルに指定。	要求を 1 つだけ使用して、一連のセグメントを階層パスにおいてリトリーブまたは挿入する。セグメントごとに個別要求を使用する必要なし。(パス CALL またはコマンド。)
F	FIRST オプション	特定セグメント・オカレンスを検索するとき、親のもとにあるセグメントの最初のオカレンスまでバックアップする。ルート・セグメントに対しては無視される。
L	LAST オプション	親のもとにあるセグメントの最後のオカレンスをリトリーブする。
M	MOVENEXT オプション	現行セグメントの次のセグメントにサブセット・ポインターを設定する。
N	置き換え不要なセグメントに対しては、SEGMENT オプションは省略。	読み取り保持要求の後でセグメントを置換するときに、置き換え不要なセグメントを指定する。 通常、セグメントのパスを置き換えるときに使用する。
P	SETPARENT	通常 (要求の最低階層レベル) よりも高いレベルで親子関係を設定する。
Q	LOCKCLASS, LOCKED	処理終了まで、他のプログラムが更新できないようにセグメントを予約する。
R	GETFIRST オプション	サブセット内の最初のセグメントをリトリーブする。
S	SET オプション	無条件で、サブセット・ポインターを現行セグメントに設定する。

表 90. 呼び出しレベル・プログラムとコマンド・レベル・プログラムの比較: コマンド・コードとオプション (続き)

呼び出しレベル	コマンド・レベル	用途 . . .
U	コマンド・レベル・プログラムには、相当するものなし。	セグメントの検索対象を、位置が設定されているセグメント・オカレンスの従属セグメントだけに限定する。
V	CURRENT オプション	現在位置のレベル以上の階層レベルをこのセグメントの修飾として使用する。
W	SETCOND オプション	条件付きで、サブセット・ポインターを現行セグメントに設定する。
Z	SETZERO オプション	サブセット・ポインターをゼロに設定する。
–	コマンド・レベルにはない。	ヌル。 コマンド・コードを指定せずにコマンド・コード形式で SSA を使用する。 実行中に、必要なコマンド・コードに置換できる。

第 35 章 データ可用性の強化

プログラムは、DL/I 全機能データベースが使用不能であることを示す状況コードを受け取ると、失敗します。 これを回避するために、ここで説明する、データ可用性の強化機能を使用できます。 DBCTL で PSB をスケジュールした後で、アプリケーション・プログラムは IMS に対して、プログラムでデータ可用性状況コードを処理できることを示し、各データベースの可用性に関する情報を取得する要求を出すことができます。

データベース可用性状況コードの受け取り

これらの状況コードが出されるのは、PSB のスケジューリングが完了したが、参照されたデータベースのすべてを使用することはできなかったためです。 ACCEPT コマンドを使用して、プログラムを異常終了させるのではなく状況コードを戻すように DBCTL に指示します。

```
EXEC DLI ACCEPT STATUSGROUP('A');
```

データベース可用性に関する情報入手

データ可用性状況コードをそれぞれの DB PCB に入れる条件は次のとおりです。

- CICS DBCTL 環境では、PSB のスケジューリング要求コマンド SCHD を使用します。
- バッチ環境および BMP 環境では、初期設定時に行います。

データ可用性状況コードは、DL/I インターフェース・ブロック (DIB) から取得できます。それには次の QUERY コマンドを使用します。

```
EXEC DLI QUERY USING PCB(n);
```

n は PCB を指定します。

QUERY コマンドが発行されるのは PSB のスケジュールと最初のデータベース呼び出しの間です。プログラムが DB PCB を使用して既に呼び出しを出している場合は、QUERY コマンドを REFRESH コマンドの次に出します。

```
EXEC DLI REFRESH DBQUERY
```

REFRESH コマンドは DIB の情報を更新します。このコマンドは 1 回だけしか出すことができません。

全機能データベースの場合、DIBSTAT には、NA、NU、TH、またはブランクが入らなければなりません。 MSDB および DEDB の場合は、DIBSTAT には必ずブランクが入らなければなりません。

CICS コマンド言語変換プログラムが EXEC DLI コマンドの変換に使用された場合は、データ可用性状況に加えて、DBDNAME が DIB フィールド DIBDBDNM に戻されます。また、データベース編成名が DIB フィールド DIBDBORG に戻されます。

第 5 部 SQL のアプリケーション・プログラミング

これらのトピックは、構造化照会言語 (SQL) での IMS アプリケーション・プログラムの開発に関する詳細情報を示します。

また、IMS データをリトリーブおよび変更するための SQL 照会の使用についての詳細な情報も提供します。この情報は、SQL に精通し、IMS がサポートする 1 つ以上のプログラミング言語について知識がある IMS アプリケーション開発者を対象としています。

第 36 章 COBOL での SQL に関する考慮事項および制約事項

現行の実装環境では、COBOL での SQL サポートには、いくつかの考慮事項と制約事項があります。

- SQL キーワードのサブセットがサポートされます。現在、IMS Universal JDBC ドライバー ではサポートされているが、COBOL アプリケーションで使用された場合にはサポートされない SQL キーワードがあります。以下に例を示します。
  - XML は、COBOL SQL の SELECT ステートメントではサポートされません。
  - SQL COMMIT および ROLLBACK キーワードはサポートされません。データベースの変更をコミットまたはロールバックするには、IMS DB システム・サービス呼び出しを使用する必要があります。

    サポートされる SQL ステートメントおよびキーワードについては、SQL ステートメント (アプリケーション・プログラミング API)を参照してください。
- IMS TM/DB 環境では、BMP、IFP、および MPR で実行されている COBOL アプリケーションで SQL ステートメントがサポートされます。また、DBCTL 環境では BMP がサポートされます。バッチおよび DB バッチはサポートされません。また、IBM CICS Transaction Server for z/OS および Db2 for z/OS ストアード・プロシージャーから IMS への EXEC SQLIMS API の使用はサポートされません。EXEC SQLIMS API の代わりに DL/I API を使用する必要があります。
- Db2 for z/OS データには、同じ COBOL アプリケーション内で DB2 SQL サポートを使用してアクセスすることができます。IMS は SSM パラメーターを使用して DB2 に接続する必要があり、DFSLI000 モジュールが組み込まれている必要があります。
- SQL Support for COBOL を使用するには、IMS カタログを使用可能にする必要があります。必ず、COBOL SQL アプリケーションで必要になるデータベース・メタデータを IMS カタログにロードしてください。
- COBOL SQL アプリケーションで IMS 従属領域用に十分なストレージがあることを確認してください。COBOL SQL アプリケーションを実行するための IMS 従属領域サイズには、少なくとも 12MB を指定してください。ストレージを使い尽くした場合、878 やその他のストレージに関連した異常終了が発生します。
- アプリケーションで一度にアクティブにできるのは、1 つのカーソルと SQL ステートメントのみです。同じアプリケーションで複数の SQL ステートメントを実行する必要がある場合は、最初に前のステートメントのカーソルを閉じてから、新規のカーソルを開くか新規のステートメントを準備します。
- COBOL でサポートされているキーワードのセットは、データベース・アクセス呼び出しを行うためのものだけです。 IMS データベース・サービス、GSAM、IMS TM、およびメッセージ処理サービスについては、引き続き DL/I API を使用します。

- 動的 SQL ステートメントはサポートされます。静的 SQL は、現在はサポートされていません。
- COBOL CODEPAGE オプションの EBCDIC CCSID 37 および 1140 コード・ページのみがサポートされます。
- ゾーン 10 進数型およびパック 10 進数型の修飾子は、クエリー内で論理演算子 OR を使用してのみ、他の修飾子と結合することができます。
- WHERE 節内の述部では、データ型が整数の場合、メモリー消費量が大きくなるおそれがあるため、連続して使用できる AND ステートメントは 10 個までです。WHERE 節に 1 つ以上の OR 述部も含まれている場合は 10 個を超えるステートメントを使用できますが、連続した 10 個の AND ステートメントは使用できません。
- SQL ステートメント用に WHERE 節を作成し、修飾ステートメントで仮想外部キーを指定した場合、データベースへのアクセスがランダムのときは、以下の制限が適用されます。
  - 仮想外部キーは、ルート・セグメントの最初の子セグメント (レベル 2) のいずれかに対するものでなければなりません。階層内でそれより下位レベルにあるセグメントに対して仮想外部キーを指定することはできません。
  - 仮想外部キーを使用する複数の修飾ステートメントを結合するには、OR 演算子を使用する必要があります。
  - 仮想外部キーのみを指定できます。同じセグメントや別のセグメントから他の列を指定することはできません。

  例えば、HOSPITAL 表がルート・セグメントであり、HOSPCODE 列がユニーク・キーであるとします。WARD 表は HOSPITAL の最初の子セグメントで、HOSPITAL_HOSPCODE は HOSPITAL 表内の HOSPCODE 列を参照する仮想外部キーです。この場合、以下の SELECT ステートメントの WHERE 節は有効です。

  ```
  SELECT WARD.WARDNAME, WARD.PATCOUNT FROM PCB01.WARD
    WHERE HOSPITAL_HOSPCODE='ARS100100D' OR HOSPITAL_HOSPCODE='ARS100100D'
  ```

  以下の SELECT ステートメントの WHERE 節は失敗します。

  ```
  SELECT WARD.WARDNAME, WARD.PATCOUNT FROM PCB01.WARD
    WHERE HOSPITAL_HOSPCODE='ARS100100D' AND XINTEGER > '5'
  ```

第 37 章 SQL 用アプリケーション・プログラムの作成

IMS と対話するアプリケーションは、まず IMS に接続する必要があります。その後、データの読み取り、追加、または変更、あるいは IMS データベースの操作を行うことができます。

アプリケーション・プログラムでの SQL ステートメントのコーディング: 一般情報

IMS には、IMS データにアクセスするための SQL ステートメントを COBOL アプリケーション内で直接発行するための手段が用意されています。IMS では、COBOL 用の IMS SQL をプログラミングする際に、Db2 for z/OS と同じ SQL コーディング手法が使用されます。

アプリケーション・プログラムに SQL ステートメントを組み込むには、次の手順で行います。

1. IMS と通信するために、次のいずれかの方法を選択します。
   - 組み込み動的 SQL
   - JDBC アプリケーション・サポート

   アプリケーションを Java で作成する場合は、JDBC アプリケーション・サポートを使用して IMS にアクセスすることができます。

   COBOL のアプリケーションを作成する場合は、組み込み動的 SQLを使用します。
2. SQL ステートメントが正常に実行されたかを検査するのにプログラムが使用することができる SQL 連絡域 (SQLIMSCA) を定義します。
3. 少なくとも 1 つの SQL 記述子域 (SQLIMSDA) を定義します。
4. IMS と COBOL の間でデータを受け渡しするために、次のいずれかのデータ項目を宣言します。
   - ホスト変数
   - ホスト構造

   適切なデータ・タイプを使用していることを確認します。
5. SQL ステートメントをコーディングして、IMS データにアクセスする。

   SELECT ステートメントを使用して IMS データを照会する場合は、カーソルを使用して一連の行を選択して、一度に 1 行ずつ処理します。
6. SQL ステートメントの実行結果を検査します。
7. SQL エラー・コードを処理します。

関連概念:

644 ページの『動的 SQL』

744 ページの『IMS Universal JDBC ドライバーを使用したプログラミング』

関連タスク:

685 ページの『カーソルを使用した一連の行のリトリーブ』

SQL ステートメントが正常に実行されたかどうかの検査に使用できる項目の定義

プログラムに SQL ステートメントが組み込まれている場合、そのステートメントが正常に実行されたかどうかを検査できるように、プログラムに SQLIMSCODE、SQLIMSSTATE、および SQLIMSERRMT 変数が含まれる SQL 連絡域 (SQLIMSCA) を組み込むことをお勧めします。

関連タスク:

659 ページの『COBOL での SQL 連絡域の定義』

関連資料:

SQLIMSCA フィールドの説明 (アプリケーション・プログラミング API)

INCLUDE (アプリケーション・プログラミング API)

SQL 記述子域の定義

プログラムが特定の SQL ステートメントを含んでいる場合は、SQL 記述子域 (SQLIMSDA) を組み込む必要があります。使用されるコンテキストに応じて、SQLIMSDA には準備済み SQL ステートメントまたはホスト変数に関する情報が格納されます。この情報は、アプリケーション・プログラムからも IMS からも読み取ることができます。

プログラムに次のいずれかのステートメントが含まれる場合、プログラムに SQLIMSDA を組み込む必要があります。

- DESCRIBE ステートメント名 INTO 記述子名
- FETCH ... INTO DESCRIPTOR *descriptor-name*

関連タスク:

660 ページの『COBOL での SQL 記述子域の定義』

関連資料:

SQLIMSCA フィールドの説明 (アプリケーション・プログラミング API)

SQL 記述子域 (SQLIMSDA) (アプリケーション・プログラミング API)

ホスト変数および標識変数の宣言

プログラムの SQL ステートメント内でホスト変数を使用して、IMS とアプリケーション間でデータを渡すことができます。

ホスト変数、ホスト変数配列、およびホスト構造を宣言するには、次の手順で行います。

COBOL に適した方法を使用します。

ホスト変数

ホスト変数を使用して、IMS とアプリケーションとの間で単一データ項目を受け渡します。

ホスト変数 は、ホスト言語で宣言される単一のデータ項目で、SQL ステートメント内で使用されます。COBOL で作成されたアプリケーション・プログラムでホスト変数を使用して、以下のアクションを実行することができます。

- データをリトリーブしてホスト変数に入れ、アプリケーション・プログラムがそれを使用できるようにする。
- EXECUTE、PREPARE、および OPEN 呼び出し中に、パラメーター・マーカーを持つ動的 SQL ステートメントのホスト変数内のデータを使用する。

関連概念:

646 ページの『SQL ステートメント内のホスト変数の規則』

関連資料:

661 ページの『COBOL でのホスト変数』

ホスト構造

ホスト構造を使用して、IMS とアプリケーションとの間でホスト変数の集まりを受け渡します。

ホスト構造 とは、単一名によって参照できるホスト変数の集まりのことです。ホスト構造は、ホスト言語のステートメントを使用して定義します。ホスト構造内のホスト変数リストを参照する場合のどのコンテキスト内でも、ホスト構造を参照できます。ホスト構造参照は、構造宣言で定義された順序で、その構造内の各ホスト変数を参照するのと同じです。標識変数 (または標識構造) をホスト構造に使用することもできます。

関連資料:

665 ページの『COBOL でのホスト構造』

標識変数、配列、および構造

標識変数は、特定のホスト変数に関連しています。それぞれの標識変数は、関連したホスト変数に関する情報を示す短精度整数値を格納します。標識構造は、ホスト変数構造と同じ目的を果たします。

標識変数を使用して、次のアクションを実行できます。

- 関連する出力ホスト変数の値が NULL かどうかを判別する
- ホスト変数への割り当て時に切り捨てられた文字ストリングの元の長さを判別する

標識構造を使用して、ホスト・データ構造内の個々の項目に対して、これらと同じアクションを実行できます。

可変長セグメントの NULL 可能フィールドに標識変数を指定する場合、そのフィールドが NULL である場合は標識変数に負の値 (-1) が設定されます。プログラムは、フィールドを使用してそのフィールドが本当に NULL であるかを判別する前に、標識変数を検査する必要があります。標識変数に正整数が含まれる場合、リトリーブされた文字ストリング値は切り捨てられています。また、この整数はストリングの元の長さです。

標識構造 は、指定されたホスト構造をサポートするハーフワード整変数の配列です。プログラムによってリトリーブされ、ホスト構造に入れられたフィールドの値

が NULL になる可能性がある場合、標識構造名をホスト構造名に付加することができます。この名前によって、ホスト構造のホスト変数に NULL 値が戻されるたびに、IMS はそれをプログラムに知らせることができます。

アプリケーションでの SQL ステートメントの使用

動的 SQL を使用して、COBOL プログラム内で SQL ステートメントをコーディングすることができます。

関連概念:

669 ページの『COBOL プログラムでの SQL ステートメント』

動的 SQL

動的 SQL ステートメントは、プログラムの実行中に準備され、実行されます。アプリケーションが実行する必要がある SQL ステートメントが実行時まで分からない場合には、動的 SQL を使用します。

動的 *SQL* では、SQL ステートメントはプログラムの実行中にプログラム内で準備され、実行されます。動的 SQL には、以下の 2 つのタイプがあります。

- 対話式 SQL

  ユーザーは IMS Explorer for Development を通して SQL ステートメントを入力します。IMS は、それらのステートメントを動的 SQL ステートメントとして準備し実行します。
- 組み込み動的 SQL

  アプリケーションは、SQL ソースをホスト変数に置き、PREPARE および EXECUTE ステートメントを組み込み、それらのホスト変数の内容を実行時に準備し、実行するよう IMS に通知します。組み込み動的 SQL を含むプログラムは、プリコンパイルし、バインドする必要があります。

動的 SQL 処理

動的 SQL を使用しているプログラムは、SQL ステートメントを文字ストリング形式で入力として受け取り、あるいはそれを生成します。既知のタイプの既知の番号を知っている場合、プログラミングを単純化することができます。最も一般的なケースでは、実行される SQL ステートメントについては事前に何も分かっていないので、プログラムは通常、以下のようなステップをとります。

1. パラメーター・マーカーも含めた入力データを、SQL ステートメントに変換する。
2. 実行する SQL ステートメントを準備して、結果セグメントの記述を取得する。
3. リトリーブされたデータを入れておくための十分な主ストレージを、SELECT ステートメント用に確保する。
4. ステートメントを実行し、あるいはデータ行をフェッチする。
5. 戻された情報を処理する。
6. SQL 戻りコードを処理する。

固定リスト SELECT ステートメントでの SQL の動的実行

固定リスト SELECT ステートメントは、既知のタイプの既知の個数の値を含む行を戻します。このタイプのステートメントを使用する場合、ホスト変数のリストを指定して、ファイル化された値を組み込むことができます。

「固定リスト」という用語は、戻されるデータ行数を事前に知る必要があるということを意味するものではありません。しかし、フィールドの数とそのフィールドのデータ・タイプは知る必要があります。固定リスト SELECT ステートメントは、いくつの行でも含むことができる結果セグメントを戻します。プログラムでは、それらの行を 1 時点に 1 つずつ、FETCH ステートメントを使用して調べます。フェッチが続けて出されると、そのつど前のフェッチと同数の値が戻され、その値は、そのつど同じデータ・タイプになっています。

固定リスト SELECT ステートメントを動的に実行するには、プログラムは以下のことを行わなければなりません。

1. SQLIMSCA を組み込む。
2. 入力 SQL ステートメントをデータ域にロードする。 上記の 2 つのステップは、プログラム内の非 SELECT ステートメントに対する動的 SQL を含めて、まったく同じです。
3. ステートメント名に対するカーソルの宣言
4. ステートメントの準備
5. カーソルを開く。
6. 結果セグメントからの行のフェッチ
7. カーソルのクローズ
8. 以上の結果として発生したエラーがあれば、その処理を行う。 このステップは、結果として起こるエラーの数とタイプを除けば、静的 SQL の場合と同じです。

例: プログラムが、以下の形式で SELECT ステートメントを動的に実行して、病院名とコードをリトリーブすると仮定します。

```
SELECT HOSPNAME, HOSPCODE FROM PCB01.HOSPITAL
```

ステートメント名に対するカーソルの宣言:

結果を SELECT ステートメントのホスト変数に入れるには、カーソルを使用します。

例: カーソルを宣言するには、ステートメント名 (これを STMT とします) を使用し、カーソルにも名前を付けます (例えば、C1)。

```
EXEC SQLIMS DECLARE C1 CURSOR FOR STMT
END-EXEC.
```

ステートメントの準備:

ステートメント (STMT) を STMTSTR から準備します。

例: 以下に、PREPARE ステートメントの例を示します。

```
EXEC SQLIMS PREPARE STMT FROM :STMTSTR
END-EXEC.
```

STMT を実行するには、プログラムがカーソルをオープンし、結果セグメントから行をフェッチし、カーソルをクローズすることが必要です。

カーソルのオープン:

OPEN ステートメントは、STMT という名前の SELECT ステートメントを評価します。

例:

```
EXEC SQLIMS OPEN C1
END-EXEC.
```

結果表からの行のフェッチ:

例: 以下のようにすると、プログラムはあるステートメントを繰り返し実行することができます。

```
EXEC SQLIMS FETCH C1 INTO :HOSPNAME, :HOSPCODE
END-EXEC.
```

このステートメントの主要な特徴は、FETCH が返す値を受け取るのにホスト変数のリストを使用していることです。 このリストには、既知のデータ・タイプ (両方とも文字ストリングで、長さはそれぞれ 15 と 4) があり、その項目数 (この場合は :HOSPNAME および :HOSPCODE の 2 項目) も既知です。

このリストを FETCH ステートメントの中でのみ使用可能な理由は、このプログラムが固定リストの SELECT のみ使用するように計画したためです。カーソル C1 が指し示す行はいずれも、その行に見合う長さの 2 文字だけの値を含んでいなければなりません。 プログラムが上記以外の処理を行う場合は、可変リスト SELECT ステートメントの動的 SQL をプログラムに組み込む手法を使用する必要があります。

カーソルのクローズ:

例: プログラムが FETCH ステートメントの実行を完了したらカーソルを閉じます。

```
EXEC SQLIMS CLOSE C1
END-EXEC.
```

関連概念:

669 ページの『COBOL プログラムでの SQL ステートメント』

関連タスク:

652 ページの『非 SELECT ステートメントでの SQL の動的実行』
647 ページの『可変リスト SELECT ステートメントでの SQL の動的実行』

SQL ステートメント内のホスト変数の規則:

単一値を表現するには、組み込み SQL ステートメント内でホスト変数を使用します。ホスト変数は、リトリーブされたデータの保管や、割り当てられたり比較に使用されたりする値の受け渡しに便利です。

ホスト変数を使用するときは、以下の要件に従います。

- このホスト変数名を使用するには、ホスト・プログラムで宣言しておく必要があります。ホスト変数は、ホスト言語の命名規則に準拠します。
- ホスト変数を使用してデータ値を表すことはできます。ホスト変数を使用してセグメント、ビュー、またはフィールド名を表すことはできません。セグメントまたはフィールド名は、動的 SQLを使用して実行時に指定することができます。
- SQL ステートメントでホスト変数を使用する場合は、ホスト言語規則に従って宣言された任意の有効なホスト変数名を指定できます。
- SQL ステートメントで使用するホスト変数の前にコロン (:) を付けて、IMS が変数名とフィールド名を区別できるようにする必要があります。ホスト変数を SQL ステートメントの外側で使用するときは、ホスト変数の前にコロンを付けないでください。
- パフォーマンスを最適化するために、ホスト言語の宣言でデータベース内の関係しているデータに最も適合したデータ・タイプを採用するようにしてください。
- さまざまなデータ・タイプや長さの IMS フィールドとホスト変数との間の割り当ておよび比較では、変換が行われる可能性があります。

関連概念:

644 ページの『動的 SQL』

割り当てと比較 (アプリケーション・プログラミング API)

可変リスト SELECT ステートメントでの SQL の動的実行

可変リスト SELECT ステートメントは、個数が不明で、タイプが不明の値を含んでいる行を戻します。このタイプのステートメントを使用する場合、その結果を保管するために宣言する必要のあるホスト変数の種類は、前もって正確には分かりません。

可変リスト **SELECT** ステートメントの場合にアプリケーション・プログラムが行うべきこと**:** 可変リスト SELECT ステートメントを動的に実行するには、プログラムは以下のステップに従わなければなりません。

1. SQLIMSCA を組み込む。
2. 入力 SQL ステートメントをデータ域にロードする。
3. ステートメントを準備して実行する。 このステップは固定リスト SELECT の場合よりも複雑です。 これは、以下のステップで行われます。
   a. SQLIMSDA (SQL 記述子域) を組み込む。
   b. カーソルを宣言し、ステートメントを準備する。
   c. 結果セグメントの各フィールドに関する情報を入手する。
   d. リトリーブされたデータの行を入れておくために必要な主ストレージ量を判別する。
   e. リトリーブされたデータの各項目を保管する場所を示すストレージ・アドレスを SQLIMSDA に入れる。
   f. カーソルを開く。
   g. 行をフェッチする。
   h. 最終的にカーソルをクローズし、主ストレージを解放する。
4. エラーが発生したとき、その処理を行う。

可変リスト **SELECT** ステートメントの準備:

プログラムが SQL ステートメントを動的に実行すると仮定します。

プログラムはステートメントを可変長文字変数に入れ、その変数に STMTSTR という名前を付けます。プログラムは次に進んで、変数からステートメントを準備し、そのステートメントに名前を付けます。ここでは S1 と呼ぶことにします。

ステートメントが SELECT ステートメントである場合、プログラムは、各行に入っている値の数、およびそのデータ・タイプを判別する必要があります。その情報源は SQL 記述子域 (SQLIMSDA) です。

SQL 記述子域:

SQLIMSDA とは、プログラムとの連絡に使用される構造のことで、そのためのストレージは通常、実行時に動的に割り振られます。

COBOL の場合は、以下のように使用します。

```
EXEC SQLIMS INCLUDE SQLIMSDA END-EXEC.
```

SQL ステートメントに関する情報の入手:

SQLIMSDA には、発生する可変数の SQLIMSVAR を入れておくことができ、その各々は、SELECT ステートメントの結果セグメントの 1 つのフィールドを記述する 5 つの一連のフィールドで構成されています。組み込み SQLIMSDA には、SQLVAR のオカレンスが最大で 750 個含まれます。これは、最大 750 個の結果列を保持できることを意味します。

ステートメントに対するカーソルの宣言:

前述と同じように、動的 SELECT にはカーソルが必要です。例えば、以下のように書きます。

```
EXEC SQLIMS
  DECLARE C1 CURSOR FOR S1
```

最小の **SQLIMSDA** を使用するステートメントの準備:

STMTSTR に入っている文字ストリングからステートメントを準備し、その記述を SQLIMSDA に入れるには、以下のように書きます。

```
EXEC SQLIMS PREPARE STMT FROM :STMTSTR END-EXEC.
EXEC SQLIMS DESCRIBE STMT INTO :SQLIMSDA END-EXEC.
```

次の図に、使用されている最小 SQLIMSDA の内容を示します。

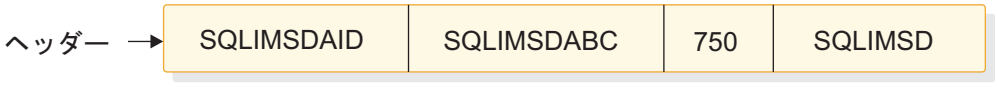


図 *98.* 最小の *SQLIMSDA* の構造

SQLIMSN による **SQLVAR** 組み込み内容の判別:

DESCRIBE (または PREPARE INTO) を使用する前に、SQLIMSN フィールドを設定しておく必要があります。このフィールドは、SQLIMSDA に割り振る SQLIMSVAR のオカレンスの数を示します。

- SQLIMSVAR の基本情報には、以下が含まれます。
  - データ・タイプ・コード
  - 長さ属性
  - 列名またはラベル
  - ホスト変数のアドレス
  - 標識変数のアドレス

DESCRIBE を実行するたびに、IMS は、以下の値を返します。この値を使用して、正しいサイズの SQLIMSDA を作成することができます。

- SQL ステートメントが SELECT ではない場合は、SQLIMSD は 0 です。 そうでない場合、SQLIMSD は結果セグメントのフィールドの数です。

ステートメントが **SELECT** ではない場合**:**

そのステートメントが SELECT であるかどうかを調べるために、プログラムは、DESCRIBE ステートメントの後に SQLIMSDA 内の SQLIMSD フィールドを照会することができます。このフィールドに 0 が入っている場合、ステートメントは SELECT ではなく、すでに準備済みで、かつプログラムはそのステートメントを実行できます。以下を使用することができます。

```
EXEC SQLIMS EXECUTE STMT END-EXEC.
```

ステートメントが **SELECT** の場合**:**

DESCRIBE ステートメントを実行した後、SQLIMSVAR の各オカレンスには、結果セグメントの 1 つのフィールドの記述が 5 つのフィールドに入っています。

次の表で、記述子域内の値を説明します。

表 *91. SQLIMSDA* に挿入される値

値	フィールド	説明
SQLIMSDA	SQLIMSDAID	『目印』
750	SQLIMSN	プログラムによって設定される、SQLIMSVAR が現れた個数
2	SQLIMSD	DESCRIBE ステートメントによって実際に使用される、SQLIMSVAR が現れた個数。
452	SQLIMSTYPE	最初に現れた SQLIMSVAR に入っている SQLIMSTYPE の値。 これは、最初のフィールドは固定長文字ストリングが入っていて、NULL を許容しないことを示しています。
3	SQLIMSLEN	列の長さ属性
未定義	SQLIMSIND	
8	SQLIMSNAME	フィールド名の文字数
HOSPCODE	SQLIMSNAME+2	最初の列のフィールド名

以下の図は、2 つのフィールドを記述した SQLIMSDA を示しています。

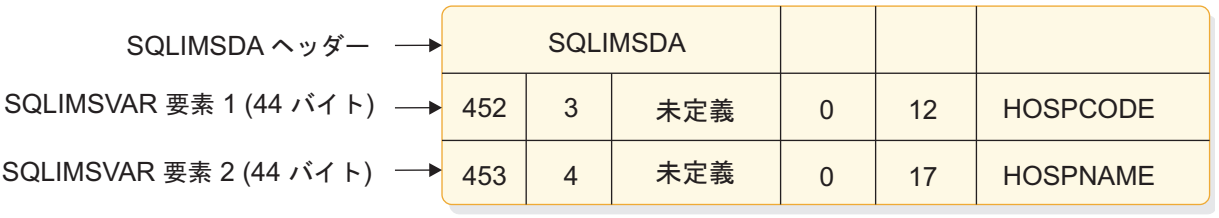


図 *99. DESCRIBE* 実行後の *SQLIMSDA* の内容

最初の SQLIMSVAR は、結果セグメントの最初のフィールド (HOSPCODE 列) に関係します。SQLIMSVAR エレメント 1 には、固定長文字ストリングが入り、NULL 値は使用できません (SQLIMSTYPE=452)。長さ属性は 3 です。

行を入れておくストレージの獲得**:**

結果セグメントの行をフェッチする前に、プログラムは以下を行わなければなりません。

1. 各 SQLIMSVAR 記述を分析して、フィールドの値にどれだけのスペースが必要であるかを判断する。
2. 必要なサイズをもつストレージ域のアドレスを導き出す。
3. そのアドレスを SQLIMSDATA フィールドに入れる。

値が NULL である可能性があることを SQLIMSTYPE フィールドが示しているときは、プログラムは、標識変数のアドレスも SQLIMSIND フィールドに入れなければなりません。次の図は、特定の処理の後の SQL 記述子域を示します。

ストレージ・アドレスの **SQLIMSDA** への設定**:**

各列の記述の分析を終えたら、プログラムは各 SQLIMSDATA フィールドの内容を、その列からの値を入れておくのに十分なサイズのストレージ域のアドレスで置き換えなければなりません。同様に、NULL を許容するすべてのフィールドについても、プログラムは SQLIMSIND フィールドの内容を置き換えなければなりません。内容は、その列の標識変数として使用できるハーフワードのアドレスで置き換える必要があります。プログラムがその目的のためにストレージ域を獲得できることはもちろんですが、使用するストレージがすべて連続している必要はありません。

次の図に、プログラムがフィールド値とその標識用のストレージを獲得し、そのアドレスを SQLIMSDA の SQLIMSDATA フィールドに入れた後の SQLIMSDA を示します。 プログラムが結果表の行を取得する前の記述子域の内容を示します。フィールドと標識変数のアドレスは、すでに SQLIMSVAR にあります。

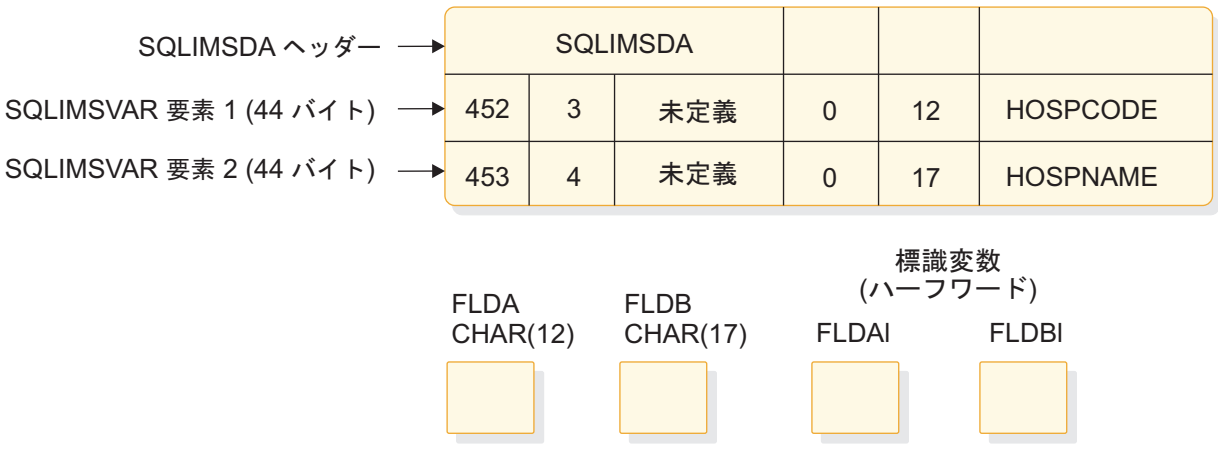


図 *100.* 記述の分析とストレージ獲得後の *SQL* 記述子域

可変リスト **SELECT** ステートメントの動的実行**:**

可変リスト SELECT ステートメントを使用して、結果表の行を簡単にリトリーブすることができます。ステートメントは、固定リストの例におけるステートメントとほとんど違いがありません。

カーソルのオープン**:** SELECT ステートメントがパラメーター・マーカーを含んでいないときは、このステップはきわめて簡単です。 以下に例を示します。

```
EXEC SQLIMS OPEN C1 END-EXEC.
```

結果表からの行のフェッチ**:** このステートメントは、固定リスト SELECT の場合の対応するステートメントとは異なります。 以下のように書きます。

```
EXEC SQLIMS
  FETCH C1 USING DESCRIPTOR :SQLIMSDA END-EXEC.
```

このステートメントの主要な特徴となっているのは、USING DESCRIPTOR :SQLIMSDA という文節です。この文節には、他の区域を指している SQLIMSVAR が置かれる SQL 記述子域の名前を指定します。 これらの他の区域には、FETCH から戻された値が入ります。 この文節を使用できる唯一の理由は、以前に SQLIMSDA を 650 ページの図 99 のようにセットアップしたためです。

次の図に、FETCH の結果を示します。結果表の単一の行からの値は、SQLIMSVAR フィールドに示されているデータ域に入れられます。

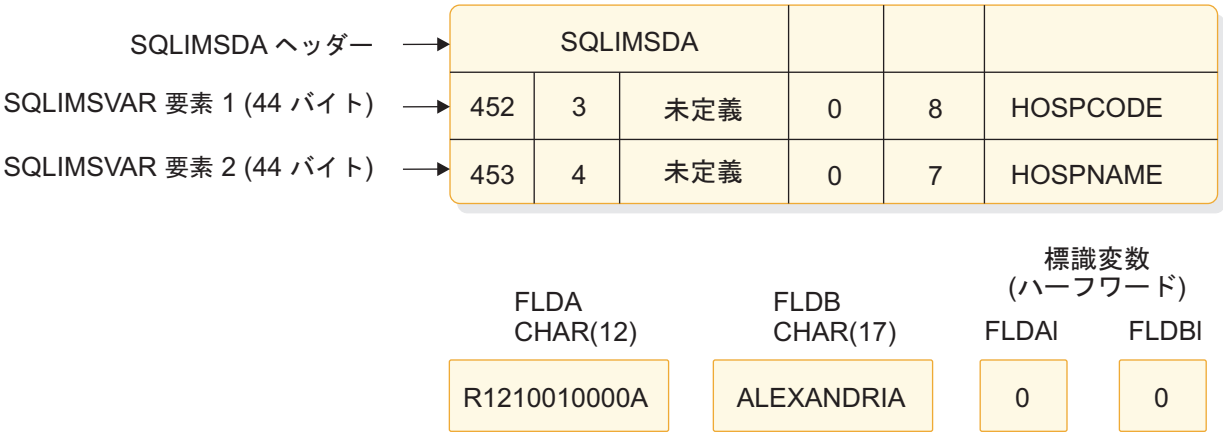


図 *101. FETCH* 実行後の *SQL* 記述子域

同じ FETCH ステートメントを続けて実行すると、結果表の連続する行からの値がこれらの同じデータ域に入れられます。

カーソルのクローズ: このステップは、固定リストの場合と同じです。 処理を必要とする行がなくなったら、次のステートメントを実行します。

```
EXEC SQLIMS CLOSE C1 END-EXEC.
```

関連概念:

669 ページの『COBOL プログラムでの SQL ステートメント』

関連資料:

DESCRIBE OUTPUT (アプリケーション・プログラミング API)

SQL 記述子域 (SQLIMSDA) (アプリケーション・プログラミング API)

SQLIMSTYPE および SQLIMSLEN (アプリケーション・プログラミング API)

SQLIMSDA ヘッダー (アプリケーション・プログラミング API)

非 SELECT ステートメントでの SQL の動的実行

動的 SQL を使用する最も簡単な方法は、INSERT、UPDATE、あるいは DELETE ステートメントなどの非 SELECT ステートメントを使用することです。

プログラムは、以下のステップをとらなければなりません。

1. SQLIMSCA を組み込む。
2. 入力 SQL ステートメントをデータ域にロードする。
3. ステートメントを実行する。
   - PREPARE と EXECUTE
4. エラーが発生したとき、その処理を行う。 最後に実行された SQL ステートメントからの戻りコードは、SQLIMSCA のホスト変数 SQLIMSCODE と SQLIMSSTATE、あるいはそれらに相当するフィールドに入って戻されます。
5. 変更している行数について SQLIMSERRD(3) をチェックします。

例:

ご使用のプログラムが、以下の形式の UPDATE ステートメントを動的に実行することで病院名を更新すると仮定します。

```
UPDATE HOSPITAL SET HOSPNAME = 'MISSION CREEK'
WHERE HOSPITAL.HOSPCODE = 'H001007'
```

この例では、UPDATE ステートメントは、ホスト変数 STMTSTR に保管されています。

ステートメントの宣言:

```
EXEC SQLIMS
  DELCARE STMT STATEMENT
END-EXEC.
```

可変長文字ホスト変数の宣言: SQL ステートメントを準備して実行する前に、そのステートメントをホスト変数に割り当てる必要があります。 SQL ステートメントに対して、可変長文字ホスト変数を宣言します。最初の 2 バイトには、SQL ステートメントの長さが含まれなければなりません。 SQL ステートメントの最大長は、32K です。以下に例を示します。

```
01  STMTSTR.
    49  STMTSTR-LEN  PIC S9(4) COMP VALUE +180.
    49  STMTSTR-TXT  PIC X(180) VALUE SPACES.
```

ステートメントの準備:

STMTSTR ホスト変数からステートメント (STMT) を準備します。

```
EXEC SQLIMS
PREPARE STMT FROM :STMTSTR
END-EXEC.
```

ステートメントの実行:

STMT を実行するために、ご使用のプログラムは EXECUTE 呼び出しを使用します。

```
EXEC SQLIMS
  EXECUTE STMT
END-EXEC.
```

関連概念:

669 ページの『COBOL プログラムでの SQL ステートメント』

関連タスク:

657 ページの『SQL ステートメントの実行結果の検査』

パラメーター・マーカーを使用した SELECT SQL ステートメントの動的実行

パラメーター・マーカーを指定して SELECT ステートメントを使用します。

病院番号のリストを使用して、SELECT ステートメントを繰り返し実行したいとします。さらに、ユーザーは、リトリーブする病院番号のリストを入力すると仮定します。ステートメント全体を動的に構成および実行する必要があります。プログラムでは以下のことを行うことができます。

- SQL ステートメントで定数値の代わりにパラメーター・マーカーを使用する

- USING 文節を指定して OPEN ステートメントを使用し、パラメーター・マーカーの値を設定する
- FETCH を使用してデータをリトリーブする

パラメーター・マーカーを使用するステートメント:

動的 SQL ステートメントはホスト変数を使用できません。 したがって、ホスト変数を含む SQL ステートメントを動的に実行することはできません。代わりに、パラメーター・マーカーを使用します。パラメーター・マーカーは、動的 SQL ステートメント内での位置を表す疑問符 (?) です。この場所に、アプリケーションが値を提供します。

パラメーター・マーカーの使用例:

```
SELECT HOSPNAME FROM PCB01.HOSPITAL WHERE HOSPCODE = ?;
```

準備済みステートメントを使用してデータを取り出す場合、ホスト変数 HOSPCODE をパラメーター・マーカーに関連付けます。

可変長文字ホスト変数の宣言: SQL ステートメントを準備して実行する前に、そのステートメントをホスト変数に割り当てる必要があります。 SQL ステートメントに対して、可変長文字ホスト変数を宣言します。最初の 2 バイトには、SQL ステートメントの長さが含まれなければなりません。 SQL ステートメントの最大長は、32K です。以下に例を示します。

```
01  STMTSTR.
    49  STMTSTR-LEN  PIC S9(4) COMP VALUE +180.
    49  STMTSTR-TXT  PIC X(180) VALUE SPACES.
```

CURSOR の宣言:

SELECT ステートメントの結果を記入するカーソルを宣言します。

カーソルを宣言するには、ステートメント名 (これを S1 とします) を使用し、カーソルにも名前を付けます (例えば、C1)。

```
EXEC SQLIMS DECLARE C1 CURSOR FOR S1
END-EXEC.
```

PREPARE ステートメントの使用:

文字ホスト変数 :STMTSTR に値 `SELECT HOSPNAME FROM PCB01.HOSPITAL WHERE HOSPCODE = ?` が入っていると仮定します。そのストリングから SQL ステートメントを準備して、それに S1 という名前を割り当てるには、次のように書きます。

```
EXEC SQLIMS PREPARE S1 FROM :STMTSTR;
```

準備済みステートメントにはまだパラメーター・マーカーが含まれているため、ステートメントの実行時にその値を指定しなければなりません。ステートメントの準備が完了したら、パラメーター・マーカーを使用することで、異なる病院コードの値を使用して何回でも同じステートメントを実行することができます。

STMT を実行するには、プログラムがカーソルをオープンし、結果セグメントから行をフェッチし、カーソルをクローズすることが必要です。

OPEN ステートメントの使用:

OPEN ステートメントは、準備済み SQL ステートメントのカーソルを開きます。SQL ステートメントにパラメーター・マーカーが含まれている場合は、OPEN の USING 文節を使用して、すべてのパラメーター・マーカーに値を指定する必要があります。OPEN ステートメントの USING 文節は、1 つ以上のホスト変数のリストまたはホスト構造を指定します。このリストに、すべてのパラメーター・マーカーの値が記載されています。C1 がカーソルで、パラメーター値がホスト変数 HOSPCODE に含まれていると仮定すると、次のように書き込みます。

```
OPEN C1 USING :HOSPCODE
```

OPEN ステートメントは、HOSPCODE に別の値を使用して実行することができます。

複数のパラメーター・マーカーの使用: 準備済みステートメント (例では S1) には、複数のパラメーター・マーカーを含めることができます。そのような場合は、EXECUTE の USING 文節で、変数のリストまたはホスト構造を指定します。これらの変数は、S1 に含まれるパラメーターの数とデータ・タイプに一致する値を、正しい順序で含んでいなければなりません。パラメーターの数とタイプは事前に把握し、プログラム内で変数を宣言する必要があります。

例えば、`OPEN C1 USING :PARM1, :PARM2`

結果表からの行のフェッチ:

この例では、ホスト変数にデータを取り出す方法を示しています。

```
EXEC SQLIMS FETCH C1 INTO :HOSPNAME, :HOSPCODE
END-EXEC.
```

カーソルのクローズ:

プログラムが FETCH ステートメントの実行を完了したらカーソルを閉じます。

```
EXEC SQLIMS CLOSE C1
END-EXEC.
```

関連概念:

669 ページの『COBOL プログラムでの SQL ステートメント』

関連資料:

PREPARE (アプリケーション・プログラミング API)

パラメーター・マーカーを使用した非 SELECT SQL ステートメントの動的実行

INSERT、UPDATE、および DELETE などの非 SELECT ステートメントに対して、パラメーター・マーカーを指定して PREPARE および EXECUTE を使用します。

病院番号のリストを使用して、UPDATE ステートメントを繰り返し実行したいとします。さらに、ユーザーは、更新する病院番号のリストを入力すると仮定します。ステートメント全体を動的に構成および実行する必要があります。次に、以下のように、プログラムは異なる方法で行わなければなりません。

- ホスト変数の代わりにパラメーター・マーカーを使用する。
- PREPARE および EXECUTE ステートメントを使用する。

PREPARE および EXECUTE でのパラメーター・マーカー: 動的 SQL ステートメントは、ホスト変数を使用することができません。したがって、ホスト変数を含む SQL ステートメントを動的に実行することはできません。代わりに、パラメーター・マーカーを使用します。パラメーター・マーカーは、動的 SQL ステートメント内での位置を表す疑問符 (?) です。この場所に、アプリケーションが値を提供します。

パラメーター・マーカーの使用例:

```
DELETE FROM PCB01.HOSPITAL WHERE HOSPCODE = ?;
```

準備済みステートメントを実行する場合、ホスト変数 HOSPCODE をパラメーター・マーカーに関連付けます。

可変長文字ホスト変数の宣言: SQL ステートメントを準備して実行する前に、そのステートメントをホスト変数に割り当てる必要があります。 SQL ステートメントに対して、可変長文字ホスト変数を宣言します。最初の 2 バイトには、SQL ステートメントの長さが含まれなければなりません。 SQL ステートメントの最大長は、32K です。以下に例を示します。

```
01  STMTSTR.
    49  STMTSTR-LEN  PIC S9(4) COMP VALUE +180.
    49  STMTSTR-TXT  PIC X(180) VALUE SPACES.
```

PREPARE ステートメントの使用: 文字ホスト変数 :STMTSTR に値 `DELETE FROM PCB01.HOSPITAL WHERE HOSPCODE = ?` が入っていると仮定します。そのストリングから SQL ステートメントを準備して、それに S1 という名前を割り当てるには、次のように書きます。

```
EXEC SQLIMS PREPARE S1 FROM :STMTSTR;
```

準備済みステートメントにはまだパラメーター・マーカーが含まれているため、ステートメントの実行時にその値を指定しなければなりません。ステートメントの準備が完了したら、パラメーター・マーカーを使用することで、異なる病院コードの値を使用して何回でも同じステートメントを実行することができます。

EXECUTE ステートメントの使用: EXECUTE ステートメントは、1 つ以上のホスト変数のリストまたはホスト構造を指定することで、準備済み SQL ステートメントを実行します。このリストに、すべてのパラメーター・マーカーの値が記載されています。S1 が準備済みステートメントで、パラメーター値がホスト変数 HOSPCODE に含まれていると仮定すると、次のように書き込みます。

```
EXECUTE S1 USING :HOSPCODE
```

EXECUTE ステートメントは、HOSPCODE に別の値を使用して実行することができます。

複数のパラメーター・マーカーの使用: 準備済みステートメント (例では S1) には、複数のパラメーター・マーカーを含めることができます。そのような場合は、EXECUTE の USING 文節で、変数のリストまたはホスト構造を指定します。これらの変数は、S1 に含まれるパラメーターの数とデータ・タイプに一致する値を、正しい順序で含んでいなければなりません。パラメーターの数とタイプは事前に把握し、プログラム内で変数を宣言する必要があります。

例:

STMT に 2 つのパラメーター・マーカーがある場合は、以下のステートメントが必要です。

```
EXEC SQLIMS
  EXECUTE STMT USING :PARM1, :PARM2
END-EXEC.
```

SQL ステートメントの実行結果の検査

SQL ステートメントの実行後、プログラムではデータをコミットする前にエラー・コードを検査し、これらのコードが示しているエラーを処理する必要があります。

SQL ステートメントの実行結果の検査は、次のいずれかの方法で行うことができます。

- SQLIMSCA の特定のフィールドを表示する方法。
- 特定の値の SQLIMSCODE または SQLIMSSTATE をテストする方法。
- ご使用のアプリケーション・プログラムの WHENEVER ステートメントを使用する方法。
- 標識変数をテストして数値エラーを検出する方法。

関連タスク:

659 ページの『COBOL での SQL 連絡域の定義』

SQLIMSCA を使用した SQL ステートメントの実行結果の検査

SQL ステートメントが正常に実行されたかどうか確認する方法の 1 つは、SQL 連絡域 (SQLIMSCA) を使用することです。この領域は、IMS との通信のために予約されています。

SQLIMSCA 使用時は、ご使用アプリケーション・プログラムの SQLIMSCA に含まれる情報を表示するのに必要な命令をコーディングしてください。

- IMS は、SQL ステートメントを処理する際に、ステートメント実行の成否を示す戻りコードを SQLIMSCODE および SQLIMSSTATE に入れます。
- IMS が FETCH ステートメントを処理し、FETCH が正常に行われると、SQLIMSCA の SQLIMSERRD(3) の内容が設定されて戻された行数になります。
- IMS が FETCH ステートメントを処理した場合、そのセグメントにある最後の行がその行のセットとともに戻されると、SQLIMSCODE の内容が設定されて +100 になります。
- IMS が UPDATE、INSERT、または DELETE ステートメントを処理し、ステートメントが正常に実行されると、SQLIMSCA の SQLIMSERRD(3) の内容が設定されて、更新、挿入、または削除が行われた行数になります。

関連タスク:

658 ページの『SQLIMSCODE および SQLIMSSTATE を使用した SQL ステートメントの実行結果の検査』

659 ページの『COBOL での SQL 連絡域の定義』

関連資料:

SQLIMSCA フィールドの説明 (アプリケーション・プログラミング API)

SQLIMSCODE および SQLIMSSTATE を使用した SQL ステートメントの実行結果の検査

どの SQL ステートメントも、それが実行されると、SQLIMSCA の SQLIMSCODE と SQLIMSSTATE フィールドに戻りコードが入ります。SQLIMSCA の SQLIMSERRMT フィールドには、エラーについて記述したメッセージ・テキストが入ります。

SQLIMSCODE:

IMS は、SQLIMSCODE として以下のコードを返します。
- SQLIMSCODE = 0 の場合、実行は成功です。
- SQLIMSCODE > 0 の場合、実行は成功ですが、警告が出されています。
- SQLIMSCODE < 0 の場合、実行は失敗です。

SQLIMSCODE が 100 の場合、データが見つからなかったことを示します。

SQLIMSSTATE: SQLIMSSTATE を使用すると、アプリケーション・プログラムは、異なる種類の IBM データベース管理システムに対して、同じ方法でエラーを検査できます。

関連タスク:

659 ページの『COBOL での SQL 連絡域の定義』

関連資料:

SQL コード (メッセージおよびコード)

WHENEVER ステートメントを使用した SQL ステートメントの実行結果の検査

WHENEVER ステートメントにより、IMS は SQLIMSCA を検査し、プログラム処理を続行します。エラー、例外、または警告が発生すると、IMS は、プログラム内の別区域に分岐させます。ご使用プログラムの条件処理区域は、ここで SQLIMSCODE または SQLIMSSTATE を検査して、エラーまたは例外に特別に対処できます。

WHENEVER ステートメントを使用すると、一般的条件が真のときに取るべき処置を指定することができます。複数の WHENEVER ステートメントをプログラムの中で指定することもできます。そうした場合は、最初に指定された WHENEVER ステートメントは、次の WHENEVER ステートメントが指定されるまでは、ソース・プログラム内の後続の SQL ステートメントのすべてに適用されます。

WHENEVER ステートメントは、以下のとおりです。

```
EXEC SQLIMS
  WHENEVER condition action
END-EXEC
```

WHENEVER ステートメントの条件 は、以下の 3 つの値のいずれかです。

SQLWARNING

SQLIMSWARN0 = W のとき、あるいは SQLIMSCODE が 100 以外の正の値のときのとるべき処置を示します。 IMS は、いくつかの理由から (例

えば、フィールド値が、ホスト変数に移されるときに切り捨てられる場合)、SQLIMSWARN0 を設定できます。プログラムはそれをエラーとは見なさないこともあります。

SQLERROR
SQL ステートメントの結果としてエラー・コード (SQLIMSCODE < 0) が IMS から戻されたときにとるべき処置を示します。

NOT FOUND
IMS が SQL ステートメントを満たす行を見つけることができないとき、あるいはフェッチすべき行が残っていないとき (SQLIMSCODE = 100)、取るべき処置を示します。

WHENEVER ステートメントもアクションは、以下の 2 つのいずれかです。

CONTINUE
ソース・プログラムの次の順次ステートメントを指定します。

GOTO または **GO TO** ホスト・ラベル
ホスト・ラベル によって識別されるステートメントを指定します。ホスト・ラベルには、1 つのトークンを代入します。その頭にはコロンを付けることもできます。トークンの形式はホスト言語によって異なります。例えば、COBOL では、セクション名 か、または非修飾の段落名 となります。

WHENEVER ステートメントは、その作用が及ぶ最初の SQL ステートメントより前に置かなければなりません。しかし、プログラムが SQLIMSCODE を直接検査する場合は、各 SQL ステートメントの後に SQLIMSCODE を検査する必要があります。

関連資料:

WHENEVER (アプリケーション・プログラミング API)

COBOL アプリケーション・プログラムでの SQL ステートメントのコーディング

COBOL アプリケーション・プログラムで SQL ステートメントをコーディングする場合、特定のガイドラインに従う必要があります。

COBOL での SQL 連絡域の定義

SQL ステートメントが含まれた COBOL プログラムには、SQL ステートメントが正常に実行されたかどうかを検査するための SQL 連絡域 (SQLIMSCA) を組み込むことができます。

SQL 連絡域を定義するには、次の手順で行います。

次の SQL INCLUDE ステートメントを使用して、標準の SQLIMSCA 宣言を要求します。

```
EXEC SQLIMS INCLUDE SQLIMSCA
```

INCLUDE SQLIMSCA または SQLIMSCODE の宣言は、WORKING-STORAGE SECTION で、77 レベルまたはレコード記述項目が指定できる個所ならばどこにで

も指定できます。
IMS は、各 SQL ステートメントの実行後に、SQLIMSCA 内の SQLIMSCODE、SQLIMSSTATE、および SQLIMSERRMT の値を設定します。アプリケーションは、これらの値を検査して、最後の SQL ステートメントが正常完了したかどうかを判別する必要があります。

関連タスク:

657 ページの『SQL ステートメントの実行結果の検査』

657 ページの『SQLIMSCA を使用した SQL ステートメントの実行結果の検査』

658 ページの『SQLIMSCODE および SQLIMSSTATE を使用した SQL ステートメントの実行結果の検査』

642 ページの『SQL ステートメントが正常に実行されたかどうかの検査に使用できる項目の定義』

COBOL での SQL 記述子域の定義

プログラムが特定の SQL ステートメント (DESCRIBE など) を含んでいる場合は、少なくとも 1 つの SQL 記述子域 (SQLIMSDA) を定義する必要があります。使用されるコンテキストに応じて、SQLIMSDA には準備済み SQL ステートメントまたはホスト変数に関する情報が格納されます。この情報は、アプリケーション・プログラムからも IMS からも読み取ることができます。

SQL 記述子域を定義するには、次の手順で行います。

1. 次の SQL INCLUDE ステートメントを使用して、標準の SQLIMSDA 宣言を要求します。

   ```
   EXEC SQLIMS INCLUDE SQLIMSDA
   ```

2. SQLIMSDA 宣言をプログラムの WORKING-STORAGE SECTION、LINKAGE SECTION、または LOCAL-STORAGE SECTION に置く必要があり、そのセクション内でレコード記述項目を指定できる任意の個所に宣言を配置できます。SQLIMSDA 宣言は、データ記述子を参照する最初の SQL ステートメントの前に置かなければなりません。

関連タスク:

642 ページの『SQL 記述子域の定義』

COBOL でのホスト変数および標識変数の宣言

プログラムの SQL ステートメント内でホスト変数およびホスト構造を使用して、IMS とアプリケーションの間でデータを受け渡すことができます。

ホスト変数およびホスト構造を宣言するには、次のようにします。

1. 以下の規則とガイドラインに従って、変数を宣言します。
   - SQL ステートメントで使用されるすべてのホスト変数は、プログラムの DATA DIVISION の WORKING-STORAGE SECTION または LINKAGE SECTION で、明示的に宣言する必要があります。
   - 各ホスト変数は、SQL ステートメントで使用する前に、明示的に宣言しておく必要があります。
   - デフォルトのタイプ、または IMPLICIT ステートメントを使用して、ホスト変数を暗黙的に宣言することはできません。

- ホスト変数を使用する SQL ステートメントが、その変数を宣言したステートメントの有効範囲内にあることを確認してください。

2. オプション: 関連する標識変数、配列、および構造を定義します。

関連タスク:

642 ページの『ホスト変数および標識変数の宣言』

COBOL でのホスト変数

COBOL プログラムでは、数値および文字のホスト変数を指定することができます。

制約事項:

- 一部の有効な COBOL 宣言だけが、ホスト変数宣言として認識されます。 ある変数の宣言が無効な場合に、その変数を参照する SQL ステートメントがあると、メッセージ UNDECLARED HOST VARIABLE が出される場合があります。
- 1 つ以上の REDEFINES 項目の後に、任意のレベル 77 データ記述記入項目を続けることができます。しかし、これらの項目の名前を SQL ステートメントの中で使用することはできません。 FILLER という名前の項目は無視されます。

推奨事項:

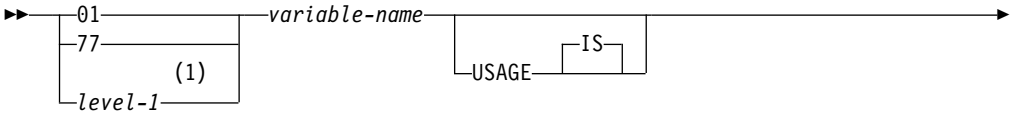
- オーバーフローには注意してください。例えば、INTEGER フィールド値をリトリーブして、PICTURE S9(4) ホスト変数に入れる場合に、そのフィールド値が 32767 より大きいか、あるいは -32768 より小さいと、オーバーフローになります。標識変数を指定するかどうかによって、オーバーフローの警告またはエラーを受け取ります。
- 切り捨てには注意してください。 例えば、80 文字の CHAR フィールド値をリトリーブして、PICTURE X(70) ホスト変数に入れると、リトリーブされたストリングの右端 10 文字が切り捨てられます。 倍精度浮動小数点または 10 進フィールドをリトリーブして、PIC S9(8) COMP ホスト変数に入れると、小数部の値が除去されます。同様に、DECIMAL データ・タイプのフィールド値をリトリーブして、精度の低い COBOL 10 進変数に入れると、値が切り捨てられる可能性があります。

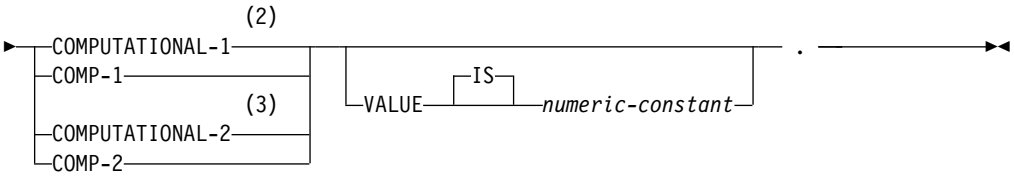
数値ホスト変数

次の形式の数値ホスト変数を指定できます。

- 浮動小数点数
- 整数および短整数
- 10 進数

次の図は、浮動小数点変数または実ホスト変数の宣言の構文です。





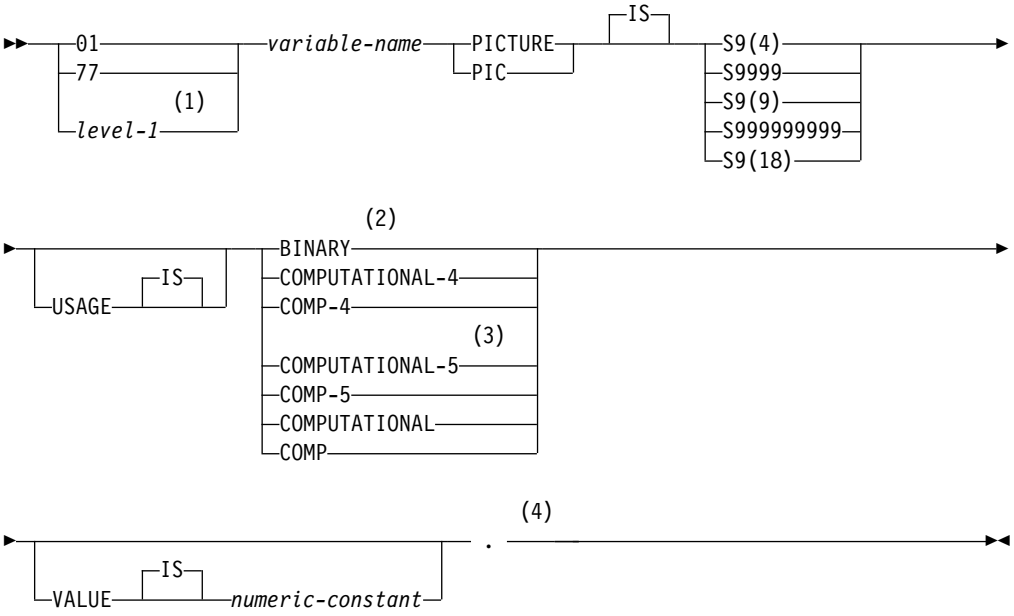
注:

1 *level-1* は 2 から 48 までの COBOL レベルを示します。

2 COMPUTATIONAL-1 と COMP-1 は同等です。

3 COMPUTATIONAL-2 と COMP-2 は同等です。

次の図は、整数ホスト変数、短精度整数ホスト変数、および長精度整数ホスト変数の宣言の構文です。



注:

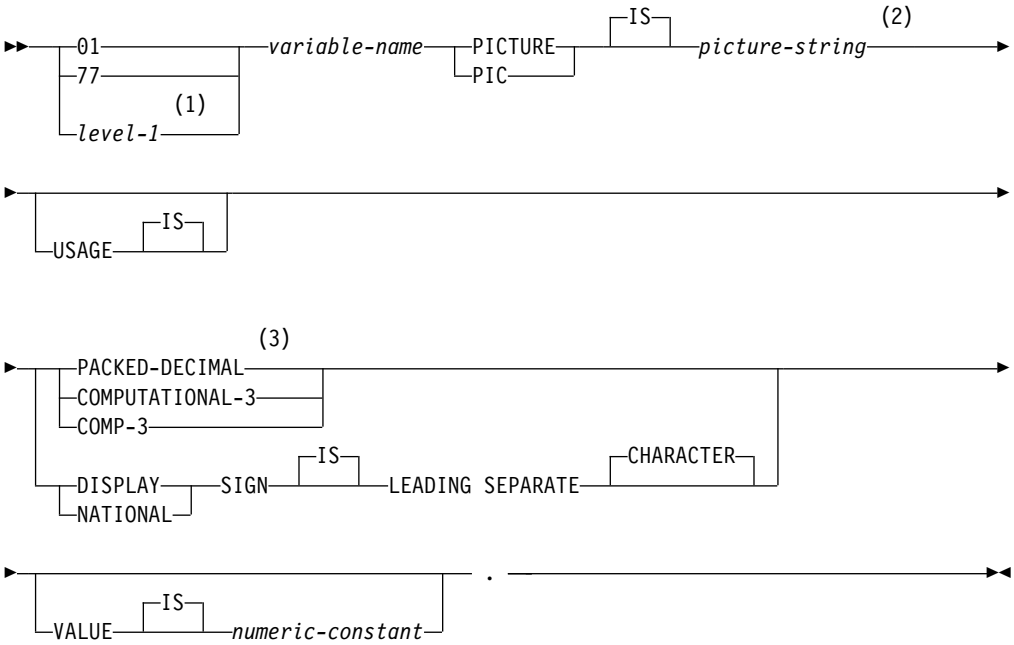
1 *level-1* は 2 から 48 までの COBOL レベルを示します。

2 COBOL 2 進整数データ・タイプの BINARY、COMPUTATIONAL、COMP、 COMPUTATIONAL-4、および COMP-4 は同等です。

3 COMPUTATIONAL-5 (および COMP-5) は、ほかのデータ・タイプを TRUNC(BIN) でコンパイルする場合は、ほかの COBOL 2 進整数データ・タイプと同等です。

4 位取りの指定は無視されます。

次の図は、10 進ホスト変数の宣言の構文です。



注:

1 *level-1* は 2 から 48 までの COBOL レベルを示します。

2 SIGN LEADING SEPARATE に関連付けられる *picture-string* の形式は、S9(*i*)V9(*d*) (つまり、9 が *i* 回と *d* 回現れる S9...9V9...9 か、9 が *i* 回現れる S9...9V) でなければなりません。

3 PACKED-DECIMAL、COMPUTATIONAL-3、および COMP-3 は同等です。これらのタイプに関連付けられる上図の *picture-string* の形式は、S9(*i*)V9(*d*) (つまり 9 が *i* 回および *d* 回現れる S9...9V9...9) か、S9(*i*)V でなければなりません。

COBOL では、SMALLINT および INTEGER データ・タイプを、10 進数の桁数として宣言します。IMS は、整数の全体サイズを使用して (TRUNC(BIN) コンパイラー・オプションでの処理に類似した方法で)、COBOL 宣言で指定された桁数で許容される場合を上回る値をホスト変数に入れることができます。TRUNC(OPT) または TRUNC(STD) でコンパイルする場合は、アプリケーションにある数字のサイズが、宣言された桁数の範囲内にあることを確認してください。

9999 を超える可能性のある短精度整数の場合は、S9(4) COMP-5 を使用するか、TRUNC(BIN) でコンパイルしてください。999 999 999 を超える可能性のある大きい整数の場合は、10 進データ・タイプを使用可能にするために S9(10) COMP-3 を指定してください。COBOL の PICTURE を超える整数用に COBOL を使用する場合には、必ず、フィールドを 10 進と指定して、データ・タイプが一致し、正しく機能するようにしてください。

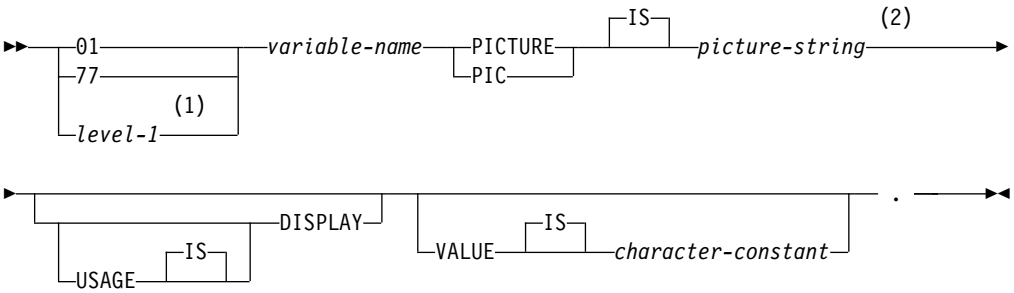
18 桁を超える 10 進数をサポートしない COBOL コンパイラーを使用する場合は、以下のいずれかのデータ・タイプを使用して、18 桁を超える値を収容します。

- 実際のデータ値が収まる場合での、精度が 18 より小か等しい桁数の 10 進変数。 10 進値をリトリーブして、データベース内のソース・フィールドより少ない位取りを持つ 10 進変数に入れると、その値の小数部分が切り捨てられることがあります。
- 整数または浮動小数点変数であって、値が変換されるもの。整変数を使用すると、その小数部分は失われます。10 進数が整数の最大値を超える可能性がある場合、あるいは小数部の値を保持する必要がある場合には、浮動小数点変数を使用します。浮動小数点数は、実数の近似値です。したがって、10 進数を浮動小数点変数に割り当てると、結果が元の数値と異なる場合があります。

文字ホスト変数

次の形式の文字ホスト変数を指定できます。
- 固定長ストリング

次の図は、固定長文字ホスト変数の宣言の構文です。

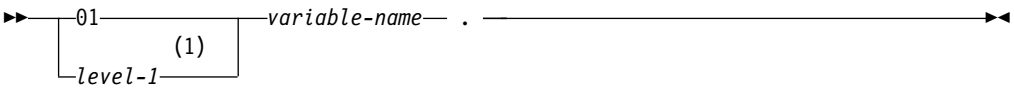


注:

1 *level-1* は 2 から 48 までの COBOL レベルを示します。

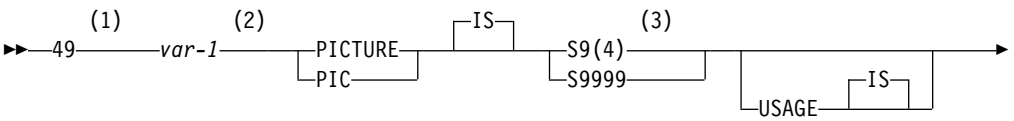
2 これらの形式に関連付けられる *picture-string* は、固定長ストリングの場合は、X(*m*) (つまり、X が *m* 回現れる XX...X) でなければなりません。ここで、*m* は COBOL の制限までです。

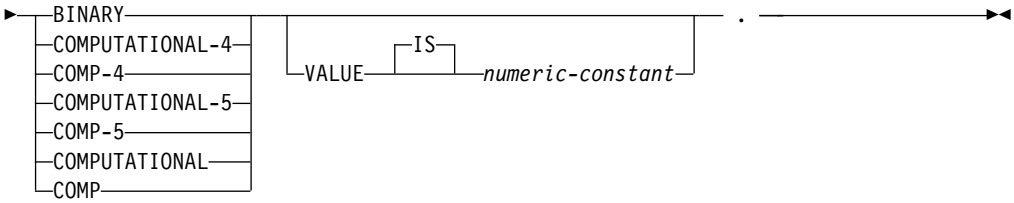
次の図は、可変長文字ホスト変数の宣言の構文です。



注:

1 *level-1* は 2 から 48 までの COBOL レベルを示します。



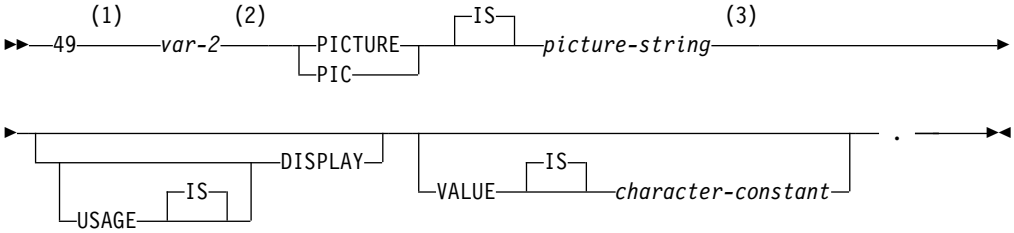


注:

1 レベル 49 に REDEFINE を挿入して使用することはできません。

2 *var-1* をホスト変数として直接参照することはできません。

3 TRUNC(STD) を指定した場合に COBOL が認識する値は最大でも 9999 ですが、IMS は S9(4) BINARY 変数の全長を使用します。この性質のため、COBOL ステートメントの実行時にはデータ切り捨てエラーが起こる可能性があり、可変長文字ストリングの最大長が事実上 9999 に制限されることになります。TRUNC(BIN) コンパイラー・オプションまたは USAGE COMP-5 を使用して、データ切り捨てを避けることができます。



注:

1 レベル 49 に REDEFINE を挿入して使用することはできません。

2 *var-2* をホスト変数として直接参照することはできません。

3 固定長ストリングの場合、*picture-string* は、X(*m*) (つまり、X が *m* 回現れる XX) でなければなりません。ここで、*m* は COBOL の制限までです。

関連概念:

642 ページの『ホスト変数』

関連タスク:

644 ページの『アプリケーションでの SQL ステートメントの使用』

COBOL でのホスト構造

COBOL ホスト構造は、プログラムの WORKING-STORAGE SECTION または LINKAGE SECTION に定義されている名前付きのホスト変数のセットです。

要件: COBOL でのホスト構造宣言は、以下の要件を満たす必要があります。

- COBOL ホスト構造は、そのホスト構造が複数レベル構造内に現れる場合でも、持つことができるのは最大 2 レベルです。
- ホスト構造名をグループ名にし、そのグループ名の従属レベルで基本データ項目の名前を付けるようにすることができます。

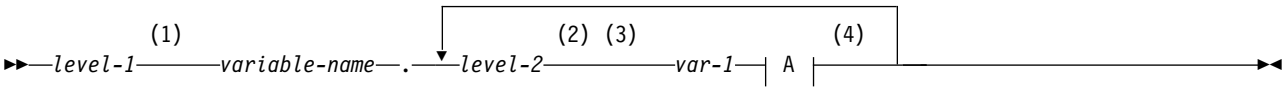
- IMS コプロセッサーを使用する場合、以下の項目のいずれかの後に従属レベルのホスト変数またはホスト構造を宣言しないでください。
  - 区域 A で始まる COBOL 項目
  - いずれかの SQL ステートメント (SQL INCLUDE を除く)
  - 組み込みメンバー内の SQL ステートメント

  IMS プリコンパイラーは、ホスト構造で上記の項目のいずれかを検出すると、構造が完了したものと見なします。

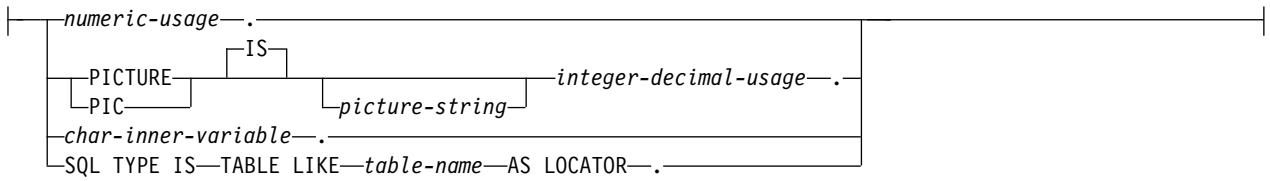
修飾されたホスト変数名を含む SQL ステートメントを作成するとき (例えば、構造内のフィールドを識別するためなど) は、その構造名と、その後ろにピリオドとフィールド名を付けて指定します。例えば、フィールド C1 を含む構造の場合、`C1 OF B` や `C1 IN B` ではなく `B.C1` と指定します。

ホスト構造

次の図は、ホスト構造の宣言の構文です。



A:



注**:**

1 *level-1* は 1 から 47 までの COBOL レベルを示します。

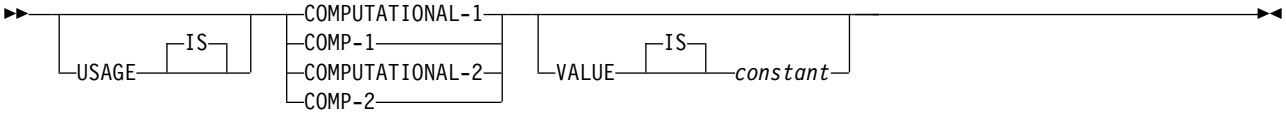
2 *level-2* は 2 から 48 までの COBOL レベルを示します。

3 構造体内のエレメントの場合、最大 2 つのレベルまでの、02 から 48 までの任意のレベルを使用します (01 または 77 以外)。

4 ホスト構造体宣言内で FILLER または任意指定の FILLER 項目を使用すると、構造体全体を無効にする可能性があります。

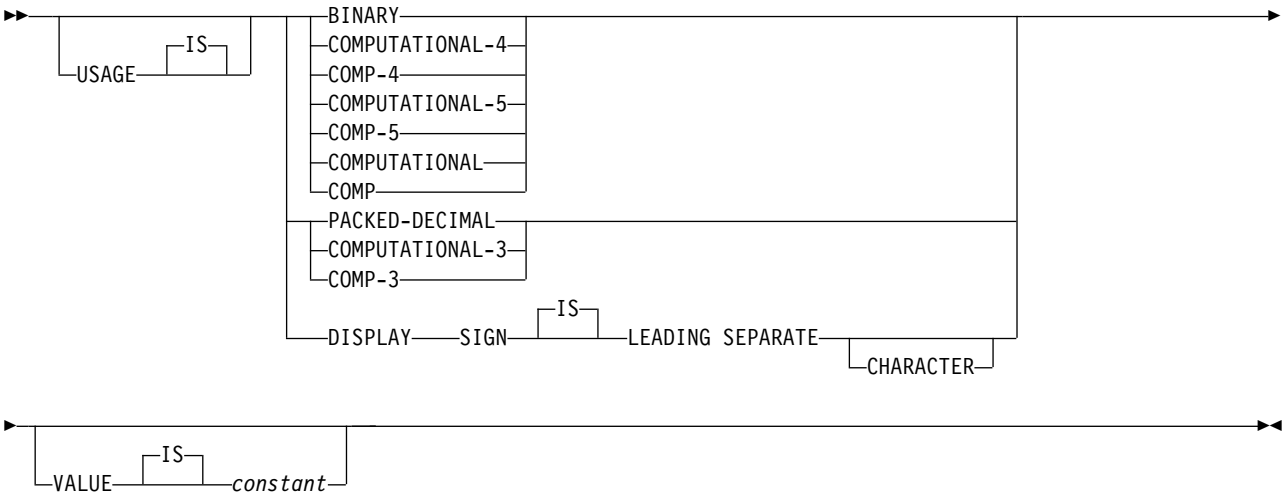
数値使用法項目

次の図は、ホスト構造の宣言内で使用される数値使用法項目の構文です。



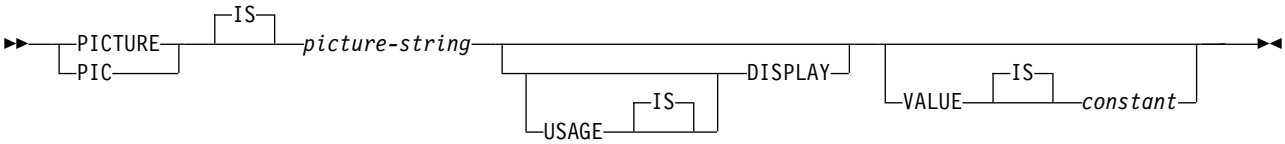
整数および 10 進数使用法項目

次の図は、ホスト構造の宣言内で使用される整数および 10 進数使用法項目の構文です。



CHAR 内部変数

次の図は、ホスト構造の宣言内で使用される CHAR 内部変数の構文です。



関連概念:

643 ページの『ホスト構造』

同等の SQL および COBOL のデータ・タイプ

COBOL プログラム内でホスト変数を宣言する際に、プリコンパイラーは同等な SQL データ・タイプを使用します。特定の SQL データ・タイプをリトリーブしてホスト変数に入れる場合、そのホスト変数のデータ・タイプが同等であることを確認する必要があります。

次の表には、SQL データ・タイプ、および基本 SQLIMSTYPE と SQLIMSLEN の値が記載されており、これらの値をプリコンパイラーは SQL ステートメント内のホスト変数に使用します。

表 92. プリコンパイラーが *COBOL* プログラム内のホスト変数に対して使用する *SQL* データ・タイプ、*SQLIMSLEN* 値、および *SQLIMSTYPE* 値

COBOL ホスト変数データ・タイプ	ホスト変数の **SQLIMSTYPE**[1]	ホスト変数の **SQLIMSLEN**	**SQL** データ・タイプ
COMP-2	480	8	DOUBLE PRECISION、または FLOAT(*n*) 22<=*n*<=53

表 92. プリコンパイラーが *COBOL* プログラム内のホスト変数に対して使用する *SQL* データ・タイプ、*SQLIMSLEN* 値、および *SQLIMSTYPE* 値 (続き)

COBOL ホスト変数データ・タイプ	ホスト変数の **SQLIMSTYPE**[1]	ホスト変数の **SQLIMSLEN**	**SQL** データ・タイプ
S9(*i*)V9(*d*) COMP-3 または S9(*i*)V9(*d*) PACKED-DECIMAL	484	バイト 1 は *i+d*、バイト 2 は *d*	DECIMAL(*i+d,d*) または NUMERIC(*i+d,d*)
S9(*i*)V9(*d*) DISPLAY SIGN LEADING SEPARATE	504	バイト 1 は *i+d*、バイト 2 は *d*	同等なものはありません。DECIMAL(*i+d,d*) または NUMERIC(*i+d,d*) を使用してください
S9(4) COMP-4、S9(4) COMP-5、S9(4) COMP、または S9(4) BINARY	500	2	SMALLINT
S9(9) COMP-4、S9(9) COMP-5、S9(9) COMP、または S9(9) BINARY	496	4	INTEGER
S9(18) COMP-4、S9(18) COMP-5、S9(18) COMP、または S9(18) BINARY	492	8	BIGINT
固定長文字データ	452	*n*	CHAR(*n*)
SQL TYPE is BINARY(*n*), 1<=*n*<=255	912	*n*	BINARY(*n*)

注:

1. ホスト変数が標識変数を含んでいる場合、SQLIMSTYPE 値は基本 SQLIMSTYPE 値に 1 を加えたものになります。

次の表は、それぞれの SQL データ・タイプごとに同等な COBOL ホスト変数を示しています。この表を使用して、データベースからの出力を受け取るために定義するホスト変数の COBOL データ・タイプを判別してください。例えば、TIMESTAMP データをリトリーブする場合は、長さ *n* の固定長文字ストリング変数を定義できます。

この表は、SQL データ・タイプと COBOL データ・タイプ間での直接の変換を示しています。ただし、多数の SQL データ・タイプが互換可能です。互換データ・タイプを持つデータを割り当てたり、比較したりする場合は、IMS がそれらの互換データ・タイプ間の変換を行います。

表 93. 特定の *SQL* データ・タイプのデータをリトリーブする際に使用できる、同等な *COBOL* ホスト変数

SQL データ・タイプ	同等な **COBOL** ホスト変数	注

SMALLINT	`S9(4) COMP-4,`
	`S9(4) COMP-5,`
	`S9(4) COMP,`
	`or` `S9(4) BINARY`	

INTEGER	`S9(9) COMP-4,`
	`S9(9) COMP-5,`
	`S9(9) COMP,`
	`or` `S9(9) BINARY`	

表 93. 特定の SQL データ・タイプのデータをリトリーブする際に使用できる、同等な COBOL ホスト変数 (続き)

SQL データ・タイプ	同等な **COBOL** ホスト変数	注

DECIMAL(*p*,*s*)	`S9(p-s)V9(s) COMP-3` または
	`S9(p-s)V9(s)`
	`PACKED-DECIMAL`
	`DISPLAY SIGN`
	`LEADING SEPARATE`
		p は精度、*s* は位取り。0<=*s*<=*p*<=31。*s*=0 なら、S9(*p*)V または S9(*p*) を使用してください。s=p なら、SV9(*s*) を使用してください。COBOL コンパイラーが 31 桁の 10 進数をサポートしない場合は、完全に同等なものはありません。COMP-2 を使用してください。

DOUBLE PRECISION、DOUBLE または FLOAT (*n*)	`COMP-2`	22<=*n*<=53
BIGINT	`S9(18) COMP-4`, `S9(18) COMP-5`, `S9(18) COMP`, **or** `S9(18) BINARY`	
CHAR(*n*)	固定長文字ストリング。 例: `01 VAR-NAME PIC X(n).`	1<=*n*<=255
BINARY(*n*)	SQL TYPE IS BINARY(*n*)	1<=*n*<=255
日付	長さ *n* の固定長文字ストリング。例えば、次のようになります。 `01 VAR-NAME PIC X(n).`	
TIME	長さ *n* の固定長文字ストリング。例えば、次のようになります。 `01 VAR-NAME PIC X(n).`	
TIMESTAMP	長さ *n* の固定長文字ストリング。例えば、次のようになります。 `01 VAR-NAME PIC X(n).`	*n* は少なくとも 19 でなければなりません。マイクロ秒まで含めるには、*n* は 26 でなければなりません。 *n* が 26 未満の場合は、マイクロ秒部分の切り捨てが行われます。

COBOL プログラムでの SQL ステートメント

特定の COBOL プログラム・セクションに SQL ステートメントをコーディングできます。

コーディングできるセクションを次の表に示します。

表 94. COBOL プログラム・セクションに入れることができる SQL ステートメント

SQL ステートメント	プログラム・セクション
INCLUDE SQLIMSCA	WORKING-STORAGE SECTION[1] または LINKAGE SECTION
INCLUDE テキスト・ファイル名	PROCEDURE DIVISION または DATA DIVISION[2]
DECLARE CURSOR	DATA DIVISION または PROCEDURE DIVISION
その他	PROCEDURE DIVISION

注:

1. IMS コプロセッサーを使用する場合は、表に WORKING-STORAGE SECTION がリストされている個所で、LOCAL-STORAGE SECTION を使用できます。
2. ホスト変数宣言を組み込むときは、INCLUDE ステートメントが WORKING-STORAGE SECTION または LINKAGE SECTION 内になければなりません。

COBOL プログラムの DECLARATIVES セクションに、SQL ステートメントを入れることはできません。

COBOL プログラム内の各 SQL ステートメントは、EXEC SQLIMS で始まり、END-EXEC で終わらなければなりません。EXEC および SQLIMS キーワードは、別の行にあっても構いません。 2 つのキーワード EXEC と SQL の間には、COBOL コメントを除いて、トークンを含めないでください (デバッグ行を含む)。キーワード EXEC と SQLIMS の間には、SQL コメントを含めないでください。

SQL ステートメントが 2 つの COBOL ステートメントの間にある場合、END-EXEC の後に付けるピリオドはオプションであり、適切でない場合があります。この SQL ステートメントが、COBOL ステートメントの IF...THEN セット内に現れた場合は、間違って IF ステートメントを終了しないように、最後のピリオドを省いてください。

コメント*:* COBOL のコメント行 (フィールド 7 の *) は、ブランクが使用できる個所ならば、SQL ステートメント内のどこにでも入れることができます。キーワード EXEC と SQLIMS の間に COBOL コメント行を含めることはできません。IMS コプロセッサーは、COBOL 規則に基づいてデバッグ行を扱います。この規則は、WITH DEBUGGING モードの設定によって異なります。

SQL INCLUDE ステートメントの場合、IMS コプロセッサーは、このテキストを COBOL プログラム構文の一部として扱います。

さらに、どの組み込み SQL ステートメントにも、SQL コメント ('--') を組み込めます。

デバッグ行*:* IMS コプロセッサーは、デバッグ行については COBOL 言語の規則に従います。

SQL ステートメントの継続*:* COBOL プログラムに組み込まれた SQL ステートメント内で文字ストリング定数をある行から次の行に継続するときの規則は、COBOL 内で非数値リテラルを継続するときの規則と同じになります。しかし、継続行の B 領域の最初の非ブランク文字には、引用符とアポストロフィのいずれでも使用できます。区切り ID の継続にも同じ規則が適用され、ストリング区切りオプションに左右されません。

SQL 標準に準拠するには、文字ストリング定数はアポストロフィで区切り、文字ストリング定数の継続行の B 領域で最初の非ブランク文字として引用符を使用します。

IMS コプロセッサーを使用する場合、SQL ステートメントの継続行は、フィールド 12 から 72 まで可能です。

セグメントの宣言*:* COBOL プログラムには、そのプログラムがアクセスする各セグメントおよびビューを記述するステートメント DECLARE TABLE を組み込む必要があります。DECLARE TABLE ステートメントは、IMS 宣言生成プログラムを使用して生成することができます。生成されたメンバーは、DATA DIVISION に組み込む必要があります。

COBOL プログラムでの動的 ***SQL:*** 通常、COBOL プログラムでは動的 SQL ステートメントの処理が容易です。COBOL プログラムが SELECT ステートメントを処理できるのは、戻されるデータ・タイプおよびフィールド数が決まっている場合です。変数リスト SELECT ステートメントを使用したい場合は、SQLIMSDA を使用する必要があります。

コードの組み込み*:* 区分データ・セットのメンバーから SQL ステートメントまたは COBOL ホスト変数宣言を組み込むには、ソース・コードのそれらのステートメントを入れたい個所に、以下の SQL ステートメントを入れてください。

```
EXEC SQLIMS INCLUDE member-name END-EXEC.
```

'EXEC SQLIMS' と 'END-EXEC' のキーワード対は、SQL ステートメントのみを組み込む場合に使用します。COPY や REPLACE などの COBOL ステートメントは、許可されていません。

マージン*:* EXEC SQLIMS から始まる SQL ステートメントは、フィールド 12 から 72 にコーディングする必要があります。

名前*:* ホスト変数には、COBOL で有効な名前であれば、どの名前でも使用できます。「DFS」または「DQF」から始まる入り口名は使用しないでください。また、「SQL」または「SQLIMS」から始まるホスト変数名は使用しないでください。これらの名前は、IMS 用に予約済みです。

シーケンス番号*:* IMS コプロセッサーによって生成されるソース・ステートメントには、シーケンス番号は含まれません。

ステートメント・ラベル*:* PROCEDURE DIVISION 内の実行可能 SQL ステートメントの前に、必要に応じて、段落名を付けることができます。

WHENEVER ステートメント*:* SQL WHENEVER ステートメントの GOTO 文節のターゲットは、PROCEDURE DIVISION にあるセクション名または非修飾の段落名でなければなりません。

COBOL に関する特殊な考慮事項*:* 以下の考慮事項が、COBOL で作成されるプログラムに適用されます。

- ホスト変数名として複数レベル構造でエレメントを使用する COBOL プログラムでは、IMS コプロセッサーは最も低い 2 レベル名を生成します。
- 数値の切り捨てを防ぐには、次のいずれかの方法を使用してください。
  - 2 進整数のホスト変数には、COMP-5 データ・タイプを使用します。
  - 以下の COBOL コンパイラー・オプションを指定します。
    - TRUNC(OPT) - アプリケーションが各 2 進変数へ移送しようとするデータ精度が、2 進変数の PICTURE 文節での定義を上回っていないことが確実な場合に指定します。

- TRUNC(BIN) - 各 2 進変数へ移送しているデータの精度が PICTURE 文節の値を超える可能性がある場合に指定します。

IMS は、COBOL コンパイラー・オプション TRUNC(BIN) を指定したか、あるいは COMP-5 データ・タイプを使用したかのように、2 進整数ホスト変数に値を割り当てます。

- COBOL 表意定数 (ZERO や SPACE など)、記号文字、参照修正、および添え字は、SQL ステートメント内では使用できません。
- SQL ID の命名規則を順守します。しかし、COBOL の場合のみ、SQL ID の名前は、IMS オブジェクトに許される長さを超えなければ、COBOL ワードの命名規則に従うことができます。例えば、名前 1ST-TIME は有効な COBOL ワードであるため、有効なカーソル名ですが、名前 1_TIME は、有効な SQL ID または有効な COBOL ワードではないため無効です。
- ハイフンについては、以下の規則に従ってください。
  - 減算の演算子としてハイフンを使用する場合には、前後にスペースを入れてください。 IMS は通常、前後にスペースがないハイフンについてはホスト変数名の一部と解釈します。
- SQL ステートメントを COBOL `PERFORM ... THRU` 段落に組み込み、かつ SQL ステートメント `WHENEVER ...GO` も指定すると、COBOL コンパイラーは、警告メッセージ IGYOP3094 を戻します。このメッセージは、問題を指摘している可能性があります。この使用法は推奨されません。

関連概念:

SQL ID (アプリケーション・プログラミング API)

関連タスク:

644 ページの『アプリケーションでの SQL ステートメントの使用』

642 ページの『SQL 記述子域の定義』

COBOL プログラムの SQL ステートメントでの区切り文字

IMS が特定の SQL ステートメントの終わりを判別できるように、COBOL プログラム内で SQL ステートメントを区切る必要があります。

COBOL プログラム内の SQL ステートメントは、開始キーワード `EXEC SQLIMS` と `END-EXEC` で区切ります。

例

`EXEC SQLIMS` および `END-EXEC` を使用して、COBOL プログラムの SQL ステートメントを区切ります。

```
EXEC SQLIMS
  an SQL statement
END-EXEC.
```

COBOL でのサポートされている SQL 集約関数

COBOL および .NET アプリケーション (IMS Enterprise Suite Microsoft .NETを使用) では、SQL 集約関数がサポートされます。

- AVG

- COUNT
- MAX
- MIN
- SUM

これらの関数は、GROUP BY 節だけでなく、ORDER BY 節と連携して動作します。

- ORDER BY
  - ASC
  - DESC
- GROUP BY

制約事項: サポートされている SQL 集約関数は、引数としてセグメント内の単一フィールド名のみを受け入れます (DISTINCT キーワードは許可されていません)。

次の表は、集約関数で受け入れられるフィールドのデータ型と、ResultSet での結果のデータ型を示しています。

表 *95.* サポートされている *SQL* 集約関数とそれらがサポートするデータ型

機能	引数の型	結果および結果の型
AVG	COBOL プログラムで SQL ステートメント用にサポートされているすべての数値データ型がサポートされます。これには、 TINYINT、 SMALLINT、 INTEGER、 BIGINT、 ゾーン 10 進数、およびパック 10 進数が含まれます。	• この関数は、NULL 値を除外した引数値から生成された値セットに対して適用されます。 • INTEGER の場合、平均値の小数部分は破棄されます。結果がヌルになる場合があります。 • DECIMAL 以外のデータ型の場合、結果のデータ型は常に LONG です。 • 引数値のデータ型が DECIMAL の場合、結果はパック 10 進数です。結果の位取りは、引数値の位取りと同じであり、結果の精度は 31 桁です。 • この関数が空のセットに適用された場合、結果はヌル値になります。 • 平均値は、結果のデータ型の範囲内でなければなりません。

表 *95.* サポートされている *SQL* 集約関数とそれらがサポートするデータ型 *(*続き*)*

機能	引数の型	結果および結果の型
COUNT	サポートされているすべてのデータ型	• 所定の列にある値の合計数がカウントされる場合 (COUNT(*column*))、NULL 値はカウントされません。 • 表内の行の数がカウントされる場合 (COUNT(*))、NULL 値もカウントされます。 • 空の表の COUNT(*) は、値が 0 である 1 つの行を返します。
MAX	TINYINT、 INTEGER、 BIGINT、 ゾーン 10 進数、 パック 10 進数、 CHAR、 BINARY、 DATE、 TIME、 および TIMESTAMP	• 文字ストリング引数およびバイナリー・ストリング引数は、長さ属性が 32704 を超えることはできません。 • 結果のデータ型とその他の属性 (例えば、ストリングや日時値の長さと CCSID) は、引数値のデータ型および属性と同じです。 • 結果がヌルになる場合があります。 • この関数は、NULL 値を除いた引数値から生成された値セットに対して適用されます。 • この関数が空のセットに適用された場合、結果はヌル値になります。
MIN	TINYINT、 INTEGER、 BIGINT、 ゾーン 10 進数、 パック 10 進数、 CHAR、 BINARY、 DATE、 TIME、 および TIMESTAMP	• 結果のデータ型とその他の属性 (例えば、ストリングや日時値の長さと CCSID) は、引数値のデータ型および属性と同じです。 • 結果がヌルになる場合があります。 • この関数は、NULL 値を除いた引数値から生成された値セットに対して適用されます。 • この関数が空のセットに適用された場合、結果はヌル値になります。

表 95. サポートされている SQL 集約関数とそれらがサポートするデータ型 (続き)

機能	引数の型	結果および結果の型
SUM	COBOL プログラムで SQL ステートメント用にサポートされているすべての数値データ型がサポートされます。これには、TINYINT、SMALLINT、 INTEGER、BIGINT、 ゾーン 10 進数、およびパック 10 進数が含まれます。	• 合計値は、結果のデータ型の範囲内でなければなりません。 • この関数は、NULL 値を除いた引数値からなる値セットに対して適用されます。 • この関数が空のセットに適用された場合、結果はヌル値になります。 • 合計が実行される順序は定義されていませんが、どの中間結果も結果のデータ型の範囲内でなければなりません。 • 引数値のデータ型が DECIMAL の場合、結果はパック 10 進数です。結果の位取りは、引数値の位取りと同じであり、結果の精度は 31 桁です。

集約関数によって生成される列名

集約関数での生成される列名は、下線文字 (_) で区切られた集約関数名とフィールド名の組み合わせです。例えば、ステートメント `SELECT MAX(age)` の場合は、結果は列名 MAX_age となります。

集約関数の引数フィールドはテーブル修飾され、生成される列名は下線文字 (_) で区切られた集約関数名、テーブル名、および列名の組み合わせとなります。例えば、`SELECT MAX(Employee.age)` の場合は、結果は列名 MAX_Employee_age となります。

集約関数は最初に実行され、その後、必要な数の結果行が結果セットから取り出されます。

ORDER BY 節および GROUP BY 節の使用

GROUP BY 節または ORDER BY 節で指定されるフィールド名は、SELECT ステートメントで指定されるフィールド名と正確に一致する必要があります。 GROUP BY を正しく行うために、SELECT リスト内で指定されるフィールドは、GROUP BY リスト内でも指定されている必要があります。

```
SELECT HOSPNAME, COUNT(PATNAME) FROM PCB01.HOSPITAL, PATIENT GROUP BY HOSPNAME
 ORDER BY HOSPNAME
```

制約事項: 集約関数を GROUP BY ステートメントや ORDER BY ステートメントの中で使用することはできません。例えば、GROUP BY COUNT(PATNAME) や ORDER BY AVG(COST) はサポートされません。

関連資料:

SELECT (アプリケーション・プログラミング API)

データの追加と変更

ご使用のアプリケーション・プログラムは、ユーザーが適切なレベルのアクセス権限を持つ IMS セグメントのデータを照会、変更、または削除できます。

行の挿入

SQL INSERT ステートメントを使用して、データをセグメントに挿入します。

セグメントまたはビューに新しい行を追加するには、INSERT ステートメントを使用します。INSERT ステートメントを使用して、単一行に挿入するフィールド値を指定することができます。 VALUE 文節を使用して、定数またはパラメーター・マーカーを指定することができます。

挿入したすべての行に対して、すべてのキー・フィールドの値を指定する必要があります。 INSERT 呼び出しで値を指定しない場合、IMS が値を 0 に設定します。

単一行の挿入**:**

INSERT ステートメントの VALUES 文節を使用すれば、複数のフィールド値から成る単一行をセグメントに挿入できます。値を指定するフィールドのすべての名前を指定することも、フィールド名のリストを省略することもできます。フィールド名リストを省略する場合は、フィールドのすべての値を指定する必要があります。フィールドは、まず IMS カタログ内の場所によって順序付けされ、次にフィールドの長さで順序付けされます。

非ルート・レベルで表にレコードを挿入する場合は、表のすべての外部キー・フィールドの値を指定する必要があります。外部キー・フィールドにより、標準の SQL 処理を使用して階層パス内に挿入する新規レコード (またはセグメント・インスタンス) が正しく配置されます。これはリレーショナル・データベースの外部キーと似ています。

推奨事項**:** INSERT ステートメントでは、以下の理由によって値を指定するすべてのフィールドの名前を指定します。

- INSERT ステートメントがセグメントの形式に左右されない。 (例えば、フィールドがセグメントに追加される際、ステートメントの変更は不要です。)
- 値を順序どおりに指定しているかどうかが確認できる。
- ソース・ステートメントを読むだけで、処理の内容が理解しやすくなる。

フィールド名をリストする場合は、それらに対応する値をフィールド名のリストと同じ順序で指定しなければなりません。

例**:** 次のステートメントは、新しい病院に関する情報を HOSPITAL セグメントに挿入します。

```
INSERT INTO PCB01.HOSPITAL (HOSPCODE, HOSPNAME)
  VALUES ('R1210050000A', 'MALLEY CLINIC')
```

新しい病院を HOSPITAL セグメントに挿入した後、SELECT ステートメントを使用して、セグメントにロードした内容を確認することができます。次の SQL ステートメントにより、挿入した新しい部門行のすべてが表示されます。

```
SELECT HOSPCODE, HOSPNAME
  FROM PCB01.HOSPITAL
```

結果セグメントは、以下の出力のようになります。

```
+------------+----------------+
|HOSPCODE    |HOSPNAME        |
+------------+----------------+
|R1210010000A|ALEXANDRIA      |
|R1210020000A|SANTA TERESA    |
|R1210030000A|SANTA CLARA     |
|R1210040000A|NEW ENGLAND     |
|R1210050000A|MALLEY CLINIC   |
```

例: 次のステートメントは、新規 WARD レコードを特定の HOSPITAL 表に挿入します。この例では、WARD 表には外部キー HOSPITAL_HOSPCODE があります。新規レコードは、HOSPITAL 表に、値が 'R1210050000A' の HOSPCODE が存在する場合にのみ挿入されます。

```
INSERT INTO PCB01.WARD
  WARDNO, HOSPITAL_HOSPCODE, WARDNAME)
  VALUES ('0001', 'R1210050000A', 'EMGY')
```

例: 次のステートメントも、列名を指定せずに行を HOSPITAL セグメントに挿入します。すべての列の値を VALUES 文節で指定する必要があります。

```
INSERT INTO PCB01.HOSPITAL
  VALUES (900, 'R1210050000A', 'MALLEY CLINIC');
```

セグメント・データの更新

フィールドの値を別の値に変更したり、フィールドの値を完全に除去したりすることができます。

セグメントのデータを変更するには、UPDATE ステートメントを使用します。UPDATE ステートメントを使用すると、フィールドの値を空ストリングに変更することによって、(行を除去することなく) フィールドから値を除去することもできます。

例: 次のステートメントは、病院「H001007」の病院名を「MISSION CREEK」に更新します。

```
UPDATE HOSPITAL SET HOSPNAME = 'MISSION CREEK'
WHERE HOSPITAL.HOSPCODE = 'H001007'
```

SET 文節によって、更新したいフィールド名と、そのフィールドに割り当てたい値を指定します。SET 文節のフィールドの値は、以下の項目のいずれかに置き換えることができます。

- 式としては、以下のいずれかの項目を指定できます。
  - 定数
  - パラメーター・マーカー

次に、更新する行を指定します。

- 単一行を更新する場合、1 行だけを指定する WHERE 文節を使用します。
- 複数行を更新する場合、更新したい行だけを指定する WHERE 文節を使用します。

WHERE 文節を省略すると、IMS は、セグメント内のすべての行を指定した値で更新します。識別したい行を IMS が見つけることができない場合、SQLIMSCODE の 100 がアプリケーションに返されます。

UPDATE が正常に行われると、SQLIMSCA の SQLIMSERRD(3) には更新された行数が入ります。この数には、UPDATE ステートメントで指定したセグメントの中で更新された行の数だけが含まれています。

更新規則:

更新値は、以下の規則を満たしている必要があります。満たしていない場合、あるいは UPDATE ステートメントの実行中に他のエラーが発生した場合は、どの行も更新されず、カーソルの位置は変更されません。

- 割り当て。更新値は、言語エレメントに記述された割り当て規則を使用して、列に割り当てられます。
- 非ルート・レベルで表内のレコードを更新する場合、更新する正確なレコード (またはセグメント・インスタンス) を識別するために、表のすべての外部キー・フィールドの値を指定する必要があります。
- 外部キー・フィールドでの UPDATE の実行は無効です。

IMS は、UPDATE ステートメントの実行中にエラー (例えば、更新値がそのフィールドには大きすぎるなど) を検出すると、エラーを戻します。エラーを受け取ると、アプリケーションは、既に変更されている行の管理方法を決定する必要があります。変更をコミットまたはロールバックすることができます。

次のステートメントは、特定の HOSPITAL の下の WARD レコードを更新します。この例では、WARD 表には仮想外部キー HOSPITAL_HOSPCODE があります。レコードは、HOSPITAL 表に、値が 'H5140070000H' の HOSPCODE が存在する場合にのみ更新されます。

```
UPDATE WARD SET WARDNAME = 'EMGY',
   DOCCOUNT = '2', NURCOUNT = '4'
WHERE HOSPITAL_HOSPCODE = 'H5140070000H'
   AND WARDNO = '01'
```

セグメントからのデータの削除

セグメントからデータを削除するには、セグメントから 1 つ以上の行を削除するか、セグメントからすべての行を削除するか、セグメントからフィールドをドロップします。

セグメント内の 1 つ以上の行を削除するには、以下のようにします。

- WHERE 節を指定した DELETE ステートメントを使用して、検索条件を指定します。

  DELETE ステートメントは、WHERE 文節で指定した検索条件を満たす行の数に応じて、セグメントの行を除去しないか、1 つ以上の行を除去します。

  次の DELETE ステートメントは、病院名が ALEXANDRIA および SANTA TERESA である HOSPITAL セグメント内の各行を削除します。

```
DELETE FROM PCB01.HOSPITAL WHERE HOSPNAME = 'ALEXANDRIA' OR HOSPNAME = 'SANTA TERESA';
```

このステートメントを実行すると、IMS は HOSPITAL セグメントから検索条件を満たす行を削除します。

IMS が DELETE ステートメントの実行中にエラーを検出した場合、アプリケーションは、変更をコミットおよびロールバックして、SQLIMSCA 内の SQLIMSCODE および SQLIMSSTATE 変数にエラー・コードを返します。 (セグメント内のデータは変更されません。)

DELETE が正常に行われると、SQLIMSCA の SQLIMSERRD(3) には削除された行数が入ります。この数には、DELETE ステートメントで指定したセグメントの中で削除された行の数だけが含まれています。

セグメント内のすべての行を削除するには、以下のようにします。

- WHERE 節を指定せずに DELETE ステートメントを使用します。

  次の DELETE ステートメントは、HOSPITAL セグメントのすべて行を削除します。

  ```
  DELETE FROM HOSPITAL;
  ```

  このステートメントを実行すると、このセグメントは残りますが (つまり、表に行を挿入できる)、表の中は空になります。

関連概念:

SQL 連絡域 (SQLIMSCA) (アプリケーション・プログラミング API)

データへのアクセス

ご使用のプログラムは、ユーザーが読み取り権限を持つ IMS セグメントから、SQL SELECT ステートメントを使用してデータを読み取ることができます。

関連概念:

761 ページの『IMS Universal JDBC ドライバーを使用した IMS データベースにアクセスする SQL 照会の作成』

SELECT ステートメントを使用したデータのリトリーブ

データをリトリーブする最も簡単な方法は、SQL SELECT ステートメントを使用して結果セグメントを指定することです。リトリーブするフィールドと行を指定できます。

IMS データを選択するためにフィールド名を知っている必要はありません。指定したセグメントの選択行ごとにすべてのフィールドをリトリーブすることを示すには、SELECT 文節にアスタリスク (*) を使用してください。これらのフィールドの値を表示するには、フィールド名を指定する必要があります。

SELECT * ステートメント内のフィールドは、まず IMS カタログ内の場所によって順序付けされ、次にフィールドの長さで順序付けされます。

例: **SELECT *:** 次のステートメントは、PATIENT セグメントのすべてのフィールドをリトリーブします。

```
SELECT *
  FROM PCB01.HOSPITAL;
```

結果セグメントは、以下の出力のようになります。

```
+------------+-----------------+
|HOSPCODE    |HOSPNAME         |
+------------+-----------------+
|R1210010000A|ALEXANDRIA       |
|R1210020000A|SANTA TERESA     |
|R1210030000A|SANTA CLARA      |
|R1210040000A|NEW ENGLAND      |
```

WHERE 文節が指定されていないので、このステートメントは、すべての行のデータをリトリーブします。

ホスト変数の互換性とパフォーマンス上の理由から、静的ホスト構造にフェッチする場合、SELECT * は推奨されません。

注:

- SELECT * が参照するセグメントにフィールドを追加すると仮定します。そのフィールドの受け取りホスト変数を定義していない場合、エラーが発生します。
- アスタリスクを使用する代わりに、SELECT ステートメントにフィールド名をリストすることで、SELECT * の使用で発生する可能性がある問題を回避することができます。また、受け取りホスト変数と結果セグメント内のフィールドの関係を確認することもできます。

いくつかのフィールドの選択: **SELECT** フィールド名:

各フィールドの名前を指定して、リトリーブしたいフィールド (複数可) を選択します。すべてのフィールドは、セグメント内の順序ではなく、アプリケーションで指定した順序で取り出されます。

例: **SELECT** フィールド名: 次のステートメントは、WARD 表と PATIENT 表から、それぞれ病棟名と患者名をリトリーブします。

```
SELECT HOSPNAME FROM PCB01.HOSPITAL
```

結果セグメントは、以下の出力のようになります。

```
+-----------------+
|HOSPNAME         |
+-----------------+
|ALEXANDRIA       |
|SANTA TERESA     |
|SANTA CLARA      |
|NEW ENGLAND      |
```

1 つの SELECT ステートメントによって選択できるデータのフィールド数は、1 列から最大 750 フィールドまでです。

検索条件を使用した行の選択: **WHERE:**

WHERE 節を使用することにより、特定の条件を満たす行を選択できます。
WHERE 文節は検索条件を指定します。 1 つの探索条件 は、1 つまたは複数の述部から構成される。述部 は、IMS にセグメントの行ごとに適用させるテストを指定します。

IMS は、各行の述部を、真、偽、または不明であると評価します。結果が不明になるのは、オペランドが NULL の場合だけです。

次のセグメントに、比較のタイプ、比較演算子、および WHERE 文節の述部で使用できる比較演算子の各タイプの例を示します。

表 *96.* 条件で使用される比較演算子

比較のタイプ	比較演算子	例
等しい	=	HOSPCODE = 'R1210010000A'
等しくない	<>	HOSPCODE <> 'R1210020000A'
より小さい	<	SALARY < 30000
以下	<=	AGE <= 25
より小さくない	>=	AGE >= 21
より大きい	>	WARDNO > '0001'
以上	>=	WARDNO >= '0003'
より大きくない	<=	PATNUM <= '0010'
2 つの条件のうち少なくとも 1 つ	OR	HOSPCODE >= 'R1210010000A' OR HOSPCODE < 'R1210050000A'
2 つの条件とも	AND	HOSPCODE = 'R1210050000A' AND HOSPNAME = 'SANTA TERESA'

これらの形式のどちらの述部も、1 つの値がもう 1 つの値と等しいか、両方の値が NULL に等しい式を作成します。

関連概念:

642 ページの『ホスト変数』

述部 (アプリケーション・プログラミング API)

結果セグメントのフォーマット

SQL ステートメントは、結果セグメントと呼ばれるセグメントにデータを戻します。フィールド名、行の順序付け方法、行に番号を付けるかどうかなど、結果セグメントのいくつかの属性を指定できます。

結果セグメント:

SQL ステートメントによってリトリーブされるデータは常にセグメント形式で、結果セグメントと呼ばれます。データのリトリーブ元のセグメントと同様に、結果セグメントにも行とフィールドがあります。プログラムは、このデータを一度に 1 行ずつフェッチしてきます。

結果セグメントの例: 以下の SELECT ステートメントを発行すると仮定します。このステートメントは、HOSPITAL セグメントから病院コードと名前をリトリーブし、その結果を病院名によって昇順で並べ替えます。

```
SELECT HOSPCODE, HOSPNAME
  FROM PCB01.HOSPITAL
  ORDER BY HOSPNAME
```

結果セグメントは、以下の出力のようになります。

```
+------------+-----------------+
|HOSPCODE    |HOSPNAME         |
+------------+-----------------+
```

```
|R2100010000A|ALEXANDRIA       |
|R2100040000A|NEW ENGLAND      |
|R2100030000A|SANTA CLARA      |
|R2100020000A|SANTA TERESA     |
```

結果セグメントの行の順序付け:

結果セグメントの行が特定の方法で確実に順序付けられるようにするには、SELECT ステートメントに順序を指定する必要があります。順序を指定しなければ、IMS はどのような順序でも行を戻すことができます。

特定の順序で行をリトリーブするには、ORDER BY 文節を使用します。ORDER BY の使用は、ソートしたい順序で行を並べることを保証できる唯一の方法です。次のトピックで、ORDER BY 文節の使用法を説明します。

ORDER BY 文節でのソート・キーの指定:

選択された行の順序は、ORDER BY 文節に指定したソート・キーに基づきます。ソート・キーは、セグメントのフィールド名です。 IMSは、まず第 1 ソート・キーを基準に、続いて第 2 ソート・キー、以下同様にして行の順序を決めます。

行は昇順でも降順でもリストできます。NULL 値は、昇順の場合は最後に、降順の場合は最初に示されます。

例: ソート・キーとしてフィールド名を指定する **ORDER BY** 節: 次のステートメントは、すべての病院名をアルファベット順でソートしてリトリーブします。

```
SELECT HOSPITAL.HOSPNAME
   FROM PCB01.HOSPITAL
   ORDER BY HOSPITAL.HOSPNAME ASC
```

小さいセットの行に対するリトリーブの最適化

数千の行のうち、照会の条件を満たす数行のみ必要である場合は、指定した数の行のみを返すように IMS に対して指示できます。

質問: 照会を満足させる行が大量にある場合、そのうちの数行だけを検索したいことを IMS に知らせるにはどのようにしますか?

回答: FETCH FIRST *n* ROWS ONLY を使用してください。

最初の数行だけをリトリーブしたい場合があります。例えば、最初の 50 行をリトリーブするには、次のようにコーディングします。

```
SELECT * FROM PCB01.HOSPITAL
FETCH FIRST 50 ROWS ONLY
```

FETCH FIRST *n* ROWS ONLY を使用して、結果セグメントの行数を *n* 行に制限します。FETCH FIRST *n* ROWS ONLY には、次のような利点があります。

- FETCH ステートメントを使用して結果セグメントからデータをリトリーブする場合に、FETCH FIRST n ROWS ONLY 文節を使用すれば、必要な行数だけが IMS によってリトリーブされます。このことは、特に分散アプリケーションの場合にはパフォーマンスに有利です。 FETCH ステートメントを使用して *n+1* 番目の行をリトリーブしようとすると、IMS は、+100 SQLCODE を戻します。

SELECT * の使用による影響

一般的に、SELECT * を使用する必要があるのは、すべてのフィールドを選択する場合のみです。そうでなければ、表示したい特定のフィールドを指定してください。

質問*:* SELECT * を使用することにどのような意味があるのですか?

回答*:* 一般に、IMS は選択されたフィールドの数の影響を受けるため、必要とするフィールドのみを選択する必要があります。SELECT * を使用するのは、隠しフィールドを除くすべてのフィールドを選択する必要があることが確実であるときだけにしてください。

可変長データベース・セグメントのサポート

SQL 言語の規則では、ターゲット・データベースがリレーショナルであることを前提としています。リレーショナル・データベース・マネージャーは、外部アプリケーション・プログラムによって管理される「可変長」データ構造の概念を使用しません。

IMS は階層データベースであるため、IMS は、SQL ステートメントを IMS Database Manager で解釈可能な DL/I 呼び出しに変換します。IMS Database Manager を使用している場合、アプリケーションでは、セグメントの LL フィールドを使用して、個々の可変長セグメント・インスタンスの長さを管理する必要があります。IMS Universal DL/I ドライバー または SQL Support for COBOL を使用するアプリケーションは、LL フィールドの管理を担当します。

IMS Universal データベース・リソース・アダプターまたは IMS Universal JDBC ドライバーを使用するアプリケーション・プログラムは、IMS データベースを標準 JDBC データ・ソースとして取り扱います。 IMS Universal Database リソース・アダプターおよび IMS Universal JDBC ドライバーは、アプリケーションの代わりに LL フィールドを管理するため、アプリケーション・プログラムが入出力域のセグメント長やサイズを管理する必要はありません。読み取り操作の場合、IMS Universal Database リソース・アダプターおよび IMS Universal JDBC ドライバーは、返されたすべてのセグメントとフィールドのオフセットおよび長さを処理します。デフォルトでは、SQL 結果セットには LL フィールドの情報は含まれていません。更新操作または挿入操作の場合、可変長セグメントの各インスタンスは、セグメント・インスタンスの最大フィールド (フィールド長とオフセットで決まる) が入るように自動的に拡張されます。

SQL Support for COBOL では、COBOL アプリケーションが LL フィールドの管理を担当します。

可変長セグメントの一部のフィールドは、NULL 可能な場合があります。IMS では、NULL 可能フィールドは、開始オフセット、またはオフセットと長さの組み合わせが、セグメントの最小長より大きいフィールドです。インスタンスの LL 値と、NULL 可能フィールドのオフセットと長さの組み合わせを比較することで、NULL 可能フィールドが特定のセグメント・インスタンスに存在するかどうかを判別できます。LL 値がフィールドのオフセットと長さの組み合わせより小さい場合、そのフィールドは NULL です。例えば、セグメント定義にオフセット 50 から始まる長さが 5 のフィールドがあり、セグメントの最小長が 55 より小さい場合は、

そのフィールドは NULL 可能です。特定のセグメントのインスタンスの LL 値が 55 より小さい場合、そのインスタンスのフィールドは NULL です。

COBOL の場合は NULL 標識変数、IMS Universal Database リソース・アダプターおよび IMS Universal JDBC ドライバーの場合は java.sql.ResultSet.wasNull メソッドを使用することで、LL データを調査することなく可変長セグメントのインスタンスに NULL 可能フィールドがあるかどうかを判別することもできます。

IMS Universal Database リソース・アダプター および IMS Universal JDBC ドライバー を伴う LL フィールドの使用

デフォルトでは、可変長セグメントの LL フィールドは、SQL 照会の可視列として返されません。LL フィールドが要求されない場合、IMS Universal データベース・リソース・アダプターおよび IMS Universal JDBC ドライバーがアプリケーション・プログラムに代わって LL フィールドを管理します。アプリケーション・プログラムが IMS へのデータ接続を作成するときに、明示的に列を要求しない限りは、いかなるタイプの照会 (`SELECT *` を含む) も LL フィールドのデータにアクセスできません。

LL フィールドをアプリケーション・レベルで管理し、アプリケーションで IMS Universal Database リソース・アダプターまたは IMS Universal JDBC ドライバーを使用する場合、LLField プロパティーを `true` に設定することで、LL フィールドのデータを明示的に要求する必要があります。

アプリケーションでは、以下のいずれかのインターフェースの標準プロパティー・リストにある LLField プロパティーを `true` に設定できます。

`java.sql.DriverManager.getConnection(String url, Properties properties)`

`com.ibm.ims.jdbc.Datasource.setProperties(Properties properties)`

LLField=true プロパティーが設定されている場合、すべての操作で、LL フィールドは標準 SQL 結果セットの通常列として表示されます。ユーザーは、LL フィールド・データの読み取り、挿入、更新を直接行うことができます。LL フィールドのデータを削除すると、関連のデータベース・レコードのその他の部分も削除されます。フィールドを NULL 状態に設定するには、セグメント (LL フィールド列の値) の長さを、セグメント内のフィールドのオフセットよりも小さい値に設定します。

LL フィールドの長さは 2 バイトで、BINARY、SHORT、または USHORT データとして処理する必要があります。

COBOL を伴う LL フィールドの使用

LL フィールドは、すべての操作で標準 SQL 結果セットの通常列として取り扱われます。ユーザーは、LL フィールド・データの読み取り、挿入、更新を直接行うことができます。LL フィールドのデータを削除すると、関連のデータベース・レコードのその他の部分も削除されます。フィールドを NULL 状態に設定するには、セグメント (LL フィールド列の値) の長さを、セグメント内のフィールドのオフセットよりも小さい値に設定します。

LL フィールドの長さは 2 バイトで、BINARY、SHORT、または USHORT データとして処理する必要があります。

IMS Universal DL/I ドライバーを使用した NULL フィールドのインスタンスの検査

IMS Universal DL/I ドライバーを使用するアプリケーションは、常に可変長セグメントの LL フィールドのデータを受け取ります。次の 2 つの方法のいずれかで、セグメント・インスタンス内のフィールドが NULL であるかどうかを判別することができます。LL フィールドのデータをフィールドのオフセットと比較する方法、または com.ibm.ims.dli.Path.wasNull() メソッドを使用する方法です。

com.ibm.ims.dli.Path.wasNull() メソッドは、読み取られた最後のフィールドが NULL 状態の場合、ブール値を返します。フィールドが NULL の場合、戻り値は `true` です。フィールドが NULL であるかどうかを判別するには、wasNull() メソッドを呼び出す前にフィールドを読み取る必要があります。

COBOL での NULL フィールド・インスタンスの検査

SQL for COBOL を使用するアプリケーションは、常に可変長セグメントの LL フィールドのデータを受け取ります。セグメント・インスタンス内のフィールドが NULL であるかどうかを判別するには、LL フィールドのデータをフィールドのオフセットと比較するか、NULL 標識変数を使用します。

関連概念:

可変長セグメント (データベース管理)

関連タスク:

642 ページの『ホスト変数および標識変数の宣言』

660 ページの『COBOL でのホスト変数および標識変数の宣言』

カーソルを使用した一連の行のリトリーブ

アプリケーション・プログラムでは、IMS から一連の行をリトリーブすることができます。

結果セグメントから行をリトリーブするには、次のタイプのカーソルを使用します。

- 行位置付けカーソルは、一度に最大 1 行ずつ結果セグメントからリトリーブして、ホスト変数に入れます。 すべての時点で、カーソルが位置付けされるのは、最大 1 行です。 行位置付けカーソルの使用方法については、 686 ページの『行位置付けカーソルを使用したデータへのアクセス』を参照してください。

カーソル

カーソルは、セグメントの行セットから 1 つの行を指すメカニズムです。アプリケーション・プログラムは、カーソルを使用してセグメントから行をリトリーブすることができます。

カーソルのタイプ:

行位置指定のスクロール不能カーソルを宣言して、結果表からデータをリトリーブすることができます。

順方向カーソルの使用:

最も単純なタイプのカーソルは、順方向カーソルです。 行位置付け順方向カーソルは、その結果セグメントを、一度に 1 行ずつ順方向に移動します。

行位置付けカーソルを使用したデータへのアクセス

行位置付けカーソルは、単一行を指し、一度に最大 1 行ずつ結果セグメントからリトリーブます。フェッチ要求を指定して、現行カーソル位置を基準にリトリーブする行を指定できます。

行位置付けカーソルを使用してデータにアクセスするには、以下のようにします。

1. DECLARE CURSOR ステートメントを実行して、カーソルが操作する結果セグメントを定義する。『行カーソルの宣言』を参照してください。
2. OPEN CURSOR を実行して、カーソルをアプリケーションに利用可能にする。 687 ページの『行カーソルのオープン』を参照してください。
3. すべての行がリトリーブされたときにプログラムが行うことを指定する。 687 ページの『データの最後に到達したときに行カーソルが行うアクションの指定』を参照してください。
4. SQL ステートメントを実行し、セグメントからデータをリトリーブする。 688 ページの『行カーソルを使用した SQL ステートメントの実行』を参照してください。
5. CLOSE CURSOR ステートメントを実行して、カーソルをアプリケーションに利用不能にする。 688 ページの『行カーソルのクローズ』を参照してください。

ユーザーのプログラムでは、複数のカーソルを持つことができます。そのそれぞれについて、上述のステップを行います。

行カーソルの宣言:

行位置付けカーソルを使用して行をリトリーブするには、その前にカーソルを宣言する必要があります。カーソルを宣言する際には、カーソルを使用してアクセスする行のセットを指定する必要があります。

行カーソルを宣言するには、DECLARE CURSOR ステートメントを発行します。 DECLARE CURSOR ステートメントは、カーソルに名前を付け、準備済み SELECT ステートメントを指定します。 SELECT ステートメントは、結果セグメントを構成する行の基準を定義します。

次の例は、準備済みステートメント STMT に対して単純形式の DECLARE CURSOR ステートメントを使用して宣言された C1 という名前のカーソルを示しています。

```
EXEC SQLIMS
  DECLARE C1 CURSOR FOR STMT
END-EXEC.
```

関連資料:

- DECLARE CURSOR (アプリケーション・プログラミング API)
- SELECT (アプリケーション・プログラミング API)

行カーソルのオープン:

行カーソルを宣言した後、結果セグメントの先頭行を処理する準備ができたことを IMS に伝える必要があります。このアクションは、カーソルのオープンと呼ばれます。

行カーソルをオープンするには、プログラム内で OPEN ステートメントを実行します。これを受けて、IMS は、DECLARE CURSOR ステートメント内の SELECT ステートメントを使用して、一連の行を識別します。その SELECT ステートメントの検索条件でパラメーター・マーカーを使用する場合、USING 文節を使用してパラメーター・マーカーの値を指定する必要があります。 IMS は、変数の現行値を使用して行を選択します。結果セグメントには、検索条件がどの程度満たされたかによって、ゼロ個、1 個、またはそれ以上の個数の行が含まれます。

OPEN ステートメントの例を次に示します。

```
EXEC SQLIMS
  OPEN C1
END-EXEC.
```

準備済み SELECT ステートメントにパラメーター・マーカーがある場合の OPEN ステートメントの例。準備済み SELECT ステートメントの WHERE 文節にパラメーター・マーカーがあると仮定します。

```
SELECT HOSPCODE, HOSPAME FROM PCB01.HOSPITAL
WHERE HOSPNAME = ?
```

ホスト変数 PARM1 からパラメーター・マーカーの値を設定するには、次のように USING 文節を指定して OPEN ステートメントを使用します。

```
EXEC SQLIMS
OPEN C1 USING :PARM1
END-EXEC.
```

データの最後に到達したときに行カーソルが行うアクションの指定:

行をフェッチするために行カーソルを使用する場合、「データの終わり」状態を認識し、それを処理するようにプログラムをコーディングする必要があります。

プログラムが最後のデータ行までリトリーブされたかどうかを判別するには、SQLIMSCODE フィールドをテストして 100 の値の有無、または SQLIMSSTATE フィールドをテストして '02000' の値の有無を調べます。これらのコードは、FETCH ステートメントが結果セグメントの最後の行をリトリーブしてしまった後、プログラムから続いて FETCH が出されたときに発生します。以下に例を示します。

```
IF SQLIMSCODE = 100 GO TO DATA-NOT-FOUND.
```

この他に、WHENEVER NOT FOUND ステートメントをコーディングする方法があります。 WHENEVER NOT FOUND ステートメントを使用すると、プログラムは別の部分へブランチし、次にそれが CLOSE ステートメントを出します。例えば、FETCH ステートメントが行を戻さなかったときにはラベル DATA-NOT-FOUND にブランチしたいのであれば、次のステートメントを使用します。

```
EXEC SQLIMS
  WHENEVER NOT FOUND GO TO DATA-NOT-FOUND
END-EXEC.
```

WHENEVER NOT FOUND ステートメントについて詳しくは、 657 ページの『SQL ステートメントの実行結果の検査』を参照してください。

行カーソルを使用した **SQL** ステートメントの実行**:**

行カーソルを使用して、FETCH ステートメント、を実行できます。

コピー・データを結果セグメントの行から 1 つ以上のホスト変数にコピーするには、FETCH ステートメントを使用します。

次の例は、HOSPITAL セグメントから選択したフィールドをリトリーブする FETCH ステートメントを示しています。

```
EXEC SQLIMS
  FETCH C1 INTO :HOSPCODE, :HOSPNAME, :WARDNAME, :PATNAME
END-EXEC.
```

プログラムが FETCH ステートメントを実行すると、IMS はカーソルを結果セグメントの行に置きます。この行を、現在行 と呼びます。 IMS は、ここで現在行の内容を、FETCH の INTO 文節で指定したプログラム・ホスト変数にコピーします。この手順は、結果セグメントのすべての行を処理するまで、FETCH を出すたびに繰り返されます。

行カーソルのクローズ**:**

行カーソルが行の処理を終了したときに、リソースを解放したい場合、またはカーソルを再度使用したい場合は、行カーソルをクローズします。あるいは、現行トランザクションの終了時、またはプログラムの終了時に、IMS が自動的にカーソルをクローズするようにもできます。

カーソルが保持しているリソースを解放するには、CLOSE ステートメントを発行してカーソルを明示的にクローズします。

rowset カーソルを再度使用したい場合は、カーソルを再オープンします。

行カーソルをクローズするには、以下のようにします。

CLOSE ステートメントを発行します。 CLOSE ステートメントの例を次に示します。

```
EXEC SQLIMS
  CLOSE C1
END-EXEC.
```

データのコミットまたはロールバック

ご使用のアプリケーションが SQL ステートメントを発行して IMS データベース内のデータを変更した後、アプリケーションでデータベースの変更をコミットまたはロールバックする必要がある場合があります。 IMS データベースの変更をコミットあるいはロールバックするには、IMS DB システム・サービス DL/I 呼び出しを使用します。

例えば、ROLB を発行して変更をロールバックしたり、CHKP を発行して変更をコミットしたりします。

SQL キーワード COMMIT および ROLLBACK は、現在はサポートされていません。

関連資料:

325 ページの『第 17 章 データベースのリカバリーとデータベース保全性の維持』

IMS 上で実行するためのアプリケーションの準備

SQL ステートメントを含むアプリケーションを準備および実行するには、それらのステートメントをコプロセス、コンパイル、およびリンク・エディットする必要があります。

ヒント**:** 作業のやり直しを防ぐため、まず IMS Enterprise Suite Explorer for Development を使用して SQL ステートメントをテストします。次に、SQL ステートメントを使用するプログラムをコンパイルし、すべてのコンパイラー・エラーを解決します。最後に、IMS コプロセッサーを使用して COBOL プログラムのデプロイメントおよびコンパイルを実行し、SQL ステートメントを変換します。

SQL ステートメントの処理

実行する SQL アプリケーションを準備するための最初のステップは、プログラム内の SQL ステートメントを処理することです。ステートメントを処理するには、IMS コプロセッサーを使用します。このステップの実行中に、SQL ステートメントは IMS 言語インターフェース・モジュール (DFSLI000) への呼び出しに置き換えられます。

COBOL アプリケーションの場合、以下のいずれかの方法を使用して、SQL ステートメントを処理することができます。

- プログラムをコンパイルするのに使用するホスト言語用の IMS コプロセッサーを呼び出す。 COBOL ホスト・コンパイラーでは IMS コプロセッサーを使用することができます。 IMS コプロセッサーを呼び出すには、SQLIMS コンパイラー・オプションの後にそのサブオプションを付けて指定します。
  - COBOL の場合、この技法を使用するには、Enterprise COBOL (z/OS 版) バージョン 5 リリース 1 以降が必要です。COBOL IMS コプロセッサーについて詳しくは、「*Enterprise COBOL for z/OS* プログラミング・ガイド」を参照してください。

IMS コプロセッサーは、コンパイル時にプリコンパイラー機能を実行します。 IMS コプロセッサーを使用すると、コンパイラーがプログラムのスキャンを行い、変更されたソース・コードを戻します。

IMS コプロセッサーを使用した SQL ステートメントの処理

IMS コプロセッサーは、コンパイル時に SQL ステートメントを処理します。

IMS コプロセッサーを使用して SQL ステートメントを処理するには、以下のアクションを実行します。

- SQL ステートメントを含む IMS アプリケーションを処理する JCL ジョブをサブミットする。次の情報を組み込みます。
  - プログラムをコンパイルする際の SQLIMS コンパイラー・オプションを指定する。

    この SQLIMS コンパイラー・オプションは、IMS コプロセッサーを呼び出すようにコンパイラーに指示します。IMS コプロセッサーを使用して、デフォルト・オプションを使用するには、SQLIMS のみを指定します。

    以下に例を示します。

    ```
    //COBOL1  EXEC PGM=IGYCRCTL,
    // PARM='LIST,XREF,CP(37),SQLIMS'
    ```

    IMS コプロセッサーは、以下のデフォルトのオプション値を使用します。

    PERIOD
    SQL ステートメントでは、10 進数リテラル内または浮動小数点リテラル内のピリオド (.) は、小数点標識として認識されます。

    APOSTSQL
    SQL ステートメント内では、アポストロフィ (') はストリング区切り文字、二重引用符 (") は SQL 拡張文字として認識されます。

    さらに、ソース・プログラムが書き込まれている CCSID を判別するために、COBOL CODEPAGE オプションが使用されます。現在、EBCDIC CCSID 37 および 1140 のみがサポートされます。
  - コンパイル・ステップ用の JCL の中に以下のデータ・セット用の DD ステートメントを組み込む。
    - IMS ロード・ライブラリー (IGYV5R10.SQGYCOMP)

      IMS コプロセッサーは、IMS モジュールを呼び出して、SQL ステートメントを処理します。このため、コンパイル・ステップの STEPLIB 連結に、IMS ロード・ライブラリー・データ・セットの名前を組み込む必要があります。
    - SQL INCLUDE ステートメント用のライブラリー

      ソース・プログラムへの 2 次入力を指定する SQL INCLUDE *member-name* ステートメントがプログラムに含まれている場合は、*member-name* のデータ・セットも指定する必要があります。コンパイル・ステップの SYSLIB 連結に、*member-name* を含むデータ・セットの名前を組み込みます。

第 6 部 IMS 用の Java アプリケーション開発

IMS は Java プログラミング言語を使用したアプリケーション開発にサポートを提供します。

第 38 章 Java 開発用の IMS ソリューションの概要

Java 開発用の IMS ソリューションのドライバーおよびリソース・アダプターを使用して、IMS データベースへのアクセスや IMS トランザクションの処理を実行する Java アプリケーションを作成できます。

Java 開発用の IMS ソリューションには、IMS Universal ドライバー、IMS Java 従属領域リソース・アダプター、および IMS Transaction Manager リソース・アダプター (IMS TM Resource Adapter) が含まれています。これらのソリューションは、Java オンデマンド機能 FMID を介して提供されます。分散プラットフォームでも実行できるソリューションの場合は、IMS ダウンロード・サイト (http://www.ibm.com/software/data/ims/downloads.html) からダウンロードすることができます。

IMS 内で実行する Java アプリケーションを開発するには、システム・プログラマーが環境をセットアップするための作業も含め、エンドツーエンドのガイダンス情報について、Java in IMS ソリューション導入キットを参照してください。

IMS Universal ドライバー

IMS Universal ドライバーとは、Java ドライバーおよびリソース・アダプターのことであり、z/OS および分散 (非 z/OS) プラットフォームからの IMS へのアクセスを可能にします。IMS Universal ドライバーは、業界標準に基づいて構築され、オープン仕様となっています。 IMS Universal ドライバーでは、同じ LPAR 上の IMS データベースへのローカル接続 (タイプ *2* 接続)、および TCP/IP を介した分散接続 (タイプ *4* 接続) という、2 つの接続タイプがサポートされています。タイプ 2 IMS Universal ドライバーを使用する Java アプリケーションは、IMS サブシステムと同じ論理区画 (LPAR) に配置する必要があります。タイプ 4 IMS Universal ドライバー を使用する Java アプリケーションは、IMS サブシステムと同じ論理区画 (LPAR) 上または異なる LPAR 上のどちらに配置しても構いません。

IMS Universal ドライバーを使用すると、以下のような複数の環境から IMS にアクセスできるようになります。

- WebSphere Application Server for z/OS
- CICS Transaction Server for z/OS
- JMP および JBP 領域があるホスト上の IMS

IMS Universal ドライバーには、以下のものが含まれています。

- IMS Universal Database リソース・アダプター。これは、Java EE コネクター・アーキテクチャー (JCA) 1.6 準拠のリソース・アダプターです。
- IMS Universal JDBC ドライバー。これは、JDBC 4.0 API を実装した Java データベース接続 (JDBC) ドライバーです。
- IMS Universal DL/I ドライバー。これは、従来の DL/I プログラミング・セマンティクスでの呼び出し実行用の Java API です。

IMS Java 従属領域リソース・アダプター

IMS Java 従属領域リソース・アダプターは、一連の Java クラスおよびインターフェースのことであり、Java バッチ処理 (JBP) 領域および Java メッセージ処理 (JMP) 領域内での IMS データベース・アクセスおよび IMS メッセージ・キュー処理をサポートします。 IMS Java 従属領域リソース・アダプターは、JMP または JBP 領域で稼働する Java アプリケーション・プログラムに、メッセージ処理プログラム (MPP) および非メッセージ・ドリブン BMP 領域に提供されているものと同様の、以下のような DL/I 機能を提供できます。

- メッセージの読み取りおよび書き込みのための、IMS メッセージ・キューへのアクセス
- プログラム間通信の実行
- コミットおよびロールバック処理
- IMS DB/TM 環境での IMS データベースへのアクセス
- IMS DB/TM および DCCTL 環境での GSAM データベースへのアクセス
- データベース・リカバリー (CHKP/XRST)

IMS TM Resource Adapter

IMS TM リソース・アダプターを使用すると、Java EE アプリケーションを介して IMS トランザクションにアクセスできるだけでなく、IMS 従属領域で実行される IMS アプリケーションから外部の Java EE アプリケーションに対するコールアウト要求を行うこともできます。IMS TM リソース・アダプターは Java EE Connector Architecture およびその Common Client Interface を実装し、Java EE 1.4 以降の任意の汎用アプリケーション・サーバーで使用できます。

IMS Universal ドライバーおよび IMS Java 従属領域リソース・アダプターが Java または Java EE アプリケーションを介して IMS データにアクセスするためのインターフェースとクラスを提供するのに対して、IMS TM リソース・アダプターは、その名前が示すとおり、基本的に Java EE アプリケーションを介して IMS トランザクションと対話することを目的としています。

関連概念:

701 ページの『第 40 章 IMS Universal ドライバーを使用したプログラミング』

825 ページの『第 41 章 Java 従属領域のプログラミング』

関連タスク:

Java 環境の外部接続の構成 (コミュニケーションおよびコネクション)

関連資料:

Java API 文書 (Javadoc) (アプリケーション・プログラミング API)

第 39 章 階層データベースとリレーショナル・データベースの比較

ここでは、IMS データベースの階層モデルと標準的なリレーショナル・データベース・モデルの相違点について説明します。

データベース・セグメント定義では、セグメントのインスタンスにフィールドを定義します。この定義方法は、リレーショナル表で表内の行に列を定義する方法と類似しています。このように、セグメントはリレーショナル表に関連し、セグメント内のフィールドはリレーショナル表内の列に関連します。

IMS セグメントの名前は、SQL 照会内の表名になり、フィールドの名前は SQL 照会内の列名になります。

階層データベース内のセグメントとリレーショナル・データベース内の表との間の基本的な相違点は、階層データベース内で、セグメントは相互に暗黙的に結合されているという点です。リレーショナル・データベースでは、2 つの表を明示的に結合する必要があります。階層データベース内のセグメント・インスタンスは、その親セグメントおよびその子セグメントとすでに結合されており、これらはすべて同じ階層パス上にあります。リレーショナル・データベース内では、表間のこの関係は外部キーおよび 1 次キーによってキャプチャーされます。

このセクションでは、ディーラー・サンプル・データベースとそのデータベースの関係表現を比較します。ディーラー・サンプル DBD は、**<インストール・ロケーション>¥IMS Explorer samples** ディレクトリーにある IMS Enterprise Suite Explorer for Development で使用可能です。

重要**:** 本書では、リレーショナル・データベースと階層データベースの比較のみを示しています。

ディーラー・サンプル・データベースには、以下の図に示す 5 つのセグメント・タイプが含まれています。 ルート・セグメントは Dealer (ディーラー) セグメントです。 Dealer セグメントの下には、その子セグメントである Model (モデル) セグメントがあります。 Model セグメントの下には、その子セグメントである Order (注文)、Sales (販売)、および Stock (在庫) の各セグメントがあります。

次の図では、ディーラー・サンプル・データベースの構造および各セグメントを示しています。

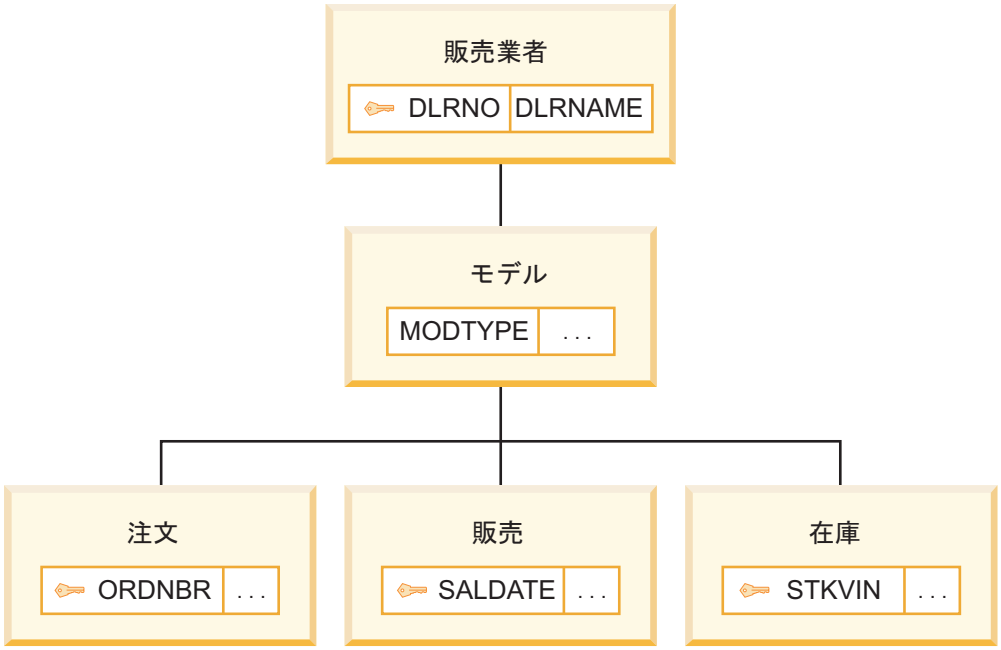


図 *102.* ディーラー・サンプル・データベースのセグメント

Dealer セグメントは、自動車を販売するディーラーを識別します。セグメントには、フィールド DLRNAME 内のディーラー名、およびフィールド DLRNO 内の固有のディーラー番号が含まれています。

ディーラーが扱う自動車のタイプごとに、Model セグメントがそれぞれ適用されます。 Model セグメントには、フィールド MODTYPE にタイプ・コードが入っています。

ディーラーに注文される自動車ごとに Order (注文) セグメントがあります。 Stock セグメントは、ディーラーの在庫の、販売可能な自動車ごとに作成されます。自動車が販売されると、Sales (販売) セグメントが作成されます。

以下の図に、図 102 に示す IMS データベース・レコードの関係表現を示します。

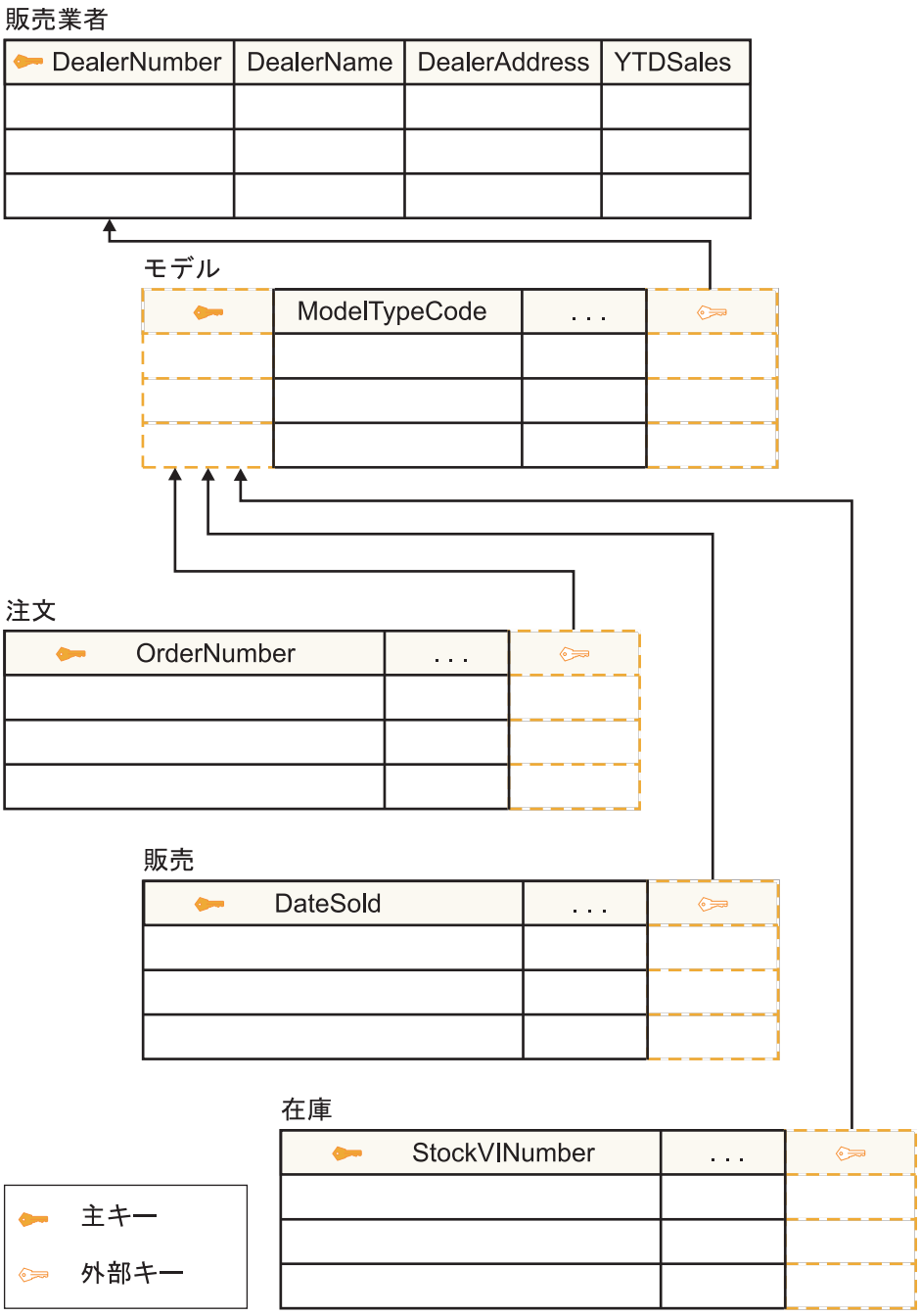


図 *103.* ディーラー・サンプル・データベースの関係表現

セグメントに、リレーショナル・データベースの 1 次キーに類似したユニーク・キーが指定されていない場合は、対応するリレーショナル表を、生成された 1 次キーがその列 (フィールド) リストに追加されているものとして表示します。生成された 1 次キーの例は、前掲の図の Model 表 (セグメント) にあります。リレーショナル・データベース内の参照保全と同様に、例えば、Order (子) セグメントを、特定の Model (親) セグメントの子にすることなくデータベースに挿入することはできません。

フィールド (列) 名が名前変更されていることにも注意してください。セグメントおよびフィールドは、IMS Explorer for Development を使用して、より分かりやすい名前に変更できます。

階層データベースでのセグメントのオカレンスは、リレーショナル・データベースの表の行 (またはタブル) に対応します。

以下の図は、3 つのディーラー・データベース・レコードを示します。

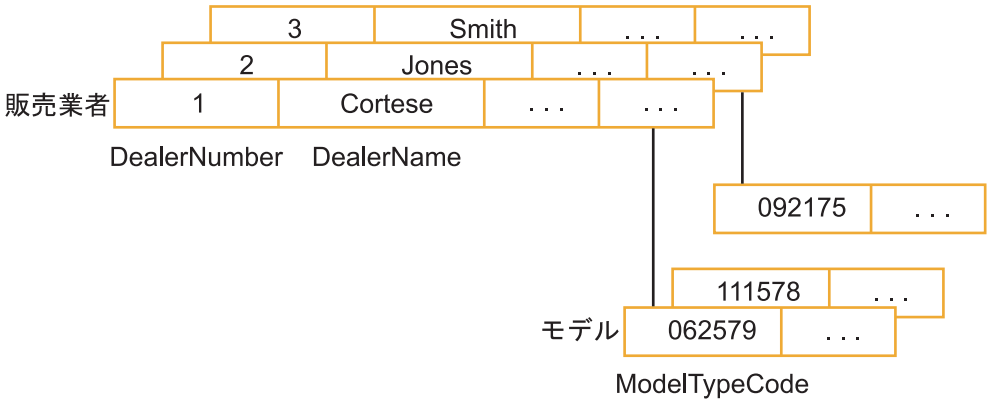


図 *104.* ディーラー・サンプル・データベース内のセグメント・オカレンス

Dealer セグメント・オカレンスは従属 Model セグメント・オカレンスをもちます。

以下の図に、従属 Model セグメント・オカレンスの関係表現を示します。

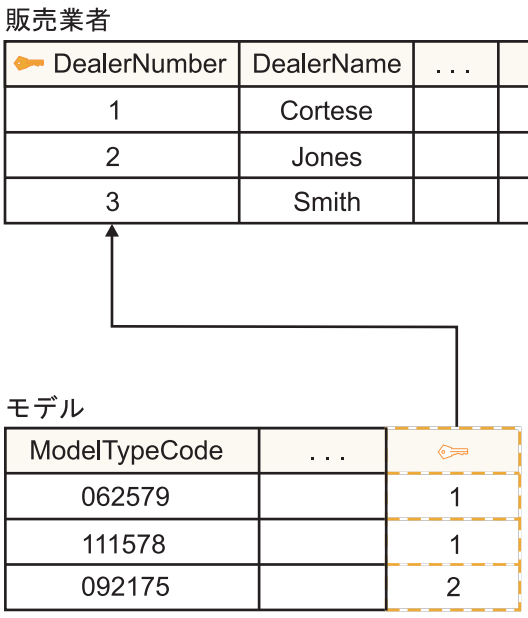


図 *105.* ディーラー・データベース内のセグメント・オカレンスの関係表現

SQL 呼び出しの SELECT ステートメントを示す次の例では、Model は、セグメント名であり、照会では表名として使用されています。

```
SELECT * FROM Model
```

次の例では、ModelTypeCode は、Model セグメント中のフィールドの名前であり、SQL 照会では列名として使用されています。

```
SELECT * FROM Model WHERE ModelTypeCode = '062579'
```

前述の 2 つの例では、Model は SEGM の EXTERNALNAME パラメーターを、ModelTypeCode は DBD の FIELD ステートメントを使用して割り当てた別名です。 EXTERNALNAME パラメーターは、クライアント・アプリケーションがフィールドまたはセグメントを参照する場合に使用する外部別名を指定するためのオプション・パラメーターです。これは、ホスト・リソース名の場合の 8 文字制限に従う必要はありません。外部別名は、IMS カタログがアクティブになっている場合にのみ使用されます。IMS カタログがアクティブであっても、セグメントまたはフィールドの別名が EXTERNALNAME パラメーターに指定されていない場合は、代わりに 8 文字のリソースの IMS 名を使用してください。

第 40 章 IMS Universal ドライバーを使用したプログラミング

IMS Universal ドライバーを使用した IMS 15 用のアプリケーション・プログラムを設計、作成、および保守するには、この章のトピックを参照してください。

関連概念:

693 ページの『第 38 章 Java 開発用の IMS ソリューションの概要』

825 ページの『IMS Java 従属領域の概要』

関連資料:

IMS Universal ドライバー または JDR リソース・アダプターを使用する Java アプリケーション・プログラムのソフトウェア要件 (リリース計画)

IMS Universal ドライバーの概要

IMS Universal ドライバーは、Java アプリケーションに、TCP/IP を使用した z/OS および分散環境から IMS データベースへの接続性およびアクセスを提供するソフトウェア・コンポーネントです。タイプ 2 IMS Universal ドライバーを使用する Java アプリケーションは、IMS サブシステムと同じ論理区画 (LPAR) に配置する必要があります。タイプ 4 IMS Universal ドライバー を使用する Java アプリケーションは、IMS サブシステムと同じ論理区画 (LPAR) 上または異なる LPAR 上のどちらに配置しても構いません。

プログラミング方法

IMS Universal ドライバー は、IMS データにアクセスするための複数のオプションを提供するアプリケーション・プログラミング・フレームワークが備えられています。これらのプログラミング・オプションには、以下のものがあります。

IMS Universal Database リソース・アダプター
: Java Platform, Enterprise Edition (Java EE) 環境からの IMS データベースへの接続、および Common Client Interface (CCI) および Java Database Connectivity (JDBC) インターフェースを使用した IMS データへのアクセスを提供します。

IMS Universal JDBC ドライバー
: IMS データベースに対する SQL ベースのデータベース呼び出しを行うためのスタンドアロン JDBC 4.0 ドライバーを提供します。

IMS Universal DL/I ドライバー
: 従来の DL/I 呼び出しに似たプログラミング・セマンティクスを使用して、IMS データベースに対する詳細な照会を作成するためのスタンドアロン Java API を提供します。

オープン・スタンダード

IMS Universal ドライバー は、以下の業界オープン・スタンダードおよびインターフェースをベースに作成されています。

Java EE Connector Architecture (JCA)
JCA は、IMS などのエンタープライズ情報システム (EIS) を Java EE フレームワークに接続するための Java 標準です。 JCA を使用すると、アプリケーション開発を単純化し、接続管理、トランザクション管理、およびセキュリティー管理など、Java EE アプリケーションによって提供されるサービスを利用することができます。 Common Client Interface (CCI) は、Enterprise JavaBeans (EJB) アプリケーション、JavaServer Pages (JSP)、および Java サーブレットなどの Java EE クライアントから、バックエンド IMS サブシステムへのアクセスを提供する、JCA のインターフェースです。

Java Database Connectivity (JDBC)
JDBC は、データベース・アクセス用の、SQL ベースの標準インターフェースです。これは Java プログラミング言語と、JDBC インターフェースが実装された任意のデータベースとの間の、データベース非依存接続用の業界標準です。

分散リレーショナル・データベース体系 **(DRDA)** の仕様
DRDA は、多様なプラットフォーム上にあるアプリケーションとデータベース・システムとの間の通信を可能にする、オープン・アーキテクチャーです。これらのアプリケーションおよびデータベース・システムは異なるベンダーが提供したものであっても構いませんし、プラットフォームは異なるハードウェア・アーキテクチャーおよびソフトウェア・アーキテクチャーであっても構いません。 DRDA は、分散 2 フェーズ・コミット・トランザクション用の組み込みサポートによる、分散データベース・アクセスを提供します。

関連資料:

DRDA DDM コマンド・アーキテクチャーの参照情報 (アプリケーション・プログラミング API)

IMS Universal ドライバー または JDR リソース・アダプターを使用する Java アプリケーション・プログラムのソフトウェア要件 (リリース計画)

IMS Universal ドライバーを使用した分散およびローカル・コネクティビティー

IMS Universal ドライバーは、IMS データベースに対する分散 (タイプ 4) およびローカル (タイプ 2) コネクティビティーをサポートします。

タイプ 4 IMS Universal ドライバー を使用した分散コネクティビティー

タイプ 4 コネクティビティーでは、IMS Universal ドライバーは、z/OS を含め、TCP/IP および Java 仮想マシン (JVM) をサポートする任意のプラットフォームで実行できます。 IMS データベースにアクセスするには、タイプ 4 IMS Universal ドライバー は最初に IMS Connect と TCP/IP ベースのソケット接続を確立します。IMS Connectは、Open Database Manager (ODBM) を使用した IMS データベースへの要求の送付、およびクライアント・アプリケーションへの応答の返信を受け持ちます。DRDA プロトコルは、タイプ 4 IMS Universal ドライバー の実装

では内部的に使用されます。ユーザーは、DRDA によるタイプ 4 IMS Universal ドライバー の使用を把握する必要はありません。

タイプ 4 IMS Universal ドライバー は、2 フェーズ・コミット (XA) トランザクションをサポートします。IMS Connect は、2 フェーズ・コミット・プロトコルをサポートするために必要な z/OS リソース・リカバリー・サービス (RRS) 構造を構築します。 2 フェーズ・コミット・トランザクションを使用しない場合は、RRS は必要ありません。

IMS との接続を確立するには、IMS への分散 (タイプ 4) コネクティビティーを指示するように **driverType** 接続プロパティーを設定する必要があります。

認証が正しく行われた後、IMS Universal ドライバーは、プログラム仕様ブロック (PSB) 名や IMS データベース・サブシステムなど、その他のソケット接続情報を IMS Connect および ODBM に送信して PSB を割り振り、データベースに接続します。

IMS データベースへの接続は、プログラム仕様ブロック (PSB) が割り振られている場合にのみ確立されます。特定の PSB に対する許可は、PSB の割り振り時に ODBM コンポーネントによって行われます。

タイプ 4 IMS Universal ドライバー では、接続プールをサポートしており、これによって TCP/IP ソケット接続の割り振りおよび割り振り解除に必要な時間は制限されます。接続の再利用を最大化するために、接続のソケット属性だけがプールされます。これらの属性には、ホスト IMS Connect が listen する IP アドレスおよびポート番号が含まれます。その結果、物理ソケット接続を再利用し、このソケットで追加属性を送信して、IMS データベースに接続することができます。タイプ 4 IMS Universal ドライバー のクライアント・アプリケーションが IMS への接続を確立した場合、これは以下のことを意味します。

- 1 つのクライアント・ソケットと 1 つ以上の IMS データベースが入っている割り振り済み PSB との間には、1 対 1 の関係が確立されています。
- IMS Connect と、同時処理できる可能な数のデータベース接続との間には、1 対多の関係が確立されています。
- IMS Connect がユーザー認証を実行します。
- ODBM は、認証されたユーザーによる、特定の PSB へのアクセスが許可されていることを保証します。

次の図は、タイプ 4 IMS Universal ドライバー が、分散環境で実行される Java クライアント・アプリケーションと IMS サブシステムのと間で通信経路を確立する方法を示しています。

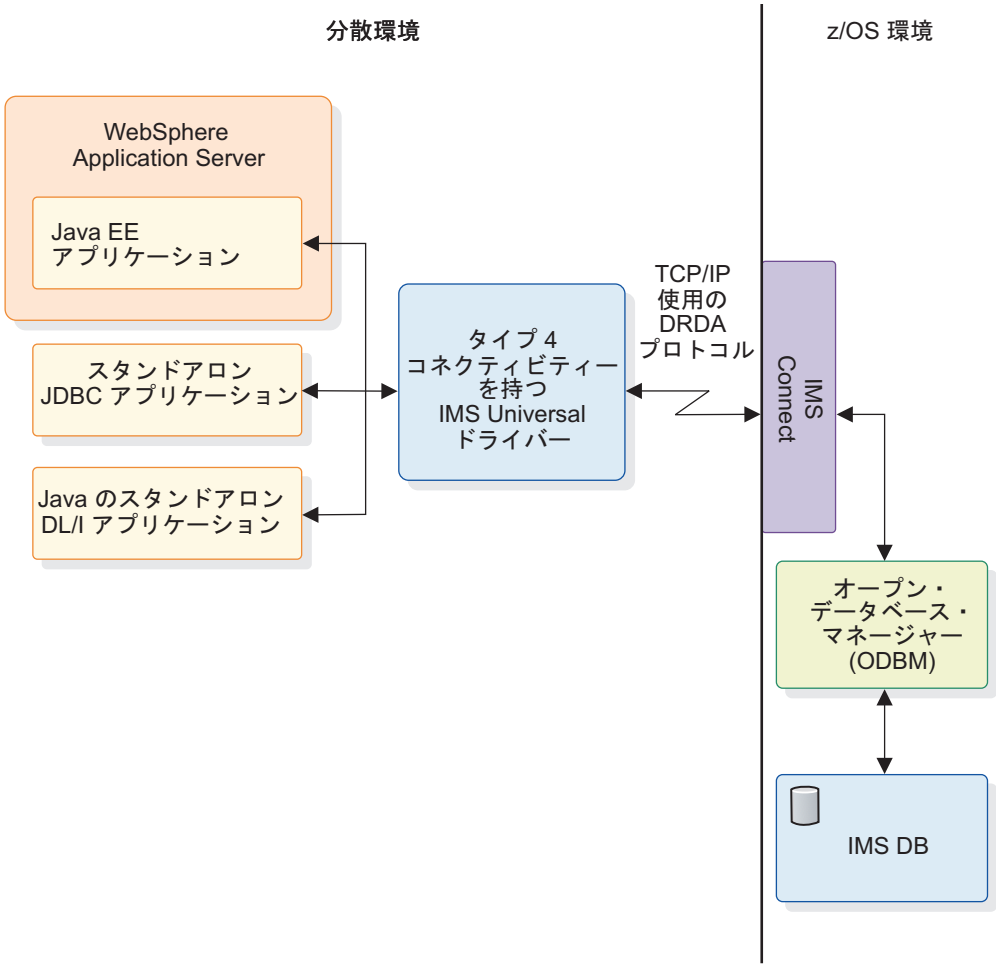


図 *106. タイプ 4 IMS Universal* ドライバー を使用した分散コネクティビティー

また、Java クライアントが z/OS 環境で実行しているが、IMS サブシステムとは別の論理区画にある場合、タイプ 4 IMS Universal ドライバー も使用できます。アプリケーション実行時環境を IMS サブシステム環境から分離したい場合は、z/OS 環境からタイプ 4 コネクティビティーを使用します。

タイプ 2 IMS Universal ドライバー を使用したローカル・コネクティビティー

タイプ 2 IMS Universal ドライバー を使用したローカル・コネクティビティーは、z/OS プラットフォームおよび実行時環境をターゲットにしています。同じ論理区画 (LPAR) 内の IMS サブシステムに接続するには、タイプ 2 コネクティビティーを使用します。

次の表は、タイプ 2 IMS Universal ドライバー のクライアント・アプリケーションをサポートする z/OS 実行時環境を示しています。

表 97. タイプ 2 IMS Universal ドライバー の z/OS 実行時環境サポート

z/OS 実行時環境	タイプ 2 IMS Universal ドライバーのサポート
WebSphere Application Server for z/OS	• IMS Universal Database リソース・アダプター
IMS Java 従属領域 (JMP 領域および JBP 領域)、CICS	• IMS Universal DL/I ドライバー • IMS Universal JDBC ドライバー

これは IMS サブシステムと同じ LPAR 上で実行するため、接続時に、タイプ 2 IMS Universal ドライバー のクライアント・アプリケーションは、IP アドレス、ポート番号、ユーザー ID、またはパスワードを指定する必要がありません。 **driverType** プロパティーは、IMS に対するローカル (タイプ 2) コネクティビティーを指示するように設定されている必要があります。

次の図は、タイプ 2 の IMS Universal ドライバーが、z/OS メインフレーム環境内部の LPAR で実行されている Java クライアント・アプリケーションと、同じ LPAR 内に配置されている IMS サブシステムとの間で通信経路を確立する方法を示しています。

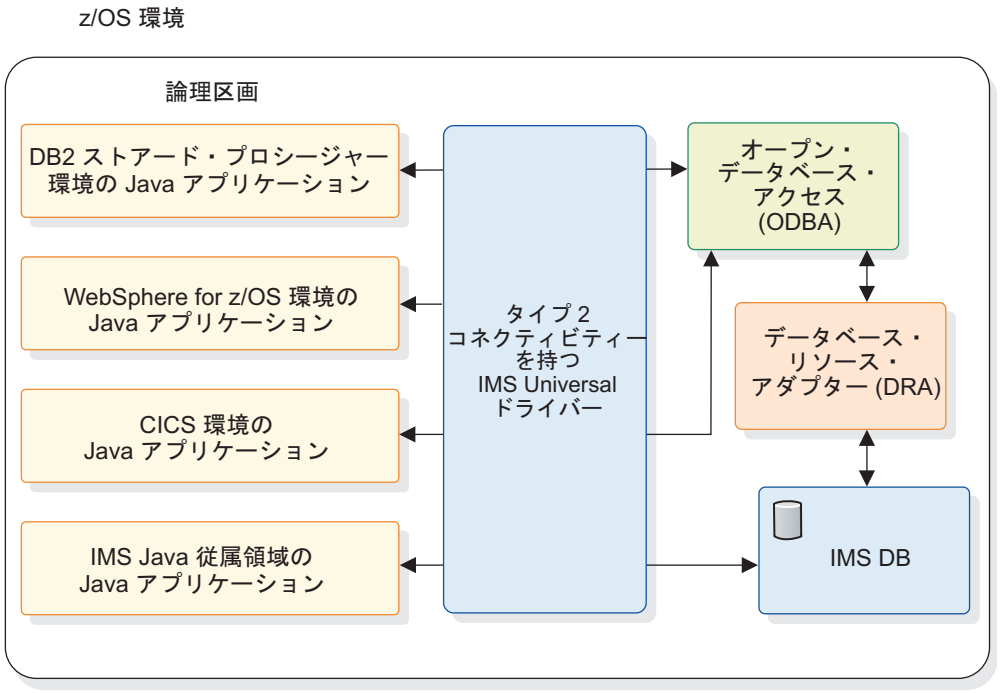


図 107. タイプ 2 IMS Universal ドライバー を使用したローカル・コネクティビティー

RRSLocalOption コネクティビティー・タイプ

タイプ 4 コネクティビティーおよびタイプ 2 コネクティビティーに加えて、RRSLocalOption コネクティビティー・タイプは、WebSphere Application Server for z/OS で実行する IMS Universal Database リソース・アダプターによってサポートされます。 RRSLocalOption コネクティビティーでは、IMS Universal Database リソース・アダプターを使用するアプリケーションは、コミット呼び出し

およびロールバック呼び出しを発行しません。代わりに、トランザクション処理は WebSphere Application Server for z/OS によって管理されます。2 フェーズ・コミット (XA) トランザクション処理は、RRSLocalOption コネクティビティー・タイプではサポートされていません。

関連概念:

IMS DB へのアクセスに対する IMS Connect のサポート (コミュニケーションおよびコネクション)

CSL ODBM の管理 (システム管理)

IMS にアクセスする IMS Universal ドライバー・プログラミング方式の比較

ご使用の IT インフラストラクチャー、ソリューション・アーキテクチャー、およびアプリケーション設計に応じて、開発シナリオに最も適した IMS Universal ドライバー・プログラミング方式を選択します。

以下の表では、アプリケーション・プログラマーが選択したアプリケーション・プラットフォーム、データ・アクセス方式、およびトランザクション処理に基づいて、使用が推奨される IMS Universal ドライバー・プログラミング方法をリストしています。

表 *98. IMS* にアクセスするプログラミング方式の比較

アプリケーション・プラットフォーム	データ・アクセス方式	必要なトランザクション処理	推奨される方式
分散プラットフォーム用 WebSphere Application Server または WebSphere Application Server for z/OS	SQL または DL/I データ操作を実行する CCI プログラミング・インターフェース。	ローカル・トランザクション処理のみ。	ローカル・トランザクション・サポート (imsudbLocal.rar) がある IMS Universal Database リソース・アダプターを使用して、SQLInteractionSpec クラスを指定した SQL 呼び出し、または DLIInteractionSpec クラスを指定した DL/I 呼び出しを実行する。
	SQL または DL/I データ操作を実行する CCI プログラミング・インターフェース。	2 フェーズ (XA) コミット処理[1] またはローカル・トランザクション処理。	XA トランザクション・サポート (imsudbXA.rar) がある IMS Universal Database リソース・アダプターを使用して、SQLInteractionSpec クラスを指定した SQL 呼び出し、または DLIInteractionSpec クラスを指定した DL/I 呼び出しを実行する。
	SQL データ操作を実行する JDBC プログラミング・インターフェース。	ローカル・トランザクション処理のみ。	ローカル・トランザクション・サポート (imsudbJLocal.rar) がある IMS Universal JCA/JDBC ドライバー・バージョンの IMS Universal Database リソース・アダプターを使用し、JDBC API を使用して SQL 呼び出しを実行する。
	SQL データ操作を実行する JDBC プログラミング・インターフェース。	2 フェーズ (XA) コミット処理[1] またはローカル・トランザクション処理。	XA トランザクション・サポート (imsudbJXA.rar) がある IMS Universal JCA/JDBC ドライバー・バージョンの IMS Universal Database リソース・アダプターを使用し、JDBC API を使用して SQL 呼び出しを実行する。
分散プラットフォームまたは z/OS プラットフォームにあるスタンドアロン Java アプリケーション (Java EE アプリケーション・サーバーの外部)。	SQL データ操作を実行する JDBC プログラミング・インターフェース。	2 フェーズ (XA) コミット処理[2] またはローカル・トランザクション処理。	IMS Universal JDBC ドライバー (imsudb.jar[3]) を使用し、JDBC API を使用して SQL 呼び出しを実行する。
	データ操作を実行する従来の DL/I プログラミング・セマンティクス。	2 フェーズ (XA) コミット処理[2] またはローカル・トランザクション処理。	IMS Universal DL/I ドライバー (imsudb.jar[3]) を使用し、PCB クラスを使用して DL/I 呼び出しを実行する。
分散プラットフォームまたは z/OS プラットフォームにあるスタンドアロン非 Java アプリケーション。	DRDA プロトコルを使用するデータ・アクセス。	2 フェーズ (XA) コミット処理またはローカル・トランザクション処理。	選択したプログラミング言語を使用して、IMS Connect に対して DDM コマンドを発行する。 2 フェーズ・コミット・メカニズムの実装について責任を持つのは、アプリケーション・プログラマーです。

注:

1. XA トランザクション・サポートは、タイプ 4 コネクティビティーの場合のみ使用可能です。
2. ローカル・トランザクションおよび XA トランザクションに対してドライバーが使用可能となっていますが、2 フェーズ・コミット・メカニズムの実装についての責任を持つのはアプリケーション・プログラマーです。 XA トランザクション・サポートは、タイプ 4 コネクティビティーの場合のみ使用可能です。
3. JDBC ドライバーと DL/I ドライバーは、同じ .jar ファイル内で提供されます。JDBC ドライバーを介して作成される SQL 呼び出しは、DL/I ドライバー内のメソッドとクラスを使用して、DL/I 呼び出しに変換されるからです。

IMS Universal ドライバーによる可変長データベース・セグメントのサポート

IMS Universal データベース・リソース・アダプターおよび IMS Universal JDBC ドライバーは、クライアント・アプリケーション・プログラムに代わって可変長セグメントを管理します。IMS Universal DL/I ドライバーを使用するアプリケーション・プログラムでは、可変長セグメントの LL フィールドのデータを管理する必要があります。

IMS Universal データベース・リソース・アダプターおよび IMS Universal JDBC ドライバーでの LL フィールドの使用

デフォルトでは、可変長セグメントの LL フィールドは、SQL 照会の可視列として返されません。LL フィールドが要求されない場合、IMS Universal データベース・リソース・アダプターおよび IMS Universal JDBC ドライバーがアプリケーション・プログラムに代わって LL フィールドを管理します。アプリケーション・プログラムが IMS へのデータ接続を作成するときに、明示的に列を要求しない限りは、いかなるタイプの照会 (`SELECT *` を含む) も LL フィールドのデータにアクセスできません。

LL フィールドをアプリケーション・レベルで管理し、アプリケーションで IMS Universal データベース・リソース・アダプターまたは IMS Universal JDBC ドライバーを使用する場合は、llField プロパティーを `true` に設定して、LL フィールドのデータを明示的に要求する必要があります。

アプリケーションでは、以下のいずれかのインターフェースの標準プロパティー・リストにある llField プロパティーを `true` に設定できます。

`java.sql.DriverManager.getConnection(String` *`url`*`, Properties` *`properties`*`)`

`com.ibm.ims.jdbc.IMSDataSource.setProperties(Properties` *`properties`*`)`

llField=true プロパティーが設定されている場合、すべての操作で、LL フィールドは標準 SQL 結果セットの通常列として表示されます。ユーザーは、LL フィールド・データの読み取り、挿入、更新を直接行うことができます。LL フィールドのデータを削除すると、関連のデータベース・レコードのその他の部分も削除されます。フィールドを NULL 状態に設定するには、セグメント (LL フィールド列の値) の長さを、セグメント内のフィールドのオフセットよりも小さい値に設定します。

LL フィールドの長さは 2 バイトで、BINARY、SHORT、または USHORT データとして処理する必要があります。

java.sql.ResultSet.wasNull メソッドを使用することで、LL データを調べずに、可変長セグメントのインスタンスに NULL 可能フィールドがあるかどうかを判別することもできます。

IMS Universal DL/I ドライバーを使用した NULL フィールドのインスタンスの検査

IMS Universal DL/I ドライバーを使用するアプリケーションは、常に可変長セグメントの LL フィールドのデータを受け取ります。次の 2 つの方法のいずれかで、セグメント・インスタンス内のフィールドが NULL であるかどうかを判別することができます。LL フィールドのデータをフィールドのオフセットと比較する方法、または com.ibm.ims.dli.Path.wasNull() メソッドを使用する方法です。

com.ibm.ims.dli.Path.wasNull() メソッドは、読み取られた最後のフィールドが NULL 状態の場合、ブール値を返します。フィールドが NULL の場合、戻り値は `true` です。フィールドが NULL であるかどうかを判別するには、wasNull() メソッドを呼び出す前にフィールドを読み取る必要があります。

関連概念:

- 可変長セグメント (データベース管理)

関連タスク:

- 可変長セグメントの指定方法 (データベース管理)

複合構造のフラット化のサポート

flattenTables 接続プロパティーを使用すると、データベース表がフラット化されたビューで生成されます。IMS カタログ内のコピーブック構造は変更されませんが、その特定の接続についての表の構造に関する情報は変更されます。 flattenTables 接続プロパティーを使用可能にすると、データベース表の照会プロセスを単純化できます。

flattenTables 接続プロパティーに従って表示される複合構造は、以下の命名規則に従ってフォーマット設定されます。

- 静的配列は、配列の名前、配列の索引、およびフィールドの名前によって参照されます。

  この規則に従った例として `CLASSES_2_INST` があります。この場合、`CLASSES` は配列の名前、`2` は配列の索引、`INST` はフィールドの名前です。
- フラット化された構造のビューには構造のサブエレメントのみが表示されます (構造自体の名前は表示されません)。

制約事項: flattenTables 接続プロパティーでは、静的配列と静的構造のみがサポートされます。動的配列は変更されません。

次のビューには、flattenTables 接続プロパティーを使用不可にした場合のコピーブック構造が表示されています。

```
01 SEGM.
   05 CLASSES OCCURS 4 TIMES.
      10 INST PIC X(15).
      10 GRADE PIC X(1).
      10 BOOKS OCCURS 2 TIMES.
```

```
        15 AUTHOR.
           20 FIRSTNAME PIC X(10).
           20 LASTNAME PIC X(15).
        15 TITLE PIC X(20).
```

次のビューは前の例と同じコピーブック構造を表示していますが、この場合は flattenTables 接続プロパティーを使用可能にしたものです。

```
01 SEGM.
   05 CLASSES_1_INST PIC X(15).
   05 CLASSES_1_GRADE PIC X(1).
   05 CLASSES_1_BOOKS_1_FIRSTNAME PIC X(10).
   05 CLASSES_1_BOOKS_1_LASTNAME PIC X(15).
   05 CLASSES_1_BOOKS_1_TITLE PIC X(20).
   05 CLASSES_1_BOOKS_2_FIRSTNAME PIC X(10).
   05 CLASSES_1_BOOKS_2_LASTNAME PIC X(15).
   05 CLASSES_1_BOOKS_2_TITLE PIC X(20).
   05 CLASSES_2_INST PIC X(15).
   05 CLASSES_2_GRADE PIC X(1).
   05 CLASSES_2_BOOKS_1_FIRSTNAME PIC X(10).
   05 CLASSES_2_BOOKS_1_LASTNAME PIC X(15).
   05 CLASSES_2_BOOKS_1_TITLE PIC X(20).
   05 CLASSES_2_BOOKS_2_FIRSTNAME PIC X(10).
   05 CLASSES_2_BOOKS_2_LASTNAME PIC X(15).
   05 CLASSES_2_BOOKS_2_TITLE PIC X(20).
   05 CLASSES_3_INST PIC X(15).
   05 CLASSES_3_GRADE PIC X(1).
   05 CLASSES_3_BOOKS_1_FIRSTNAME PIC X(10).
   05 CLASSES_3_BOOKS_1_LASTNAME PIC X(15).
   05 CLASSES_3_BOOKS_1_TITLE PIC X(20).
   05 CLASSES_3_BOOKS_2_FIRSTNAME PIC X(10).
   05 CLASSES_3_BOOKS_2_LASTNAME PIC X(15).
   05 CLASSES_3_BOOKS_2_TITLE PIC X(20).
   05 CLASSES_4_INST PIC X(15).
   05 CLASSES_4_GRADE PIC X(1).
   05 CLASSES_4_BOOKS_1_FIRSTNAME PIC X(10).
   05 CLASSES_4_BOOKS_1_LASTNAME PIC X(15).
   05 CLASSES_4_BOOKS_1_TITLE PIC X(20).
   05 CLASSES_4_BOOKS_2_FIRSTNAME PIC X(10).
   05 CLASSES_4_BOOKS_2_LASTNAME PIC X(15).
   05 CLASSES_4_BOOKS_2_TITLE PIC X(20).
```

関連タスク:

719 ページの『管理対象環境での IMS Universal Database リソース・アダプターを使用した接続』

725 ページの『管理対象環境での IMS Universal JCA/JDBC ドライバーを使用した接続』

746 ページの『JDBC DataSource インターフェースを使用した IMS データベースへの接続』

753 ページの『JDBC DriverManager インターフェースを使用した IMS データベースへの接続』

実行時 Java メタデータ・クラスの生成

IMS Universal ドライバーを使用して IMS データベースに接続するには、データベース・ビューを提供する Java メタデータ・クラスを Java クラスパスに組み込む必要があります。

注: IMS カタログを使用する場合は、IMS Universal ドライバー・アプリケーション・プログラムは、必要なメタデータをカタログ・データベースから直接取得できるので、Java メタデータ・クラス・ファイルは不要です。

Java メタデータ・クラスは、プログラム仕様ブロック (PSB) およびその関連するプログラム制御ブロック (PCB) によって指定されるアプリケーション・ビュー情報を表します。 Java メタデータ・クラスは、PSB で定義されるセグメントおよびフィールドへの 1 対 1 マッピングを提供します。

Java メタデータ・クラスは、コンパイルして、PSB を使用して IMS データにアクセスしようとする任意の Java アプリケーションのクラスパスから使用可能にしておく必要があります。

データベース接続のセットアップ中に、このメタデータ・クラスの名前をリソース・アダプターまたは JDBC ドライバーに渡します。 Java メタデータ・クラスは実行時に、IMS Universal ドライバーによって使用され、SQL および Java ベース DL/I の両方の呼び出しを処理します。

IMS Explorer for Developmentによって作成されるデータベース・メタデータ・クラスのデフォルトのセグメント・エンコードは、`cp1047` です。このセグメント・エンコードを変更するには、com.ibm.ims.base.DLIBaseSegment.setDefaultEncoding メソッドを使用します。

関連概念:

IMS Explorer for Development の概要

病院データベースの例

IMS Universal ドライバー・アプリケーションのプログラミング・トピックのサンプル・コードでは、病院データベースを使用します。

次の図は、病院データベースにおけるセグメントの階層構造を示しています。

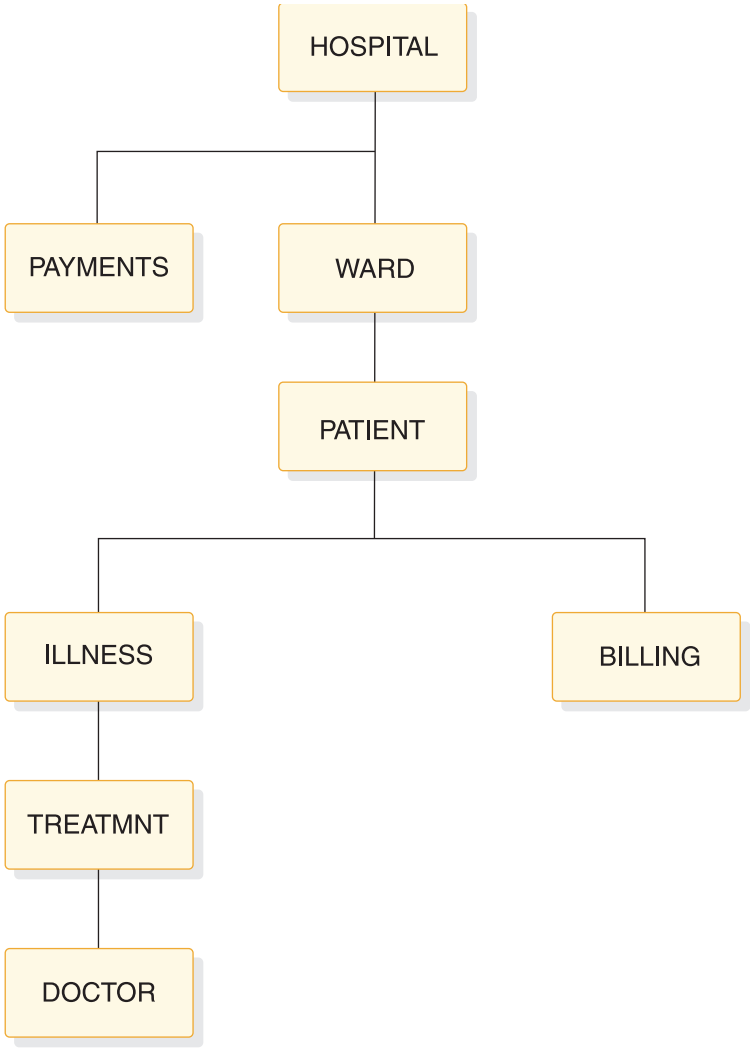


図 *108.* 病院データベースのセグメント

図の各ノードはセグメントを表します。

- HOSPITAL セグメントは、データベースのルート・セグメントです。
- PAYMENTS および WARD は、HOSPITAL セグメントの子セグメントです。
- WARD には PATIENT という名前の直接の下層セグメントがあります。
- ILLNESS および BILLING は、PATIENT の子セグメントです。
- ILLNESS には、患者の治療に関する詳細を保管する TREATMENT という名前の子セグメントがあります。
- ILLNESS、DOCTOR の子セグメントは、データベース階層で最下位のレベルのセグメントです。

次の表は、病院データベースの各セグメントのレイアウトを示しています。

HOSPITAL セグメント

次の表は HOSPITAL セグメントを示しており、以下の 2 つのフィールドがあります。

- 病院コード (HOSPCODE)

- 病院名 (HOSPNAME)

HOSPCODE は、ユニーク・キー・フィールドです。

フィールド名	フィールド長 (バイト)
HOSPCODE	12
HOSPNAME	17

PAYMENTS セグメント

次の表は PAYMENTS セグメントを示しており、以下の 2 つのフィールドがあります。

- 患者番号 (PATNUM)
- 支払額 (AMOUNT)

フィールド名	フィールド長 (バイト)
PATNUM	4
AMOUNT	8

WARD セグメント

次の表は WARD セグメントを示しており、以下の 5 つのフィールドがあります。

- 病棟番号 (WARDNO)
- 病棟名 (WARDNAME)
- 患者数 (PATCOUNT)
- 看護士数 (NURCOUNT)
- 医師数 (DOCCOUNT)

WARDNO は、ユニーク・キー・フィールドです。

フィールド名	フィールド長 (バイト)
WARDNO	2
WARDNAME	4
PATCOUNT	8
NURCOUNT	4
DOCCOUNT	2

PATIENT セグメント

次の表は PATIENT セグメントを示しており、以下の 2 つのフィールドがあります。

- 患者番号 (PATNUM)
- 患者名 (PATNAME)

PATNUM は、ユニーク・キー・フィールドです。

フィールド名	フィールド長 (バイト)
PATNUM	12
PATNAME	17

ILLNESS セグメント

次の表は ILLNESS セグメントを示しており、以下の 1 つのフィールドがあります。

- 病名 (ILLNAME)

フィールド名	フィールド長 (バイト)
ILLNAME	15

TREATMNT セグメント

次の表は TREATMNT セグメントを示しており、以下の 3 つのフィールドがあります。

- 治療日 (TREATDAY)
- 治療のタイプ (TREATMNT)
- 治療コメント (COMMENTS)

フィールド名	フィールド長 (バイト)
TREATDAY	8
TREATMNT	15
COMMENTS	10

DOCTOR セグメント

次の表は DOCTOR セグメントを示しており、以下の 2 つのフィールドがあります。

- 医師番号 (DOCTNO)
- 医師名 (DOCNAME)

フィールド名	フィールド長 (バイト)
DOCTNO	4
DOCNAME	20

BILLING セグメント

次の表は BILLING セグメントを示しており、以下の 2 つのフィールドがあります。

- 請求額 (AMOUNT)
- 請求コメント (COMMENTS)

フィールド名	フィールド長 (バイト)
AMOUNT	8
COMMENTS	20

関連概念:

799 ページの『SSAList インターフェースを使用したセグメント検索引数の指定』

IMS Universal Database リソース・アダプターを使用したプログラミング

ここでは、IMS Universal Database リソース・アダプターを使用して IMS データベースにアクセスするプログラムの作成方法について説明します。

IMS Universal Database リソース・アダプターの概要

IMS Universal Database リソース・アダプターは、Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) 1.6 標準をベースとしています。JCA の目的は、IMS などのエンタープライズ情報システム (EIS) を Java EE プラットフォームに接続することです。 JCA は、Java EE アプリケーション・サーバーによって管理される多数のサービスを提供します。これらのサービスには、セキュリティー資格情報管理、接続プール、およびトランザクション管理が含まれます。

これらのサービスは、IMS Universal Database リソース・アダプターと Java EE アプリケーション・サーバー間のシステム・レベルの契約によって提供されるため、アプリケーション・プログラマーによる追加のコーディングは必要ありません。

JCA 仕様は、Common Client Interface (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ています。このインターフェースを使用すれば、どの EIS とも通信できます。 IMS Universal Database リソース・アダプターは、IMS データベースとの対話用の CCI を実装しています。 IMS Universal Database リソース・アダプター用の CCI インターフェースは、com.ibm.ims.db.cci パッケージの中にあります。 IMS が提供する CCI 実装により、アプリケーションは SQL 呼び出しまたは DL/I 呼び出しのいずれかを出して、IMS データベースにアクセスできます。

IMS Universal Database リソース・アダプターによって提供される CCI インターフェースの他に、JDBC アプリケーションを作成して、アプリケーション・サーバーによって提供される Java EE サービスを利用しながら、管理対象環境から IMS データにアクセスすることもできます。この機能は、IMS Universal JCA/JDBC ドライバー・バージョンの IMS Universal Database リソース・アダプターによって提供されます。 IMS Universal JCA/JDBC ドライバーは、Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) 1.6 および Java Database Connectivity (JDBC) 4.0 標準をベースとしています。

IMS Universal Database リソース・アダプターは、IMS にアクセスするための TCP/IP エンドポイントとして IMS Connect と通信します。

IMS Universal ドライバーを使用した Java アプリケーションを作成する準備

IMS Universal ドライバーを使用する Java アプリケーション・プログラムでは、Java Development Kit (JDK) 7.0 が必要です。JMP 領域および JBP 領域で実行される Java プログラムは、JDK 7.0 以降が必要です。IMS Universal ドライバーを使用する Java アプリケーション・プログラムは、IMS データベースと対話するために、データベース・メタデータへのアクセス権限を持っている必要があります。IMS カタログ・データベースで直接このメタデータにアクセスするか、IMS Enterprise Suite Explorer for Development を使用して Java メタデータ・クラスとしてこのメタデータを生成することができます。

IMS Universal Database リソース・アダプターによりサポートされるトランザクション・タイプおよびプログラミング・インターフェース

IMS Universal Database リソース・アダプターでは、最適化されたトランザクション管理およびパフォーマンスのための 4 つのタイプのサポートを提供しています。

IMS Universal Database リソース・アダプターによって提供されるトランザクション・サポートのタイプは、次のとおりです。

ローカル・トランザクション・サポートがある **IMS Universal Database** リソース・アダプター **(imsudbLocal.rar)**
: このリソース・アダプターは、サポートされる任意の Java EE アプリケーション・サーバーに配置するときの、CCI プログラミング・インターフェースおよび LocalTransaction サポートを提供します。

XA トランザクション・サポートがある **IMS Universal Database** リソース・アダプター **(imsudbXA.rar)**
: このリソース・アダプターは、サポートされる任意の Java EE アプリケーション・サーバーに配置するときの、CCI プログラミング・インターフェースおよび XATransaction サポートと LocalTransaction サポートの両方を提供します。

ローカル・トランザクション・サポートがある **IMS Universal JCA/JDBC** ドライバー **(imsudbJLocal.rar)**
: このリソース・アダプターは、サポートされる任意の Java EE アプリケーション・サーバーに配置するときの、JDBC プログラミング・インターフェースおよび LocalTransaction サポートを提供します。

XA トランザクション・サポートがある **IMS Universal JCA/JDBC** ドライバー **(imsudbJXA.rar)**
: このリソース・アダプターは、サポートされる任意の Java EE アプリケーション・サーバーに配置するときの、JDBC プログラミング・インターフェースおよび XATransaction サポートと LocalTransaction サポートの両方を提供します。

制約事項**:** XA トランザクション・サポートは、タイプ 4 コネクティビティーの場合のみ使用可能です。

グローバルまたは 2 フェーズ・コミット・トランザクション処理の場合は、XA トランザクション・サポートがある IMS Universal Database リソース・アダプターを使用します。単一フェーズ・コミット機能の場合は、XA トランザクション・サポートまたはローカル・トランザクション・サポートのいずれかがある IMS Universal Database リソース・アダプターを使用します。

Java EE アプリケーションにさまざまなトランザクション・サービス品質を提供するため、同じ Java EE アプリケーション・サーバーに、複数の異なるタイプの IMS Universal Database リソース・アダプターを配置できます。

IMS Universal Database リソース・アダプターを使用して IMS データベースと複数の対話を実行する場合、すべて成功するかまたはすべて失敗するかのいずれかになるように、すべてのアクションを 1 つにまとめてグループ化できます。これは、コンテナー管理トランザクション区分または Bean 管理トランザクション区分を使用して実行できます。

コンテナー管理トランザクション では、EJB メソッド呼び出しで実行されたすべての作業は 1 つの作業単位の一部であり、アプリケーションによる明示的区分は必要ありません。トランザクションの保全性は、Java EE アプリケーション・サーバーによって管理されます。

Bean 管理トランザクション では、javax.resource.cci.LocalTransaction または javax.transaction.UserTransaction インターフェースを使用して、明示的に作業単位をプログラマチックに区分する必要があります。 LocalTransaction インターフェースを使用する Bean 管理トランザクションは、リソース・アダプターを介してのみ、実行される作業をグループ化できます。UserTransaction インターフェースを使用して、アプリケーション内のすべてのトランザクション・リソースをグループ化できます。同じ EJB メソッド呼び出し内に複数の作業単位が必要である場合は、Bean 管理 EJB を使用します。

タイプ 2 IMS Universal Database リソース・アダプターを使用する場合、driverType 接続プロパティーに 2 を指定すると、Bean 管理トランザクションまたは JDBC 接続インターフェースに javax.resource.cci.LocalTransaction を使用することができます。driverType 接続プロパティーに 2_CTX を指定した場合は、明示的なコミットおよびロールバック呼び出しを発行するアプリケーション・プログラムに javax.transaction.UserTransaction を使用することができます。

IMS Universal Database リソース・アダプターによってサポートされるソフトウェア構成

IMS Universal Database リソース・アダプターには、サポートされるソフトウェア構成に関していくつかの要件と制限があります。

次の表に、サポートされるソフトウェア構成をリストします。

表 99. *IMS Universal Database* リソース・アダプターでサポートされるソフトウェア構成

IMS Universal Database リソース・アダプターのバージョン	サポートされる **IBM WebSphere Application Server** のバージョン	**IBM** 以外の **Java EE** アプリケーション・サーバー
バージョン 14	• バージョン 8.5 以降 • バージョン 8.5 以降 (IBM WebSphere Liberty サーバーの場合)	任意の汎用 Java EE 1.4 以降の認定済みアプリケーション・サーバー[12]
バージョン 13	バージョン 7.0.1 以降	任意の汎用 Java EE 1.4 以降の認定済みアプリケーション・サーバー[12]
バージョン 12	バージョン 6.1 以降	任意の汎用 Java EE 1.4 以降の認定済みアプリケーション・サーバー[12]

1. 汎用 Java EE アプリケーション・サーバーの場合、IMS Universal Database リソース・アダプターに対する IBM のサポートは、以下のいずれかの基準を満たす特定の機能に制限されます。
   - WebSphere Application Server または WebSphere Application Server Liberty Profile のサポート対象バージョンで問題を再現できる。
   - 診断トレースにより、問題の原因が IMS Universal Database リソース・アダプターであることが確認されている。
2. 汎用 Java EE アプリケーション・サーバーの場合は、以下の機能はサポートされません。
   - 2 フェーズ (XA) コミット処理。
   - z/OS 上の汎用 Java EE アプリケーション・サーバー
   - JDBC プログラミング・インターフェースを使用するロケーション・トランザクション (imsudbJLocal.rar) での、IMS Universal Database リソース・アダプター以外のデータベース・リソース・アダプター。

IMS Universal Database リソース・アダプターを使用した IMS への接続

IMS Universal Database リソース・アダプターは、Java EE 管理対象環境から IMS データベースへのコネクティビティーを提供します。

Common Client Interface (CCI) は、エンタープライズ情報システム (EIS) との接続を確立するための ConnectionFactory インターフェースおよび Connection インターフェースを提供します。 IMS Universal Database リソース・アダプターで CCI プログラミング・インターフェースを使用する場合、Java アプリケーション・コンポーネントは Java Naming and Directory Interface (JNDI) を使用して ConnectionFactory インスタンスを参照し、ConnectionFactory インスタンスを使用して IMS データベースへの接続を取得します。

同様に、IMS Universal JCA/JDBC ドライバーで JDBC プログラミング・インターフェースを使用する場合、Java アプリケーション・コンポーネントは JNDI を使用して DataSource インスタンスを参照し、DataSource インスタンスを使用して Connection オブジェクトを取得します。

RRSLocalOption コネクティビティー・タイプ

タイプ 4 コネクティビティーおよびタイプ 2 コネクティビティーに加えて、RRSLocalOption コネクティビティー・タイプは、WebSphere Application Server for z/OS で実行する IMS Universal Database リソース・アダプターによってサポートされます。 RRSLocalOption コネクティビティーでは、IMS Universal Database リソース・アダプターを使用するアプリケーションは、コミット呼び出しおよびロールバック呼び出しを発行しません。代わりに、トランザクション処理は WebSphere Application Server for z/OS によって管理されます。2 フェーズ・コミット (XA) トランザクション処理は、RRSLocalOption コネクティビティー・タイプではサポートされていません。

管理対象環境での IMS Universal Database リソース・アダプターを使用した接続

管理対象 (または 3 層) の環境では、Java EE アプリケーションは、WebSphere Application Server および IMS Universal Database リソース・アダプターなどの Java EE アプリケーションと対話して、IMS データベースと通信します。

WebSphere Application Server で CCI Connection オブジェクトを構成して使用し、IMS データベースにアクセスするには、以下のようにします。

1. 管理コンソールを使用して、IMS Universal Database リソース・アダプターを WebSphere Application Server にデプロイします。
2. 管理コンソールを使用して、WebSphere Application Server に IMS Universal Database リソース・アダプターで使用する接続ファクトリーを作成します。
   a. 接続ファクトリーの名前および Java Naming and Directory Interface (JNDI) 名を指定します。
   b. IMS Universal Database リソース・アダプター用に、以下のカスタム接続プロパティーを設定します。

      DatastoreName

      アクセスする IMS データ・ストアの名前。

      - タイプ 4 コネクティビティーを使用する場合、**DatastoreName** プロパティーは、ODBM に定義されているデータ・ストアの名前と一致したものであるかまたはブランクである必要があります。データ・ストア名は、DATASTORE(NAME=*name*) パラメーターまたは DATASTORE(NAME=*name*, ALIAS (NAME=*aliasname*)) パラメーターのいずれかを使用して、ODBM CSLDCxxx PROCLIB メンバー内で定義されます。別名が指定される場合、**datastoreName** プロパティーの値として *aliasname* を指定する必要があります。 **DatastoreName** の値が空白 (または指定されていない) 場合、ODBM に定義されているす

べてのデータ・ストアでデータ共用が使用可能となっていると見なされるため、IMS Connect は使用可能な任意の ODBM のインスタンスに接続します。

- タイプ 2 コネクティビティーを使用する場合は、**DatastoreName** プロパティーを IMS サブシステムの別名に設定します。Java 従属領域のランタイムの場合は、これは設定する必要はありません。

DatabaseName

ターゲット IMS データベースを表すデータベース・メタデータの場所。

メタデータが IMS カタログに保管されているのか、IMS Enterprise Suite Explorer for Development によって生成された静的メタデータ・クラスとして保管されているのかに応じて、**DatabaseName** プロパティーを以下の 2 つの方法のいずれかで指定することができます。

- IMS システムで IMS カタログを使用している場合、**DatabaseName** プロパティーは、アプリケーションがターゲット IMS データベースへのアクセスに使用する PSB の名前です。
- IMS Explorer for Development を使用している場合、**databaseName** プロパティーは、IMS Explorer for Development によって生成された Java メタデータ・クラスの完全修飾名です。URL には、`class://` の接頭部を付ける必要があります (例えば、`class://com.foo.BMP255DatabaseView`)。

J2C 接続ファクトリー環境では、リソース・アダプターに指定されたデフォルト値に影響を与えずに、個別の接続の **DatabaseName** プロパティーをオーバーライドできます。

MetadataURL

ターゲット IMS データベースを表すデータベース・メタデータの場所。

このプロパティーは推奨されていません。代わりに **DatabaseName** を使用してください。

MetadataURL プロパティーは、IMS Enterprise Suite Explorer for Developmentによって生成された Java メタデータ・クラスの完全修飾名です。 URL には、`class://` の接頭部を付ける必要があります (例えば、`class://com.foo.BMP255DatabaseView`)。

J2C 接続ファクトリー環境では、リソース・アダプターに指定されたデフォルト値に影響を与えずに、個別の接続の **MetadataURL** プロパティーをオーバーライドできます。

PortNumber

IMS Connect と通信するために使用される TCP/IP サーバーのポート番号。ポート番号は、IMS Connect 構成 PROCLIB メンバーの ODACCESS ステートメントで DRDAPORT パラメーターを使

用して定義されます。デフォルトのポート番号は 8888 です。タイプ 2 コネクティビティーの使用時にはこのプロパティーを設定しないでください。

DatastoreServer
データ・ストア・サーバー (IMS Connect) の名前または IP アドレス。ホスト名 (例えば、dev123.svl.ibm.com) または IP アドレス (例えば、192.166.0.2) のいずれかを指定できます。タイプ 2 コネクティビティーの使用時にはこのプロパティーを設定しないでください。

DriverType
使用するドライバー・コネクティビティーのタイプ。 **DriverType** の値は、タイプ 4 コネクティビティーに対しては「4」、タイプ 2 コネクティビティーに対しては「2」である必要があります。ドライバーが WebSphere Application Server for z/OS で実行する場合、RRSLocalOption コネクティビティーに対しては、**DriverType** の値を「2_CTX」に設定することもできます。

user RACF 管理者によって提供された IMS Connect に接続するためのユーザー名。タイプ 2 コネクティビティーの使用時にはこのプロパティーを設定しないでください。

password
RACF 管理者によって提供された IMS Connect に接続するためのパスワード。タイプ 2 コネクティビティーの使用時にはこのプロパティーを設定しないでください。

allMetadata

オプション。このプロパティーが `true` に設定された場合、DatabaseMetadata インターフェースは、IMS カタログにあるすべてのリソースに関する情報を返します。このプロパティーが `false` に設定された場合、DatabaseMetadata インターフェースは、割り振り済み PSB に関する情報を返します。このプロパティーのデフォルト値は `false` です。

signedCompare
オプション。このプロパティーが「true」に設定された場合、署名されたデータ・タイプについての範囲が指定された照会をサポートするために特殊 SSA が生成されます。プロパティーが「false」に設定された場合は、データ・タイプ値のバイナリー表現に基づいて標準バイナリー比較が実行されます。この値を「false」に設定すると、パフォーマンスは上がりますが、誤った結果がもたらされる可能性があります。このプロパティーのデフォルト値は「true」です。

flattenTables
オプション。データベース表をフラット化されたビューで生成するかどうかを示します。値を true にすると、表の追加の列として STRUCT または ARRAY のサブエレメントが公開されます。デフォルト値は false です。

- IMS Explorer は、コピーブックをインポートする際にコピーブックの構造をフラット化します。IMS カタログ内ではコピーブック自体は変更されないままですが、その特定の接続についての各表の構造に関する情報は変更されます。
- flattenTables プロパティーを指定すると、複合構造内のフィールドを直接照会できます。複合構造のフラット化のサポートについて詳しくは、 709 ページの『複合構造のフラット化のサポート』を参照してください。

制約事項**:** flattenTables 接続プロパティーでは、静的配列と静的構造のみがサポートされます。動的配列は変更されません。

sslKeyStoreType
オプション。セキュア・ソケット接続の確立に必要な暗号オブジェクトを含むファイルの形式を指定します。有効な値は、「JKS」および「PKCS12」です。この値は、**sslConnection** が「true」に設定されていて、**sslKeyStoreType** が指定されていない場合にのみ使用します。 **sslKeyStoreType** パラメーターのデフォルトは「JKS」です。

sslSecureSocketProtocol
オプション。新しい接続の暗号通信プロトコルを指定します。サーバーがサポートしているプロトコルで、最高レベルのセキュリティーを提供するプロトコルを指定します。有効な値は、「SSL」、「SSLv3」、「TLSv1.1」、および「TLSv1.2」です。この値は、**sslConnection** が「true」に設定されている場合のみ使用します。**sslConnection** が「true」に設定されていて、**sslSecureSocketProtocol** が指定されていない場合、デフォルト・プロトコルは JRE とサーバーによって実行時に決定されます。

t2OutputBufferSize
オプション。タイプ 2 接続の場合の SELECT 操作からの結果をバイト単位で表した出力バッファーのサイズ。

t2OutputBufferSize の最小値は 500000 です。500000 より小さい値が設定された場合、このプロパティー値は 500000 に調整されます。最大限界はありません。デフォルト値は 1280000 です。

treatInvalidDecimalAsNull
オプション。Java アプリケーションで無効と見なされる特定の 10 進値 (無効な符号ビットが設定された PACKEDDECIMAL および ZONEDDECIMAL など) をヌルとして解釈するかどうかを指示します。デフォルトでは、このプロパティーは「false」であるため、Java アプリケーションが無効値を処理しているときには変換例外がスローされます。

currentSchema
オプション。動的に準備される SQL ステートメントで非修飾データベース・オブジェクトを修飾するために使用される、デフォルトのスキーマ名を指定します。

dbViewLocation
オプション。databaseView メタデータ・クラスへの絶対パスを指

定します。このプロパティーを使用して、プロジェクト・パスに配置されていないメタデータ・クラスを組み込むことができます。

dpsbOnCommit
オプション。コミットの発生時に PSB の割り振りを解除するには、このプロパティーを **true** に設定します。

推奨事項: 統合接続プールを使用する管理対象環境の場合を除き、このプロパティーを **true** に設定しないでください。

fetchSize
オプション。さらに多くの行が必要な場合に、データベースから取得する行数に関するヒントをクライアントに示します。このプロパティーに指定された数値は、現行接続でリトリーブされたデータのみに影響します。指定された値が 0 の場合、利用可能なすべての行が返されます。

管理対象接続および管理対象外接続のいずれも、このプロパティーのデフォルト値は 0 です。

llField
オプション。このプロパティーを **true** に設定すると、LL フィールドのデータは結果セットに通常の列として表示されます。LL フィールドの値を変更して、可変長セグメントのインスタンス長を変更できます。

maxRows
オプション。照会結果セットで返す最大行数を指定します。デフォルト値は 0 で、結果セットで利用可能なすべての行を返します。

traceFile
オプション。接続するトレース・ファイルの名前を指定します。

traceFileAppend
オプション。指定のトレース・ファイルが存在する場合、このプロパティーを true に設定すると、既存のトレース・ファイルを上書きせずに、そのトレース・ファイルに新規接続のトレース・データを追加する必要があることを指定します。

traceFile に値が指定されていない場合、このプロパティーは無視されます。

traceDirectory
オプション。トレース・ファイルが入っているファイル・システム・ディレクトリーを指定します。デフォルトでは、このパスはアプリケーションを実行するディレクトリーです。

traceFile に値が指定されていない場合、このプロパティーは無視されます。

traceLevel
オプション。接続に使用可能なトレースを指定します。このプロパティーの有効値は、IMSDataSource クラスの Java API 文書で定義されています。

デフォルトでは、すべてのトレースが使用不可になっています。

traceFile に値が指定されていない場合、このプロパティーは無視されます。

トレース・レベル	**traceLevel** パッケージの値	**IMSDataSource** 内の **traceLevel** 定数フィールド	**traceLevel** の **10** 進値
すべて	com.ibm.ims.db.opendb.*	TRACE_ALL	-1
DL/I	com.ibm.ims.db.opendb.dli.*	TRACE_DLI	28
DRDA	com.ibm.ims.db.opendb.drda.*	TRACE_DRDA	1
JDBC	com.ibm.ims.db.opendb.jdbc.*	TRACE_JDBC	32
Java EE	com.ibm.ims.opendb.spi.* com.ibm.ims.db.opendb.cci.*	TRACE_JEE	192

3. Java EE アプリケーションのデプロイメント記述子に、前のステップで作成した接続ファクトリーのリソース参照を追加します。 リソース参照の名前を接続ファクトリーの JNDI 名に設定し、タイプを javax.resource.cci.ConnectionFactory に設定します。
4. Java EE アプリケーションで、初期 JNDI ネーミング・コンテキストを作成し、JNDI 検索を使用して IMS Universal Database リソース・アダプター用の対応する javax.resource.cci.ConnectionFactory インスタンスを取得します。 以下のサンプル・コードは、ConnectionFactory インスタンスを取得するための JNDI 検索の実行方法の例を示しています。ここで、接続ファクトリーの JNDI 名は「imsdblocal」です。

```
Context initctx = new InitialContext();
javax.resource.cci.ConnectionFactory cf =
 (javax.resource.cci.ConnectionFactory)initctx.lookup
 ("java:comp/env/imsdblocal");
```

5. com.ibm.ims.db.cci.IMSConnectionSpec オブジェクトを作成し、必要に応じて、アプリケーション固有のプロパティー値を設定して、IMS データベースにアクセスするため接続ファクトリー・デプロイメント記述子に既に割り当てられている値をオーバーライドします。 以下のサンプル・コードは、IMSConnectionSpec オブジェクトの作成方法を示しています。ここではアプリケーションがユーザー ID およびパスワードの値をオーバーライドする必要があると想定しています。

```
IMSConnectionSpec connSpec = new IMSConnectionSpec();
connSpec.setUser("myUserId");
connSpec.setPassword("myPassword");
```

6. getConnection メソッドを呼び出し、前のステップで作成した IMSConnectionSpec インスタンスに引き渡すことによって、接続ファクトリーから IMS への接続を取得します。 アプリケーションで、接続ファクトリー・デプロイメント記述子に設定されている接続プロパティーをオーバーライドする必要がない場合は、IMSConnectionSpec オブジェクトをインスタンス化する必要はありません。代わりに、アプリケーションは引数を使用しない getConnection メソッドを呼び出すことができます。返される javax.resource.cci.Connection インスタンスは、基礎の物理接続に対するアプリケーション・レベルのハンドルを表しています。

7. 接続を使用して、CCI 対話インターフェースを使用している IMS データベースにアクセスします。 以下のサンプル・コードは、Connection オブジェクトを取得する方法を示しています。

```
javax.resource.cci.Connection conn = cf.getConnection(connSpec);
```

8. Java EE アプリケーションの接続が終了した場合は、Connection インターフェースで close メソッドを使用して接続を閉じます。

管理対象環境で IMS Universal Database リソース・アダプターを使用して IMS データベースに接続するサンプル・コード

以下のサンプル・コードは、Java EE アプリケーションから IMS Universal Database リソース・アダプターを使用して IMS データベースに接続する流れを示しています。

```
//obtain the initial JNDI Naming context
Context initctx = new InitialContext();

//perform JNDI lookup to obtain the connection factory
javax.resource.cci.ConnectionFactory cf =
   (javax.resource.cci.ConnectionFactory)initctx.lookup("java:comp/env/imsdblocal");

//specify connection properties
IMSConnectionSpec connSpec = new IMSConnectionSpec();
connSpec.setUser("user");
connSpec.setPassword("password");

//create CCI connection
javax.resource.cci.Connection conn = cf.getConnection(connSpec);
```

以下のコーディング例は、特定の接続のリソース・アダプターのデフォルトの **MetadataURL** 値をオーバーライドする方法を示しています。この方法で値をオーバーライドすると、デフォルト値は変更されず、J2C 接続ファクトリーのパラメーターを修正する必要もありません。

```
InitialContext ic = new InitialContext();
DataSource ds = (DataSource) ic.lookup("myJNDIName");
Connection con = ((IMSHybridDataSource)ds).getConnection(iSpec);
```

関連タスク:

820 ページの『SSL サポート用の IMS Universal ドライバーの構成』

関連資料:

IMS Universal ドライバー の Common Client Interfaceのサポート (アプリケーション・プログラミング API)

管理対象環境での IMS Universal JCA/JDBC ドライバーを使用した接続

管理対象 (または 3 層) 環境で JDBC プログラミング・インターフェースを使用して IMS データベースにアクセスするには、Java EE アプリケーション・サーバーに IMS Universal JCA/JDBC ドライバーを配置し、接続プロパティーを構成する必要があります。

IMS Universal JCA/JDBC ドライバーを構成して使用し、IMS データベースにアクセスするには、以下のようにします。

1. 管理コンソールを使用して、IMS Universal JCA/JDBC ドライバーを WebSphere Application Server にデプロイします。
2. 管理コンソールを使用して、WebSphere Application Server に IMS Universal JCA/JDBC ドライバーで使用する接続ファクトリーを作成します。
   a. 接続ファクトリーの名前および Java Naming and Directory Interface (JNDI) 名を指定します。 接続ファクトリー・インターフェースを javax.sql.DataSource と設定します。
   b. IMS Universal JCA/JDBC ドライバー接続ファクトリー用に次のカスタム接続プロパティーを設定します。

      DatastoreName

      アクセスする IMS データ・ストアの名前。

      - タイプ 4 コネクティビティーを使用する場合、**DatastoreName** プロパティーは、ODBM に定義されているデータ・ストアの名前と一致したものであるかまたはブランクである必要があります。データ・ストア名は、DATASTORE(NAME=*name*) パラメーターまたは DATASTORE(NAME=*name*, ALIAS (NAME=*aliasname*)) パラメーターのいずれかを使用して、ODBM CSLDCxxx PROCLIB メンバー内で定義されます。別名が指定される場合、**datastoreName** プロパティーの値として *aliasname* を指定する必要があります。 **DatastoreName** の値が空白 (または指定されていない) 場合、ODBM に定義されているすべてのデータ・ストアでデータ共用が使用可能となっていると見なされるため、IMS Connect は使用可能な任意の ODBM のインスタンスに接続します。
      - タイプ 2 コネクティビティーを使用する場合は、**DatastoreName** プロパティーを IMS サブシステムの別名に設定します。Java 従属領域のランタイムの場合は、これは設定する必要はありません。

      DatabaseName

      ターゲット IMS データベースを表すデータベース・メタデータの場所。

      メタデータが IMS カタログに保管されているのか、IMS Enterprise Suite Explorer for Development によって生成された静的メタデータ・クラスとして保管されているのかに応じて、**DatabaseName** プロパティーを以下の 2 つの方法のいずれかで指定することができます。

      - IMS システムで IMS カタログを使用している場合、**DatabaseName** プロパティーは、アプリケーションがターゲット IMS データベースへのアクセスに使用する PSB の名前です。
      - IMS Explorer for Development を使用している場合、**databaseName** プロパティーは、IMS Explorer for Development によって生成された Java メタデータ・クラスの完全修飾名です。URL には、`class://` の接頭部を付ける必要があります (例えば、`class://com.foo.BMP255DatabaseView`)。

J2C 接続ファクトリー環境では、リソース・アダプターに指定されたデフォルト値に影響を与えずに、個別の接続の **DatabaseName** プロパティーをオーバーライドできます。

MetadataURL

ターゲット IMS データベースを表すデータベース・メタデータの場所。

このプロパティーは推奨されていません。代わりに **DatabaseName** を使用してください。

MetadataURL プロパティーは、IMS Enterprise Suite Explorer for Developmentによって生成された Java メタデータ・クラスの完全修飾名です。 URL には、`class://` の接頭部を付ける必要があります (例えば、`class://com.foo.BMP255DatabaseView`)。

J2C 接続ファクトリー環境では、リソース・アダプターに指定されたデフォルト値に影響を与えずに、個別の接続の **MetadataURL** プロパティーをオーバーライドできます。

PortNumber

IMS Connect と通信するために使用される TCP/IP サーバーのポート番号。ポート番号は、IMS Connect 構成 PROCLIB メンバーの ODACCESS ステートメントで DRDAPORT パラメーターを使用して定義されます。デフォルトのポート番号は 8888 です。タイプ 2 コネクティビティーの使用時にはこのプロパティーを設定しないでください。

DatastoreServer

データ・ストア・サーバー (IMS Connect) の名前または IP アドレス。ホスト名 (例えば、dev123.svl.ibm.com) または IP アドレス (例えば、192.166.0.2) のいずれかを指定できます。タイプ 2 コネクティビティーの使用時にはこのプロパティーを設定しないでください。

DriverType

使用するドライバー・コネクティビティーのタイプ。 **DriverType** の値は、タイプ 4 コネクティビティーに対しては「4」、タイプ 2 コネクティビティーに対しては「2」である必要があります。ドライバーが WebSphere Application Server for z/OS で実行する場合、RRSLocalOption コネクティビティーに対しては、**DriverType** の値を「2_CTX」に設定することもできます。

sslConnection

オプション。この接続がデータ暗号化用に Secure Sockets Layer (SSL) を使用するかどうかを示します。 SSL を使用可能にするにはこのプロパティーを「true」に設定します。使用しない場合は「false」を設定します。タイプ 2 コネクティビティーの使用時にはこのプロパティーを設定しないでください。

sslKeyStoreType

オプション。セキュア・ソケット接続の確立に必要な暗号オブジェクトを含むファイルの形式を指定します。有効な値

は、「JKS」および「PKCS12」です。この値は、**sslConnection** が「true」に設定されていて、**sslKeyStoreType** が指定されていない場合にのみ使用します。 **sslKeyStoreType** パラメーターのデフォルトは「JKS」です。

sslSecureSocketProtocol
オプション。新しい接続の暗号通信プロトコルを指定します。サーバーがサポートしているプロトコルで、最高レベルのセキュリティーを提供するプロトコルを指定します。有効な値は、「SSL」、「SSLv3」、「TLSv1.1」、および「TLSv1.2」です。この値は、**sslConnection** が「true」に設定されている場合のみ使用します。**sslConnection** が「true」に設定されていて、**sslSecureSocketProtocol** が指定されていない場合、デフォルト・プロトコルは JRE とサーバーによって実行時に決定されます。

sslTrustStoreLocation
オプション。新しい接続の暗号トラストストア・ファイルのロケーションを指定します。この値は、**sslConnection** が true に設定されている場合のみ使用します。

sslTrustStorePassword
オプション。暗号トラストストア・ファイルにアクセスするためのパスワードを指定します。この値は、**sslConnection** が true に設定されている場合のみ使用します。

sslKeyStoreLocation
オプション。新しい接続の暗号鍵ストア・ファイルのロケーションを指定します。この値は、**sslConnection** が true に設定されている場合のみ使用します。

sslKeyStorePassword
オプション。暗号鍵ストア・ファイルにアクセスするためのパスワードを指定します。この値は、**sslConnection** が true に設定されている場合のみ使用します。

loginTimeout
オプション。接続初期設定またはサーバー要求時にドライバーがサーバーからの応答を待つ秒数を指定します。この時間が過ぎるとタイムアウトになります。このプロパティーには、秒数として負でない整数を指定します。タイムアウトの長さを無制限にする場合は、このプロパティーを 0 に設定してください。タイプ 2 コネクティビティーの使用時にはこのプロパティーを設定しないでください。

user
RACF 管理者によって提供された IMS Connect に接続するためのユーザー名。タイプ 2 コネクティビティーの使用時にはこのプロパティーを設定しないでください。

password
RACF 管理者によって提供された IMS Connect に接続するためのパスワード。タイプ 2 コネクティビティーの使用時にはこのプロパティーを設定しないでください。

signedCompare

オプション。このプロパティーが「true」に設定された場合、署名されたデータ・タイプについての範囲が指定された照会をサポートするために特殊 SSA が生成されます。プロパティーが「false」に設定された場合は、データ・タイプ値のバイナリー表現に基づいて標準バイナリー比較が実行されます。この値を「false」に設定すると、パフォーマンスは上がりますが、誤った結果がもたらされる可能性があります。このプロパティーのデフォルト値は「true」です。

flattenTables

オプション。データベース表をフラット化されたビューで生成するかどうかを示します。値を true にすると、表の追加の列として STRUCT または ARRAY のサブエレメントが公開されます。デフォルト値は false です。

- IMS Explorer は、コピーブックをインポートする際にコピーブックの構造をフラット化します。IMS カタログ内ではコピーブック自体は変更されないままですが、その特定の接続についての各表の構造に関する情報は変更されます。
- flattenTables プロパティーを指定すると、複合構造内のフィールドを直接照会できます。複合構造のフラット化のサポートについて詳しくは、 709 ページの『複合構造のフラット化のサポート』を参照してください。

制約事項**:** flattenTables 接続プロパティーでは、静的配列と静的構造のみがサポートされます。動的配列は変更されません。

t2OutputBufferSize

オプション。タイプ 2 接続の場合の SELECT 操作からの結果をバイト単位で表した出力バッファーのサイズ。

t2OutputBufferSize の最小値は 500000 です。500000 より小さい値が設定された場合、このプロパティー値は 500000 に調整されます。最大限界はありません。デフォルト値は 1280000 です。

treatInvalidDecimalAsNull

オプション。Java アプリケーションで無効と見なされる特定の 10 進値 (無効な符号ビットが設定された PACKEDDECIMAL および ZONEDDECIMAL など) をヌルとして解釈するかどうかを指示します。デフォルトでは、このプロパティーは「false」であるため、Java アプリケーションが無効値を処理しているときには変換例外がスローされます。

currentSchema

オプション。動的に準備される SQL ステートメントで非修飾データベース・オブジェクトを修飾するために使用される、デフォルトのスキーマ名を指定します。

dbViewLocation

オプション。databaseView メタデータ・クラスへの絶対パスを指定します。このプロパティーを使用して、プロジェクト・パスに配置されていないメタデータ・クラスを組み込むことができます。

dpsbOnCommit

オプション。コミットの発生時に PSB の割り振りを解除するには、このプロパティーを **true** に設定します。

推奨事項: 統合接続プールを使用する管理対象環境の場合を除き、このプロパティーを **true** に設定しないでください。

fetchSize

オプション。さらに多くの行が必要な場合に、データベースから取得する行数に関するヒントをクライアントに示します。このプロパティーに指定された数値は、現行接続でリトリーブされたデータのみに影響します。指定された値が 0 の場合、利用可能なすべての行が返されます。

管理対象接続および管理対象外接続のいずれも、このプロパティーのデフォルト値は 0 です。

llField

オプション。このプロパティーを **true** に設定すると、LL フィールドのデータは結果セットに通常の列として表示されます。LL フィールドの値を変更して、可変長セグメントのインスタンス長を変更できます。

maxRows

オプション。照会結果セットで返す最大行数を指定します。デフォルト値は 0 で、結果セットで利用可能なすべての行を返します。

traceFile

オプション。接続するトレース・ファイルの名前を指定します。

traceFileAppend

オプション。指定のトレース・ファイルが存在する場合、このプロパティーを true に設定すると、既存のトレース・ファイルを上書きせずに、そのトレース・ファイルに新規接続のトレース・データを追加する必要があることを指定します。

traceFile に値が指定されていない場合、このプロパティーは無視されます。

traceDirectory

オプション。トレース・ファイルが入っているファイル・システム・ディレクトリーを指定します。デフォルトでは、このパスはアプリケーションを実行するディレクトリーです。

traceFile に値が指定されていない場合、このプロパティーは無視されます。

traceLevel

オプション。接続に使用可能なトレースを指定します。このプロパティーの有効値は、IMSDataSource クラスの Java API 文書で定義されています。

デフォルトでは、すべてのトレースが使用不可になっています。

traceFile に値が指定されていない場合、このプロパティーは無視されます。

トレース・レベル	**traceLevel** パッケージの値	**IMSDataSource** 内の **traceLevel** 定数フィールド	**traceLevel** の **10** 進値
すべて	com.ibm.ims.db.opendb.*	TRACE_ALL	-1
DL/I	com.ibm.ims.db.opendb.dli.*	TRACE_DLI	28
DRDA	com.ibm.ims.db.opendb.drda.*	TRACE_DRDA	1
JDBC	com.ibm.ims.db.opendb.jdbc.*	TRACE_JDBC	32
Java EE	com.ibm.ims.opendb.spi.* com.ibm.ims.db.opendb.cci.*	TRACE_JEE	192

3. Java EE アプリケーションのデプロイメント記述子に、前のステップで作成した接続ファクトリーのリソース参照を追加します。 リソース参照の名前を接続ファクトリーの JNDI 名に設定し、タイプを javax.resource.cci.ConnectionFactory に設定します。
4. Java EE アプリケーションで、初期 JNDI ネーミング・コンテキストを作成し、JNDI 検索を使用して IMS Universal ハイブリッド・ドライバー用の javax.sql.DataSource インスタンスを取得します。 以下のサンプル・コードは、DataSource インスタンスを取得するための JNDI 検索の実行方法の例を示しています。ここで、接続ファクトリーの JNDI 名は「imsdblocal」です。

```
InitialContext ic = new InitialContext();
javax.sql.DataSource ds =
 (DataSource)ic.lookup("java:comp/env/imsdblocal");
```

5. 次のコード・サンプルで示すように、DataSource インスタンスで getConnection メソッドを使用して、java.sql.Connection インスタンスを取得します。

```
Connection con = ds.getConnection();
```

6. Java EE アプリケーションの接続が終了した場合は、Connection インターフェースで close メソッドを使用して接続を閉じます。

IMS Universal JCA/JDBC ドライバーを使用して IMS データベースに接続するサンプル・コード

以下のサンプル・コードは、Java EE アプリケーションから IMS Universal JCA/JDBC ドライバーを使用して IMS データベースに接続する流れを示しています。

```
//obtain the initial JNDI Naming context
InitialContext ic = new InitialContext();

//perform JNDI lookup to obtain the data source
javax.sql.DataSource ds =
 (DataSource)ic.lookup("java:comp/env/imsdblocal");

//specify connection properties
ds.setUser("myUserID");
ds.setPassword("myPassword");
props.put("sslConnection", "true");
props.put("loginTimeout", "10");

//create JDBC connection
java.sql.Connection con = ds.getConnection();
```

関連タスク:

820 ページの『SSL サポート用の IMS Universal ドライバーの構成』

関連資料:

サポートされる javax.sql.DataSource のメソッド (アプリケーション・プログラミング API)

IMS Universal Database リソース・アダプター CCI プログラミング・インターフェースを使用したサンプル EJB アプリケーション

次のサンプル EJB Bean は、管理対象環境で IMS Universal Database リソース・アダプターを使用する JCA アプリケーションの基本的なプログラミングの流れを示しています。

```
package client;

import java.sql.SQLException;
import javax.naming.InitialContext;
import javax.resource.ResourceException;
import javax.resource.cci.Connection;
import javax.resource.cci.ConnectionFactory;
import javax.resource.cci.Interaction;
import javax.resource.cci.ResultSet;
import javax.transaction.UserTransaction;
import com.ibm.ims.db.cci.SQLInteractionSpec;

/**
 * Bean implementation class for Enterprise Bean: StatefulBeanManaged
 */
public class BeanManagedSampleBean implements javax.ejb.SessionBean {

    private javax.ejb.SessionContext mySessionCtx;

    public void execute() throws Exception {
        InitialContext ic = new InitialContext();
        ConnectionFactory cf =
     (ConnectionFactory) ic.lookup("java:comp/env/MyMCF");
        Connection conn = null;
        UserTransaction ut = null;

        try {
            ut = this.mySessionCtx.getUserTransaction();
            ut.begin();

            conn = cf.getConnection();
            Interaction ix = conn.createInteraction();
            SQLInteractionSpec iSpec = new SQLInteractionSpec();

            // This query will return information for each person
            // in the phonebook with the last name WATSON
            iSpec.setSQL("SELECT * FROM " +
      "PCB01.PHONEBOOK WHERE LASTNAME='WATSON'");

            ResultSet rs = (ResultSet) ix.execute(iSpec, null);

            // Print out the first name of every person in the
            // phonebook with the last name WATSON
            while (rs.next()) {
                System.out.println(rs.getString("FIRSTNAME"));
            }

            rs.close();
            ix.close();
```

```
            ut.commit();
            conn.close();
        } catch (ResourceException e) {
            ut.rollback();
            conn.close();
        } catch (SQLException e) {
            ut.rollback();
            conn.close();
        }
    }

    /**
     * getSessionContext
     */
    public javax.ejb.SessionContext getSessionContext() {
        return mySessionCtx;
    }

    /**
     * setSessionContext
     */
    public void setSessionContext(javax.ejb.SessionContext ctx) {
        mySessionCtx = ctx;
    }

    /**
     * ejbCreate
     */
    public void ejbCreate() throws javax.ejb.CreateException {
    }

    /**
     * ejbActivate
     */
    public void ejbActivate() {
    }

    /**
     * ejbPassivate
     */
    public void ejbPassivate() {
    }

    /**
     * ejbRemove
     */
    public void ejbRemove() {
    }
}
```

DLIInteractionSpec クラスを使用した IMS データへのアクセス

IMS Universal Database リソース・アダプターを使用し、DL/I と似たプログラミング・セマンティクスを用いる IMS データベースからデータをリトリーブ、挿入、更新、および削除するには DLIInteractionSpec クラスを使用します。

アプリケーション・コンポーネントで、IMS データベースのデータをリトリーブ、挿入、更新、または削除できるようにする前に、データベースに対する物理接続用の javax.resource.cci.Connection インスタンスを取得する必要があります。

DLIInteractionSpec クラスを使用してデータをリトリーブ、挿入、更新、および削除するには、以下のようにします。

1. アプリケーション・コンポーネントで、Connection.createInteraction メソッドを使用して新規 javax.resource.cci.Interaction インスタンスを作成します。 例えば、以下のサンプル・コードでは、*con* が IMS データベースに対する javax.resource.cci.Connection インスタンスです。

   ```
   Interaction ix = con.createInteraction();
   ```

2. 新規 com.ibm.ims.db.cci.DLIInteractionSpec インスタンスを作成します。

   ```
   DLIInteractionSpec iSpec = new DLIInteractionSpec();
   ```

3. DLIInteractionSpec.setFunctionName メソッドを使用して実行する関数を設定し、入力パラメーターとして以下の表にリストされている関数の定数値を指定します。

実行するデータ操作	**setFunctionName** 値
データ・リトリーブ	DLIInteractionSpec.RETRIEVE
データ挿入	DLIInteractionSpec.CREATE
データ更新	DLIInteractionSpec.UPDATE
データ削除	DLIInteractionSpec.DELETE

   例えば、以下のサンプル・コードはデータ・リトリーブ操作を指定します。

   ```
   iSpec.setFunctionName(DLIInteractionSpec.RETRIEVE);
   ```

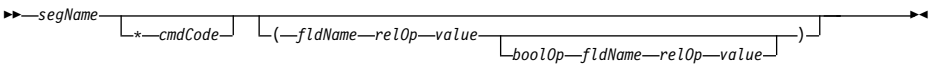
4. DLIInteractionSpec.setPCBName メソッドを使用して PCB 名を設定します。 例えば、以下のサンプル・コードでは、この対話で使用される PCB を「PCB01」と指定します。

   ```
   iSpec.setPCBName("PCB01");
   ```

5. DLIInteractionSpec.setSSAList メソッドを使用して、セグメント検索引数 (SSA) リストを設定します。 setSSAList メソッドを使用すると、従来の DL/I のような構文で SSA を指定できます。

   - 引数内のストリングとして、手動で SSA 修飾ステートメントを指定できます。構文は次のとおりです。

     setSSAList メソッドでのセグメント検索引数修飾ステートメントの構文

     segName
     IMS Enterprise Suite Explorer for Developmentによって生成された Java メタデータ・クラスで定義されているセグメントの名前。

     cmdCode (オプション**)**
     Q および C 以外のすべての DL/I コマンド・コードがサポートされます。

     fldName
     フィールドの名前。

     relOp SSA 修飾ステートメントの関係演算子。サポートされる値は次のとおりです。

     = 等しい

!= 等しくない

> より大きい

>= より大か等しい

< より小さい

<= 以下

value フィールド値のストリング表記。値が文字ベースの場合は、ストリングを引用符で囲む必要があります。値が数値の場合は、引用符で囲む必要はありません。文字ベースの値に引用符が含まれている場合は、値の中の引用符のエスケープとして単一引用符を使用します。例えば、値が「*O'brian*」の場合、「*O''brian*」と入力します。

boolOp
追加のフィールド・レベル修飾を追加するためのブール演算子。サポートされるブール演算子は次のとおりです。

- 論理 AND (* または & で指定)
- 論理 OR (+ または | で指定)
- 独立 AND (# で指定)

次のコード・サンプルは、「ALEXANDRIA」病院において、すべての病棟で、5 人より多くの医者と 3 人より少ない看護士によって治療を受けた最後の患者を返す、セグメント検索引数リストを設定する方法を示しています。*L コマンドは「最後のオカレンス」を意味します。

```
String ssaList =
"Hospital(HospName='ALEXANDRIA') Ward(Doccount>5 | Nurcount<3) Patient *L";
iSpec.setSSAList(ssaList);
```

- 手動でストリングを指定する代わりに、com.ibm.ims.db.cci.SSAListHelper クラスを使用してストリングを生成できます。

  次のサンプル・コードは、SSAListHelper を使用して SSA 修飾ステートメント・ストリングを設定する方法を示しています。

```
SSAListHelper sh = new SSAListHelper();
sh.addInitialQualification
  ("Hospital","HospName",SSAListHelper.EQUALS, "ALEXANDRIA");
sh.addInitialQualification("Ward","Doccount",
   SSAListHelper.GREATER_THAN, 5);
sh.appendQualification("Ward",SSAListHelper.OR, "Nurcount",
   SSAListHelper.LESS_THAN, 3);
sh.addCommandCode("Patient",SSAListHelper.CC_L);
iSpec.setSSA(sh.toString());
```

6. ConnectionFactory.getRecordFactory メソッドを使用して javax.resource.cci.RecordFactory インスタンスを作成します。 例えば、以下のサンプル・コードでは、ConnectionFactory インスタンス *rf* が作成されます。

```
RecordFactory rf = cf.getRecordFactory();
```

   DELETE 操作には、このステップは必要ありません。

7. RecordFactory.getMappedRecord メソッドを使用して javax.resource.cci.MappedRecord インスタンスを作成します。 RETRIEVE 操作の場合は、作成するレコードの名前を引数としてこのメソッドに渡します。CREATE または UPDATE 操作の場合は、この引数は挿入または更新するセグ

メントの名前です。 例えば、以下の RETRIEVE 操作のサンプル・コードでは、*rf* が javax.resource.cci.RecordFactory インスタンスです。

```
MappedRecord input = rf.createMappedRecord("myHospitalRecord");
```

DELETE 操作では MappedRecord は使用されていないため、このステップは必要ありません。

8. MappedRecord.put メソッドを使用して、データ操作のターゲットとするフィールドを指定します。最初の引数として、フィールドの名前をこのメソッドに渡します。 CREATE または UPDATE 操作の場合は、値だけでなく、挿入または更新するフィールドを指定するために MappedRecord を使用します。 2 番目の引数として、フィールドの値を渡します。 RETRIEVE 操作の場合、呼び出しの結果としてアプリケーションがリトリーブの候補とするフィールドを指定するために MappedRecord が使用され、put メソッド呼び出しの 2 番目の引数は無視されます (null を渡すことができます)。どのフィールドも指定しない場合、RETRIEVE 操作では、そのレコードのリーフ・セグメントのすべてのフィールドと、*D コマンドによって SSA が指定されたセグメント内のすべてのフィールドが返されます。 CREATE、UPDATE、および RETRIEVE 操作では、複数の MappedRecord.put メソッド呼び出しを行うことによって、複数のフィールドを指定できます。 例えば、以下のコード・サンプルでは、*input* が javax.resource.cci.MappedRecord インスタンスであり、「HospCode」がリトリーブするフィールドの名前です。

   ```
   input.put("HospCode", null);
   ```

   DELETE 操作では MappedRecord は使用されないため、このステップは必要ありません。

9. Interaction.execute メソッドを呼び出して照会を実行します。引数として DLIInteractionSpec オブジェクトおよび MappedRecord オブジェクトを渡します。 照会が正しく実行されると、このメソッドは照会結果と一緒に Record オブジェクトを返します。 Record インスタンスを javax.resource.cci.ResultSet にキャストし、その結果をアプリケーション・コンポーネントで表データとして処理できます。例えば、以下のコード・サンプルでは、*results* は javax.resource.cci.ResultSet インスタンス、*ix* は javax.resource.cci.Interaction インスタンス、*iSpec* は com.ibm.ims.db.cci.DLIInteractionSpec インスタンス、および *input* は javax.resource.cci.MappedRecord インスタンスです。

   ```
   results = (ResultSet)ix.execute(iSpec, input);
   ```

DLIInteraction インターフェースを使用した IMS データ操作のサンプル・コード

次に示す詳細なサンプル・コードは、DLIInteraction インターフェースを使用して WARD セグメントからフィールドをリトリーブする方法を示しています。

```
package client;

import java.sql.SQLException;
import javax.naming.InitialContext;
import javax.resource.ResourceException;
import javax.resource.cci.Connection;
import javax.resource.cci.ConnectionFactory;
import javax.resource.cci.Interaction;
import javax.resource.cci.ResultSet;
import javax.transaction.UserTransaction;
```

```
import com.ibm.ims.db.cci.SQLInteractionSpec;

/**
 * Bean implementation class for Enterprise Bean: StatefulBeanManaged
 */
public class BeanManagedSampleDLIBean implements javax.ejb.SessionBean {

    private javax.ejb.SessionContext mySessionCtx;

    public void execute() throws Exception {
        InitialContext ic = new InitialContext();
        ConnectionFactory cf =
    (ConnectionFactory) ic.lookup("java:comp/env/MyMCF");
        Connection conn = null;
        UserTransaction ut = null;

        try {
            ut = this.mySessionCtx.getUserTransaction();
            ut.begin();

            conn = cf.getConnection();
            Interaction ix = conn.createInteraction();

            DLIInteractionSpec iSpec = new DLIInteractionSpec();
            iSpec.setFunctionName("RETRIEVE");
            iSpec.setPCBName("PCB09");

            // This query will return the WARDNAME, PATCOUNT, DOCCOUNT,
            // and NURCOUNT fields for all WARDs with WARNNO = 51
            iSpec.setSSAList("WARD (WARDNO = '51')");

            // Create RecordFactory
            RecordFactory rf = cf.getRecordFactory();

            // Create Record
            MappedRecord input = rf.createMappedRecord("WARD");
            // Specify the fields to retrieve
            input.put("WARDNAME", null);
            input.put("PATCOUNT", null);
            input.put("DOCCOUNT", null);
            input.put("NURCOUNT", null);

            ResultSet results = (ResultSet) ix.execute(iSpec, input);
            while (results.next()) {
                System.out.println(results.getString("WARDNAME"));
                System.out.println(results.getString("PATCOUNT"));
                System.out.println(results.getString("DOCCOUNT"));
                System.out.println(results.getString("NURCOUNT"));
            }

            rs.close();
            ix.close();
            ut.commit();
            conn.close();
        } catch (ResourceException e) {
            ut.rollback();
            conn.close();
        } catch (SQLException e) {
            ut.rollback();
            conn.close();
        }
    }

    /**
     * getSessionContext
     */
    public javax.ejb.SessionContext getSessionContext() {
```

```
        return mySessionCtx;
    }

    /**
     * setSessionContext
     */
    public void setSessionContext(javax.ejb.SessionContext ctx) {
        mySessionCtx = ctx;
    }

    /**
     * ejbCreate
     */
    public void ejbCreate() throws javax.ejb.CreateException {
    }

    /**
     * ejbActivate
     */
    public void ejbActivate() {
    }

    /**
     * ejbPassivate
     */
    public void ejbPassivate() {
    }

    /**
     * ejbRemove
     */
    public void ejbRemove() {
    }
}
```

関連概念:

718 ページの『IMS Universal Database リソース・アダプターを使用した IMS への接続』

関連資料:

IMS Universal ドライバー の Common Client Interfaceのサポート (アプリケーション・プログラミング API)

SQLInteractionSpec クラスを使用した IMS データへのアクセス

IMS Universal Database リソース・アダプターを使用した SQL 照会を用いて IMS データベースからデータをリトリーブ、挿入、更新、および削除するには、SQLInteractionSpec クラスを使用します。 IMS Universal Database リソース・アダプターは、IMS Universal JDBC ドライバーと同じ SQL ステートメント構文および使用法をサポートし、同じ制約事項があります。

アプリケーション・コンポーネントで、IMS データベースのデータをリトリーブ、挿入、更新、または削除できるようにする前に、データベースに対する物理接続用の javax.resource.cci.Connection インスタンスを取得する必要があります。

SQLInteractionSpec クラスを使用してデータをリトリーブ、挿入、更新、および削除するには、以下のようにします。

1. アプリケーション・コンポーネントで、Connection.createInteraction メソッドを使用して新規 javax.resource.cci.Interaction インスタンスを作成します。 例

えば、以下のサンプル・コードでは、*con* が IMS データベースに対する javax.resource.cci.Connection インスタンスです。

```
Interaction ix = con.createInteraction();
```

2. 新規 com.ibm.ims.db.cci.SQLInteractionSpec インスタンスを作成します。

```
SQLInteractionSpec iSpec = new SQLInteractionSpec();
```

3. SQLInteractionSpec.setSQL メソッドを使用して、SQL 照会ストリングを設定します。 照会では、WHERE 節で ? パラメーター・マーカーを使用して、修飾カラムの値を指定できます。これは値が後で指定されることを意味します (JDBC の PreparedStatement と似ています)。
   - 次の例は、パラメーター・マーカーを使用しないで SELECT ステートメントを指定する方法を示しています。ここで *iSpec* は、SQLInteractionSpec のインスタンスです。

     ```
     iSpec.setQuery("SELECT PATIENT.PATNAME, ILLNESS.ILLNAME "+
          "FROM pcb01.HOSPITAL, pcb01.PATIENT, pcb01.ILLNESS " +
          "WHERE HOSPITAL.HOSPNAME='SANTA TERESA'");
     ```

   - 次の例は、パラメーター・マーカーを使用して SELECT ステートメントを実行する方法を示しています。ここで *iSpec* は、SQLInteractionSpec のインスタンスです。

     ```
     iSpec.setQuery("SELECT PATIENT.PatName, WARD.WardName "+
          "FROM pcb01.HOSPITAL, pcb01.PATIENT, pcb01.WARD " +
          "WHERE HOSPITAL.HospName=? AND WARD.DocCount>?");
     ```

4. ConnectionFactory.getRecordFactory メソッドを使用して javax.resource.cci.RecordFactory インスタンスを作成します。 パラメーター・マーカーを使用しない SQL 照会では、RecordFactory の作成は必要ありません。
5. RecordFactory.getIndexedRecord メソッドを使用して javax.resource.cci.IndexedRecord インスタンスを作成します。作成するレコードの名前を引数としてこのメソッドに渡します。 例えば、以下のサンプル・コードでは、*rf* が javax.resource.cci.RecordFactory インスタンスです。

   ```
   IndexedRecord input = rf.createIndexedRecord("myPatientRecord");
   ```

   sa パラメーター・マーカーを使用しない SQL 照会では、IndexedRecord の作成は必要ありません。
6. 照会ストリングでパラメーター・マーカーを使用する場合は、IndexedRecord.add メソッドを使用して WHERE 節を修飾します。 例えば、ステップ 5 のものと同じ、パラメーター・マーカーを使用した同じ照会ストリングを使用する場合、*input* は IndexedRecord のインスタンスとなります。

   ```
   input.add(1, "Santa Teresa"); //HospName value is "Santa Teresa"
   input.add(2, 5);              //DocCount value is greater than 5
   ```

7. Interaction.execute メソッドを呼び出して照会を実行します。引数として SQLInteractionSpec オブジェクトおよび IndexedRecord オブジェクトを渡します。 SQL 照会でパラメーター・マーカーを使用しない場合、execute メソッド呼び出しの 2 番目の引数は無視されます (null を渡すことができます)。照会が正しく実行されると、このメソッドは照会結果と一緒に Record オブジェクトを返します。 Record インスタンスを javax.resource.cci.ResultSet にキャストし、その結果をアプリケーション・コンポーネントで表データとして処理できます。例えば、以下のコード・サンプルでは、*results* は

javax.resource.cci.ResultSet インスタンス、*ix* は javax.resource.cci.Interaction インスタンス、*iSpec* は com.ibm.ims.db.cci.SQLInteractionSpec インスタンス、および*input* は javax.resource.cci.IndexedRecord インスタンスです。

```
results = (ResultSet)ix.execute(iSpec, input);
```

SQLInteractionSpec クラスを使用した IMS データ操作のサンプル・コード

次のサンプル・コードでは、SQLInteractionSpec クラスを使用して PATIENT レコードから患者名をリトリーブする方法を示しています。

```
package client;

import java.sql.SQLException;
import javax.naming.InitialContext;
import javax.resource.ResourceException;
import javax.resource.cci.Connection;
import javax.resource.cci.ConnectionFactory;
import javax.resource.cci.Interaction;
import javax.resource.cci.ResultSet;
import javax.transaction.UserTransaction;
import com.ibm.ims.db.cci.SQLInteractionSpec;

/**
 * Bean implementation class for Enterprise Bean: StatefulBeanManaged
 */
public class BeanManagedSampleSQLBean implements javax.ejb.SessionBean {

    private javax.ejb.SessionContext mySessionCtx;

    public void execute() throws Exception {
        InitialContext ic = new InitialContext();
        ConnectionFactory cf =
     (ConnectionFactory) ic.lookup("java:comp/env/MyMCF");
        Connection conn = null;
        UserTransaction ut = null;

        try {
            ut = this.mySessionCtx.getUserTransaction();
            ut.begin();

            conn = cf.getConnection();
            Interaction ix = conn.createInteraction();
            SQLInteractionSpec iSpec = new SQLInteractionSpec();

            // This query will return the WARDNAME, PATCOUNT, DOCCOUNT,
            // and NURCOUNT fields for the WARD with WARDNO = 51
            iSpec.setSQL("SELECT WARDNAME, PATCOUNT, DOCCOUNT, " +
      "NURCOUNT FROM PCB09.WARD WHERE WARDNO='51'");

            ResultSet rs = (ResultSet) ix.execute(iSpec, null);

            while (rs.next()) {
                System.out.println(rs.getString("WARDNAME"));
                System.out.println(rs.getString("PATCOUNT"));
                System.out.println(rs.getString("DOCCOUNT"));
                System.out.println(rs.getString("NURCOUNT"));
            }

            rs.close();
            ix.close();
            ut.commit();
            conn.close();
        } catch (ResourceException e) {
```

```
            ut.rollback();
            conn.close();
        } catch (SQLException e) {
            ut.rollback();
            conn.close();
        }
    }

    /**
     * getSessionContext
     */
    public javax.ejb.SessionContext getSessionContext() {
        return mySessionCtx;
    }

    /**
     * setSessionContext
     */
    public void setSessionContext(javax.ejb.SessionContext ctx) {
        mySessionCtx = ctx;
    }

    /**
     * ejbCreate
     */
    public void ejbCreate() throws javax.ejb.CreateException {
    }

    /**
     * ejbActivate
     */
    public void ejbActivate() {
    }

    /**
     * ejbPassivate
     */
    public void ejbPassivate() {
    }

    /**
     * ejbRemove
     */
    public void ejbRemove() {
    }
}
```

関連概念:

718 ページの『IMS Universal Database リソース・アダプターを使用した IMS への接続』

関連資料:

IMS Universal ドライバー の Common Client Interfaceのサポート (アプリケーション・プログラミング API)

767 ページの『IMS Universal JDBC ドライバーを使用した SQL ステートメントの使用法』

IMS Universal JCA/JDBC ドライバーを使用した IMS データへのアクセス

Java EE 実行時環境内で IMS Universal JDBC ドライバーの全機能を使用する必要がある場合は、IMS Universal JCA/JDBC ドライバーを使用します。

Java EE アプリケーション・コンポーネントで、IMS データベースのデータをリトリーブ、挿入、更新、または削除できるようにする前に、データベースに対する物理接続用の java.sql.Connection インスタンスを取得する必要があります。

Java EE アプリケーション・コンポーネントで、JDBC アプリケーションと同じ方法で実行する、データ操作用のアプリケーション・ロジックをコーディングします。 IMS Universal JCA/JDBC ドライバーには、IMS Universal JDBC ドライバーと同じ SQL ステートメント構文サポートおよび使用上の制約事項があります。

IMS Universal JCA/JDBC ドライバーを使用した EJB アプリケーションの例

以下のコード・サンプルは、IMS データベースに接続し、SQL SELECT 照会を使用して患者名のリストをリトリーブし、SQL UPDATE 照会を使用して患者情報を変更する、Bean 管理の EJB アプリケーションを示しています。

```
package client;
import java.sql.SQLException;
import javax.naming.InitialContext;
import javax.sql.DataSource;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;

import javax.transaction.UserTransaction;
import com.ibm.ims.db.cci.SQLInteractionSpec;

/**
 * Bean implementation class for Enterprise Bean: StatefulBeanManaged
 */
public class JDBCBeanManagedSampleSQLBean {

 private javax.ejb.SessionContext mySessionCtx;

 public void execute() throws Exception {
  InitialContext ic = new InitialContext();
     DataSource ds =
      (DataSource) ic.lookup("java:comp/env/MyMCF");
     Connection conn = null;
     UserTransaction ut = null;

     try {
      ut = this.mySessionCtx.getUserTransaction();
         ut.begin();

         conn = ds.getConnection();

            Statement st = conn.createStatement();

            // List all of the patient names in the
            // SURG ward in the ALEXANDRIA hospital
            ResultSet rs = st.executeQuery("SELECT patname from " +
                    "pcb01.hospital, ward, patient " +
                    "where hospital.hospname = 'ALEXANDRIA' " +
                    "and ward.wardname = 'SURG'");
            while (rs.next()) {
                System.out.println(rs.getString("patname"));
            }

            // Update the name of the patient with patient
            // number 0222 in ward 04 in the hospital
            // with code R1210010000A
            int updatedRecords = st.executeUpdate("UPDATE PCB01.PATIENT " +
```

```
                "SET PATNAME='UPDATED NAME' WHERE PATNUM='0222' " +
                "AND HOSPITAL_HOSPCODE='R121001000A' AND WARD_WARDNO='04'");
            System.out.println("Updated " + updatedRecords + " Record(s)");

            rs.close();
        ut.commit();
        conn.close();
      } catch (SQLException e) {
       e.printStackTrace();
        ut.rollback();
        conn.close();
      }
    }

      /**
       * getSessionContext
       */
      public javax.ejb.SessionContext getSessionContext() {
          return mySessionCtx;
      }

      /**
       * setSessionContext
       */
      public void setSessionContext(javax.ejb.SessionContext ctx) {
          mySessionCtx = ctx;
      }

      /**
       * ejbCreate
       */
      public void ejbCreate() throws javax.ejb.CreateException {
      }

      /**
       * ejbActivate
       */
      public void ejbActivate() {
      }

      /**
       * ejbPassivate
       */
      public void ejbPassivate() {
      }

      /**
       * ejbRemove
       */
      public void ejbRemove() {
      }
}
```

関連資料:

767 ページの『IMS Universal JDBC ドライバーを使用した SQL ステートメントの使用法』

IMS Universal JDBC ドライバーを使用したプログラミング

IMS では、TCP/IP を介して IMS データベースにアクセスする、SQL ベース・データベース接続用の Java Database Connectivity (JDBC) ドライバーを、IMS Universal ドライバーに組み込まれている IMS Universal JDBC ドライバーに提供しています。 IMS Universal JDBC ドライバーは、JDBC 4.0 標準をベースとしています。

JDBC は、リレーショナル・データベースまた表形式データ・ソースにアクセスするために Java アプリケーションが使用する、アプリケーション・プログラミング・インターフェース (API) です。 JDBC API は、Java プログラミング言語と、JDBC インターフェースが実装された任意のデータベースとの間の、データベース非依存接続用の業界標準です。クライアントはこのインターフェースを使用して、データベース内のデータを照会したり更新したりします。ドライバーの実装対象の特定のデータベースの、(固有の) 基礎アクセス・プロトコルの実装は、JDBC ドライバー自体が受け持ちます。ドライバーはクライアント・サイドのアダプターであり (サーバーではなくクライアント・マシンにインストールされる)、Java プログラムからの要求を、データベース管理システム (DBMS) が理解できるプロトコルに変換します。

JDBC の IMS サポートにより、動的 SQL 呼び出しを発行して IMS データにアクセスし、表データ形式で返された結果セットを処理できる、Java アプリケーションを作成できます。 IMS Universal JDBC ドライバーは、対象を IMS データベース管理システムでネイティブに処理可能なものだけに制限された機能を備えた、SQL 構文のサブセットをサポートするように設計されています。 DBMS 中心の設計により、IMS Universal JDBC ドライバーでは、IMS のハイパフォーマンス機能を十分に活用できます。 IMS Universal JDBC ドライバーではさらに、集約関数サポート、および ORDER BY と GROUP BY のサポートも提供しています。

IMS Universal ドライバーを使用した Java アプリケーションを作成する準備

IMS Universal ドライバーを使用する Java アプリケーション・プログラムでは、Java Development Kit (JDK) 7.0 が必要です。JMP 領域および JBP 領域で実行される Java プログラムは、JDK 7.0 以降が必要です。IMS Universal ドライバーを使用する Java アプリケーション・プログラムは、IMS データベースと対話するために、データベース・メタデータへのアクセス権限を持っている必要があります。IMS カタログ・データベースで直接このメタデータにアクセスするか、IMS Enterprise Suite Explorer for Development を使用して Java メタデータ・クラスとしてこのメタデータを生成することができます。

関連資料:

- developerWorks Web サイトの IBM Java Development Kit

JDBC でサポートされるドライバー

IMS Universal JDBC ドライバーは、タイプ 2 およびタイプ 4 JDBC アーキテクチャーをサポートします。

以下の表は、4 つのタイプの JDBC ドライバー・アーキテクチャーで使用可能な IMS サポートをリストしています。

表 *100. IMS* でサポートされる *JDBC* ドライバー・アーキテクチャー

JDBC ドライバー・アーキテクチャー	説明	**IMS** サポート
タイプ 1	Open Database Connectivity (ODBC) などの、他のデータ・アクセス API へのマッピングとして JDBC API を実装するドライバー。このタイプのドライバーは、通常はネイティブ・ライブラリーに依存しているため、ポータビリティーは制限されています。	IMS はタイプ 1 ドライバーをサポートしていません。
タイプ 2	一部は Java プログラミング言語で作成されており、一部はネイティブ・コードで作成されているドライバー。このドライバーは、接続先のデータ・ソースに固有のネイティブ・クライアント・ライブラリーを使用します。ネイティブ・コードであるため、そのポータビリティーは制限されています。タイプ 2 JDBC コネクティビティーを使用する Java プログラムは、ターゲット IMS サブシステムとして同じ z/OS システムまたは zSeries 論理区画 (LPAR) 上で実行できます。	WebSphere Application Server for z/OS、IMS Java 従属領域 (JDR)、および CICS から IMS にアクセスする場合は、タイプ 2 コネクティビティーで IMS Universal JDBC ドライバーを使用します。
タイプ 3	ピュア Java クライアントを使用し、データベース非依存プロトコルを使用してサーバーと通信するドライバー。サーバーは次にクライアントの要求をデータ・ソースに通知します。	IMS はタイプ 3 ドライバーをサポートしていません。
タイプ 4	ピュア Java であり、特定のデータ・ソース用のネットワーク・プロトコルを実装するドライバー。クライアントはデータ・ソースに直接接続します。	TCP/IP ネットワーク接続を介して IMS サブシステムにアクセスする場合は、タイプ 4 コネクティビティーで IMS Universal JDBC ドライバーを使用します。

IMS Universal JDBC ドライバーを使用した IMS への接続

JDBC プログラムで照会を送信して結果を受信できるようにする前に、まず IMS データベースとの接続を確立する必要があります。

JDBC DataSource インターフェースを使用した IMS データベースへの接続

IMS Universal JDBC ドライバー・アプリケーションから IMS への接続には、DataSource インターフェースを使用することをお勧めします。

IMS Universal JDBC ドライバー・アプリケーションで DataSource インターフェースを作成して使用するには、以下のようにします。

1. com.ibm.ims.jdbc.IMSDataSource クラスのインスタンスを作成します。 このクラスは、DataSource インターフェースの IMS Universal JDBC ドライバーの実装です。
2. DataSource インスタンスの以下の接続プロパティーを設定します。

   DatastoreName

   アクセスする IMS データ・ストアの名前。

   - タイプ 4 コネクティビティーを使用する場合、**DatastoreName** プロパティーは、ODBM に定義されているデータ・ストアの名前と一致したものであるかまたはブランクである必要があります。データ・ストア名は、DATASTORE(NAME=*name*) パラメーターまたは DATASTORE(NAME=*name*, ALIAS(NAME=*aliasname*)) パラメーターのいずれかを使用して、ODBM CSLDCxxx PROCLIB メンバー内で定義されます。別名が指定される場合、**datastoreName** プロパティーの値として *aliasname* を指定する必要があります。 **DatastoreName** の値が空白 (または指定されていない) 場合、ODBM に定義されているすべてのデータ・ストアでデータ共用が使用可能となっていると見なされるため、IMS Connect は使用可能な任意の ODBM のインスタンスに接続します。
   - タイプ 2 コネクティビティーを使用する場合は、**DatastoreName** プロパティーを IMS サブシステムの別名に設定します。Java 従属領域のランタイムの場合は、これは設定する必要はありません。

   DatabaseName

   ターゲット IMS データベースを表すデータベース・メタデータの場所。

   メタデータが IMS カタログに保管されているのか、IMS Enterprise Suite Explorer for Development によって生成された静的メタデータ・クラスとして保管されているのかに応じて、**DatabaseName** プロパティーを以下の 2 つの方法のいずれかで指定することができます。

   - IMS システムで IMS カタログを使用している場合、**DatabaseName** プロパティーは、アプリケーションがターゲット IMS データベースへのアクセスに使用する PSB の名前です。
   - IMS Explorer for Development を使用している場合、**databaseName** プロパティーは、IMS Explorer for Development によって生成された Java メタデータ・クラスの完全修飾名です。URL には、`class://` の接頭部を付ける必要があります (例えば、`class://com.foo.BMP255DatabaseView`)。

J2C 接続ファクトリー環境では、リソース・アダプターに指定されたデフォルト値に影響を与えずに、個別の接続の **DatabaseName** プロパティーをオーバーライドできます。

MetadataURL

ターゲット IMS データベースを表すデータベース・メタデータの場所。

このプロパティーは推奨されていません。代わりに **DatabaseName** を使用してください。

MetadataURL プロパティーは、IMS Enterprise Suite Explorer for Developmentによって生成された Java メタデータ・クラスの完全修飾名です。 URL には、`class://` の接頭部を付ける必要があります (例えば、`class://com.foo.BMP255DatabaseView`)。

J2C 接続ファクトリー環境では、リソース・アダプターに指定されたデフォルト値に影響を与えずに、個別の接続の **MetadataURL** プロパティーをオーバーライドできます。

PortNumber

IMS Connect と通信するために使用される TCP/IP サーバーのポート番号。ポート番号は、IMS Connect 構成 PROCLIB メンバーの ODACCESS ステートメントで DRDAPORT パラメーターを使用して定義されます。デフォルトのポート番号は 8888 です。タイプ 2 コネクティビティーの使用時にはこのプロパティーを設定しないでください。

DatastoreServer

データ・ストア・サーバー (IMS Connect) の名前または IP アドレス。ホスト名 (例えば、dev123.svl.ibm.com) または IP アドレス (例えば、192.166.0.2) のいずれかを指定できます。タイプ 2 コネクティビティーの使用時にはこのプロパティーを設定しないでください。

DriverType

使用するドライバー・コネクティビティーのタイプ (値はタイプ 4 コネクティビティーに対しては `IMSDataSource.DRIVER_TYPE_4`、タイプ 2 コネクティビティーに対しては `IMSDataSource.DRIVER_TYPE_2` である必要があります)。

user RACF 管理者によって提供された IMS Connect に接続するためのユーザー名。タイプ 2 コネクティビティーの使用時にはこのプロパティーを設定しないでください。

password

RACF 管理者によって提供された IMS Connect に接続するためのパスワード。タイプ 2 コネクティビティーの使用時にはこのプロパティーを設定しないでください。

allMetadata

オプション。このプロパティーが `true` に設定された場合、DatabaseMetadata インターフェースは、IMS カタログにあるすべてのリソースに関する情報を返します。このプロパティーが `false` に設定

された場合、DatabaseMetadata インターフェースは、割り振り済み PSB に関する情報を返します。このプロパティーのデフォルト値は `false` です。

sslConnection
オプション。この接続がデータ暗号化用に Secure Sockets Layer (SSL) を使用するかどうかを示します。 SSL を使用可能にするにはこのプロパティーを「true」に設定します。使用しない場合は「false」を設定します。タイプ 2 コネクティビティーの使用時にはこのプロパティーを設定しないでください。

sslKeyStoreType
オプション。セキュア・ソケット接続の確立に必要な暗号オブジェクトを含むファイルの形式を指定します。有効な値は、「JKS」および「PKCS12」です。この値は、**sslConnection** が「true」に設定されていて、**sslKeyStoreType** が指定されていない場合にのみ使用します。 **sslKeyStoreType** パラメーターのデフォルトは「JKS」です。

sslSecureSocketProtocol
オプション。新しい接続の暗号通信プロトコルを指定します。サーバーがサポートしているプロトコルで、最高レベルのセキュリティーを提供するプロトコルを指定します。有効な値は、「SSL」、「SSLv3」、「TLSv1.1」、および「TLSv1.2」です。この値は、**sslConnection** が「true」に設定されている場合のみ使用します。**sslConnection** が「true」に設定されていて、**sslSecureSocketProtocol** が指定されていない場合、デフォルト・プロトコルは JRE とサーバーによって実行時に決定されます。

sslTrustStoreLocation
オプション。新しい接続の暗号トラストストア・ファイルのロケーションを指定します。この値は、**sslConnection** が true に設定されている場合のみ使用します。

sslTrustStorePassword
オプション。暗号トラストストア・ファイルにアクセスするためのパスワードを指定します。この値は、**sslConnection** が true に設定されている場合のみ使用します。

sslKeyStoreLocation
オプション。新しい接続の暗号鍵ストア・ファイルのロケーションを指定します。この値は、**sslConnection** が true に設定されている場合のみ使用します。

sslKeyStorePassword
オプション。暗号鍵ストア・ファイルにアクセスするためのパスワードを指定します。この値は、**sslConnection** が true に設定されている場合のみ使用します。

loginTimeout
オプション。接続初期設定またはサーバー要求時にドライバーがサーバーからの応答を待つ秒数を指定します。この時間が過ぎるとタイムアウ

トになります。このプロパティーには、秒数として負でない整数を指定します。タイムアウトの長さを無制限にする場合は、このプロパティーを 0 に設定してください。タイプ 2 コネクティビティーの使用時にはこのプロパティーを設定しないでください。

signedCompare
オプション。このプロパティーが「true」に設定された場合、署名されたデータ・タイプについての範囲が指定された照会をサポートするために特殊 SSA が生成されます。プロパティーが「false」に設定された場合は、データ・タイプ値のバイナリー表現に基づいて標準バイナリー比較が実行されます。この値を「false」に設定すると、パフォーマンスは上がりますが、誤った結果がもたらされる可能性があります。このプロパティーのデフォルト値は「true」です。

flattenTables
オプション。データベース表をフラット化されたビューで生成するかどうかを示します。値を true にすると、表の追加の列として STRUCT または ARRAY のサブエレメントが公開されます。デフォルト値は false です。

- IMS Explorer は、コピーブックをインポートする際にコピーブックの構造をフラット化します。IMS カタログ内ではコピーブック自体は変更されないままですが、その特定の接続についての各表の構造に関する情報は変更されます。
- flattenTables プロパティーを指定すると、複合構造内のフィールドを直接照会できます。複合構造のフラット化のサポートについて詳しくは、 709 ページの『複合構造のフラット化のサポート』を参照してください。

制約事項**:** flattenTables 接続プロパティーでは、静的配列と静的構造のみがサポートされます。動的配列は変更されません。

t2OutputBufferSize
オプション。タイプ 2 接続の場合の SELECT 操作からの結果をバイト単位で表した出力バッファーのサイズ。

t2OutputBufferSize の最小値は 500000 です。500000 より小さい値が設定された場合、このプロパティー値は 500000 に調整されます。最大限界はありません。デフォルト値は 1280000 です。

treatInvalidDecimalAsNull
オプション。Java アプリケーションで無効と見なされる特定の 10 進値 (無効な符号ビットが設定された PACKEDDECIMAL および ZONEDDECIMAL など) をヌルとして解釈するかどうかを指示します。デフォルトでは、このプロパティーは「false」であるため、Java アプリケーションが無効値を処理しているときには変換例外がスローされます。

3. オプション: IMSDataSource オブジェクトの setProperties メソッドを使用して、追加の接続プロパティーを設定します。

currentSchema
オプション。動的に準備される SQL ステートメントで非修飾データベース・オブジェクトを修飾するために使用される、デフォルトのスキーマ名を指定します。

dbViewLocation
オプション。databaseView メタデータ・クラスへの絶対パスを指定します。このプロパティーを使用して、プロジェクト・パスに配置されていないメタデータ・クラスを組み込むことができます。

dpsbOnCommit
オプション。コミットの発生時に PSB の割り振りを解除するには、このプロパティーを **true** に設定します。

推奨事項: 統合接続プールを使用する管理対象環境の場合を除き、このプロパティーを **true** に設定しないでください。

fetchSize
オプション。さらに多くの行が必要な場合に、データベースから取得する行数に関するヒントをクライアントに示します。このプロパティーに指定された数値は、現行接続でリトリーブされたデータのみに影響します。指定された値が 0 の場合、利用可能なすべての行が返されます。

管理対象接続および管理対象外接続のいずれも、このプロパティーのデフォルト値は 0 です。

llField
オプション。このプロパティーを **true** に設定すると、LL フィールドのデータは結果セットに通常の列として表示されます。LL フィールドの値を変更して、可変長セグメントのインスタンス長を変更できます。

maxRows
オプション。照会結果セットで返す最大行数を指定します。デフォルト値は 0 で、結果セットで利用可能なすべての行を返します。

traceFile
オプション。接続するトレース・ファイルの名前を指定します。

traceFileAppend
オプション。指定のトレース・ファイルが存在する場合、このプロパティーを true に設定すると、既存のトレース・ファイルを上書きせずに、そのトレース・ファイルに新規接続のトレース・データを追加する必要があることを指定します。

traceFile に値が指定されていない場合、このプロパティーは無視されます。

traceDirectory
オプション。トレース・ファイルが入っているファイル・システム・ディレクトリーを指定します。デフォルトでは、このパスはアプリケーションを実行するディレクトリーです。

traceFile に値が指定されていない場合、このプロパティーは無視されます。

traceLevel

オプション。接続に使用可能なトレースを指定します。このプロパティーの有効値は、IMSDataSource クラスの Java API 文書で定義されています。

デフォルトでは、すべてのトレースが使用不可になっています。

traceFile に値が指定されていない場合、このプロパティーは無視されます。

トレース・レベル	**traceLevel** パッケージの値	**IMSDataSource** 内の **traceLevel** 定数フィールド	**traceLevel** の **10** 進値
すべて	com.ibm.ims.db.opendb.*	TRACE_ALL	-1
DL/I	com.ibm.ims.db.opendb.dli.*	TRACE_DLI	28
DRDA	com.ibm.ims.db.opendb.drda.*	TRACE_DRDA	1
JDBC	com.ibm.ims.db.opendb.jdbc.*	TRACE_JDBC	32
Java EE	com.ibm.ims.opendb.spi.* com.ibm.ims.db.opendb.cci.*	TRACE_JEE	192

4. DataSource オブジェクトで getConnection メソッドを呼び出して、データ・ソースとの接続を確立します。
5. アプリケーションが接続を終了した後に、Connection インターフェースで close メソッドを使用して接続を閉じます。

以下のサンプル・コードでは、DataSource インターフェースを使用して IMS Universal JDBC ドライバー・アプリケーションから IMS データベースへのタイプ 4 接続を作成する方法を示しています。

```
import java.sql.*;
import javax.sql.*;
import com.ibm.ims.jdbc.*;

Connection conn = null;

// Create an instance of DataSource
IMSDataSource ds = new com.ibm.ims.jdbc.IMSDataSource();

// Set the URL of the fully qualified name of the Java metadata class
ds.setDatabaseName("class://BMP255.BMP255DatabaseView");

// Set the data store name
ds.setDatastoreName("IMS1");

// Set the data store server
ds.setDatastoreServer("ecdev47.svl.ibm.com");

// Set the port number
ds.setPortNumber(5555);

// Set the JDBC connectivity driver typ
ds.setDriverType(IMSDataSource.DRIVER_TYPE_4);

// Enable SSL for connection
ds.setSSLConnection("true");

// Set timeout for connection
ds.setLoginTimeout("10");
```

```
// Set user ID for connection
ds.setUser("myUserID");

// Set password for connection
ds.setPassword("myPassword");

// Create JDBC connection
conn = ds.getConnection();
```

代わりに、Properties オブジェクトのすべての接続プロパティーをキー値のペアとして設定し、次に IMSDataSource.setProperties メソッドを使用して接続プロパティーを同時に設定することができます。

```
Properties props = new Properties();
props.put( "user", "MyUserID" );
props.put( "password", "MyPassword" );

IMSDataSource ds = new com.ibm.ims.jdbc.IMSDataSource();
ds.setProperties(props);
```

DataSource オブジェクトの標準的な使用法では、システム管理者はそれを別個に作成して管理します。 DataSource オブジェクトを作成して管理するプログラムも、Java Naming and Directory Interface (JNDI) を使用して DataSource オブジェクトに論理名を割り振ります。次いで DataSource オブジェクトを使用する JDBC アプリケーションは、その論理名によりオブジェクトを参照できるため、データ・ソースに関する基礎的な情報は必要としません。さらに、システム管理者がデータ・ソースの属性を変更できるため、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。

推奨事項: ポータビリティーを最大化するには、DataSource インターフェースのみを使用して接続を取得します。

DataSource オブジェクトを使用して接続を取得するために、システム管理者が既にオブジェクトを作成し、それに論理名を割り当て済みである場合には、以下のようにします。

1. システム管理者から、接続する必要があるデータ・ソースの論理名を取得します。
2. 次の手順で使用する Context オブジェクトを作成します。 Context インターフェースは、Java Naming and Directory Interface (JNDI) の一部であり、JDBC の一部ではありません。
3. アプリケーション・プログラムで、JNDI を使用して、論理データ・ソース名に関連付けられている DataSource オブジェクトを取得します。
4. DataSource インスタンスで getConnection メソッドを使用して、接続を取得します。

次のコードは、DataSource オブジェクトを使用して接続を取得する場合に、アプリケーション・プログラムで必要とされるサンプル・コードを示しています。この例では、接続する必要のあるデータ・ソースの論理名は「jdbc/sampledb」です。

```
import java.sql.*;
import javax.naming.*;
import javax.sql.*;
```

```
...
Context ctx = new InitialContext();
DataSource ds = (DataSource)ctx.lookup("jdbc/sampledb");
Connection con = ds.getConnection();
```

関連タスク:

820 ページの『SSL サポート用の IMS Universal ドライバーの構成』

関連資料:

サポートされる javax.sql.DataSource のメソッド (アプリケーション・プログラミング API)

JDBC DriverManager インターフェースを使用した IMS データベースへの接続

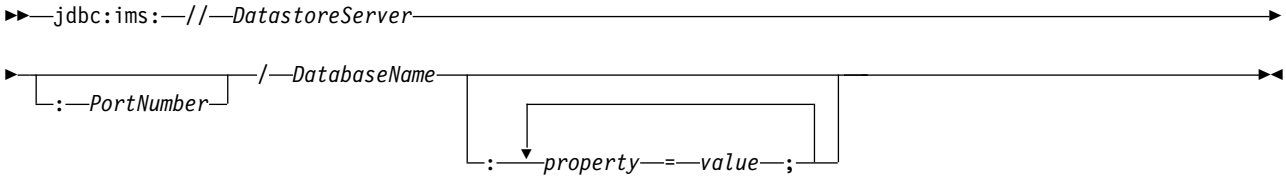
JDBC アプリケーションは、java.sql パッケージの一部である JDBC DriverManager インターフェースを使用して、データ・ソースへの接続を確立できます。

Java アプリケーションは、Class.forName メソッドを呼び出して、最初に JDBC ドライバーをロードします。アプリケーションは、ドライバーをロードした後に、DriverManager.getConnection メソッドを呼び出してデータベース・サーバーと接続します。以下に例を示します。

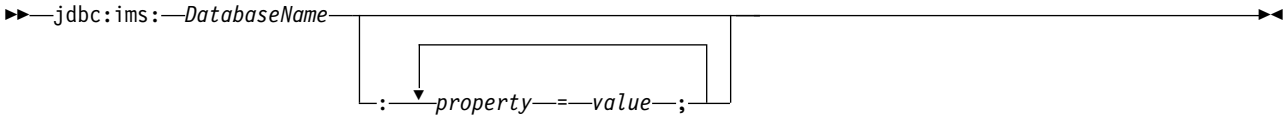
```
Connection conn = DriverManager.getConnection(url);
```

IMS Universal JDBC ドライバー・アプリケーションで DriverManager インターフェースを使用して IMS データベースに接続するには、以下のようにします。

1. 引数 com.ibm.ims.jdbc.IMSDriver を指定して Class.forName メソッドを呼び出すことで、DriverManager インターフェースを備えた IMS Universal JDBC ドライバーをロードします。
2. DriverManager.getConnection メソッドを呼び出して、IMS データベースに接続します。 URL はデータ・ソースを表し、使用している JDBC コネクティビティーのタイプを示します。
   - タイプ 4 コネクティビティーの場合は、以下の形式で URL を指定します。



   - タイプ 2 コネクティビティーの場合は、以下の形式で URL を指定します。



URL の各部の意味は次のとおりです。

jdbc:ims:

接続先が IMS データベースであることを示します。

PortNumber

IMS Connect と通信するために使用される TCP/IP サーバーのポート番号。ポート番号は、IMS Connect 構成 PROCLIB メンバーの ODACCESS ステートメントで DRDAPORT パラメーターを使用して定義されます。デフォルトのポート番号は 8888 です。タイプ 2 コネクティビティーの使用時にはこのプロパティーを設定しないでください。

MetadataURL

ターゲット IMS データベースを表すデータベース・メタデータの場所。

このプロパティーは推奨されていません。代わりに **DatabaseName** を使用してください。

MetadataURL プロパティーは、IMS Enterprise Suite Explorer for Developmentによって生成された Java メタデータ・クラスの完全修飾名です。 URL には、`class://` の接頭部を付ける必要があります (例えば、`class://com.foo.BMP255DatabaseView`)。

J2C 接続ファクトリー環境では、リソース・アダプターに指定されたデフォルト値に影響を与えずに、個別の接続の **MetadataURL** プロパティーをオーバーライドできます。

DatabaseName

ターゲット IMS データベースを表すデータベース・メタデータの場所。

メタデータが IMS カタログに保管されているのか、IMS Enterprise Suite Explorer for Development によって生成された静的メタデータ・クラスとして保管されているのかに応じて、**DatabaseName** プロパティーを以下の 2 つの方法のいずれかで指定することができます。

- IMS システムで IMS カタログを使用している場合、**DatabaseName** プロパティーは、アプリケーションがターゲット IMS データベースへのアクセスに使用する PSB の名前です。
- IMS Explorer for Development を使用している場合、**databaseName** プロパティーは、IMS Explorer for Development によって生成された Java メタデータ・クラスの完全修飾名です。URL には、`class://` の接頭部を付ける必要があります (例えば、`class://com.foo.BMP255DatabaseView`)。

J2C 接続ファクトリー環境では、リソース・アダプターに指定されたデフォルト値に影響を与えずに、個別の接続の **DatabaseName** プロパティーをオーバーライドできます。

DatastoreServer

データ・ストア・サーバー (IMS Connect) の名前または IP アドレス。ホスト名 (例えば、dev123.svl.ibm.com) または IP アドレス (例えば、192.166.0.2) のいずれかを指定できます。タイプ 2 コネクティビティーの使用時にはこのプロパティーを設定しないでください。

property

次の接続プロパティーのうちのいずれかです。

allMetadata

オプション。このプロパティーが `true` に設定された場合、DatabaseMetadata インターフェースは、IMS カタログにあるすべてのリソースに関する情報を返します。このプロパティーが `false` に設定された場合、DatabaseMetadata インターフェースは、割り振り済み PSB に関する情報を返します。このプロパティーのデフォルト値は `false` です。

datastoreName

オプション。アクセスする IMS データ・ストアの名前。

- タイプ 4 コネクティビティーを使用する場合、**`DatastoreName`** プロパティーは、ODBM に定義されているデータ・ストアの名前と一致したものであるかまたはブランクである必要があります。データ・ストア名は、DATASTORE(NAME=*name*) パラメーターまたは DATASTORE(NAME=*name,* ALIAS(NAME=*aliasname*)) パラメーターのいずれかを使用して、ODBM CSLDCxxx PROCLIB メンバー内で定義されます。別名が指定される場合、**`datastoreName`** プロパティーの値として *aliasname* を指定する必要があります。 **`DatastoreName`** の値が空白 (または指定されていない) 場合、ODBM に定義されているすべてのデータ・ストアでデータ共用が使用可能となっていると見なされるため、IMS Connect は使用可能な任意の ODBM のインスタンスに接続します。
- タイプ 2 コネクティビティーを使用する場合は、**`DatastoreName`** プロパティーを IMS サブシステムの別名に設定します。Java 従属領域のランタイムの場合は、これは設定する必要はありません。

loginTimeout

オプション。接続初期設定またはサーバー要求時にドライバーがサーバーからの応答を待つ秒数を指定します。この時間が過ぎるとタイムアウトになります。このプロパティーには、秒数として負でない整数を指定します。タイムアウトの長さを無制限にする場合は、このプロパティーを 0 に設定してください。タイプ 2 コネクティビティーの使用時にはこのプロパティーを設定しないでください。

password

RACF 管理者によって提供された IMS Connect に接続するためのパスワード。タイプ 2 コネクティビティーの使用時にはこのプロパティーを設定しないでください。

sslConnection

オプション。この接続がデータ暗号化用に Secure Sockets Layer (SSL) を使用するかどうかを示します。 SSL を使用可能にするにはこのプロパティーを「true」に設定します。使用しな

い場合は「false」を設定します。タイプ 2 コネクティビティーの使用時にはこのプロパティーを設定しないでください。

sslKeyStoreType
オプション。セキュア・ソケット接続の確立に必要な暗号オブジェクトを含むファイルの形式を指定します。有効な値は、「JKS」および「PKCS12」です。この値は、**sslConnection** が「true」に設定されていて、**sslKeyStoreType** が指定されていない場合にのみ使用します。 **sslKeyStoreType** パラメーターのデフォルトは「JKS」です。

sslSecureSocketProtocol
オプション。新しい接続の暗号通信プロトコルを指定します。サーバーがサポートしているプロトコルで、最高レベルのセキュリティーを提供するプロトコルを指定します。有効な値は、「SSL」、「SSLv3」、「TLSv1.1」、および「TLSv1.2」です。この値は、**sslConnection** が「true」に設定されている場合のみ使用します。**sslConnection** が「true」に設定されていて、**sslSecureSocketProtocol** が指定されていない場合、デフォルト・プロトコルは JRE とサーバーによって実行時に決定されます。

sslTrustStoreLocation
オプション。新しい接続の暗号トラストストア・ファイルのロケーションを指定します。この値は、**sslConnection** が true に設定されている場合のみ使用します。

sslTrustStorePassword
オプション。暗号トラストストア・ファイルにアクセスするためのパスワードを指定します。この値は、**sslConnection** が true に設定されている場合のみ使用します。

sslKeyStoreLocation
オプション。新しい接続の暗号鍵ストア・ファイルのロケーションを指定します。この値は、**sslConnection** が true に設定されている場合のみ使用します。

sslKeyStorePassword
オプション。暗号鍵ストア・ファイルにアクセスするためのパスワードを指定します。この値は、**sslConnection** が true に設定されている場合のみ使用します。

user RACF 管理者によって提供された IMS Connect に接続するためのユーザー名。タイプ 2 コネクティビティーの使用時にはこのプロパティーを設定しないでください。

signedCompare

オプション。このプロパティーが「true」に設定された場合、署名されたデータ・タイプについての範囲が指定された照会をサポートするために特殊 SSA が生成されます。プロパティーが「false」に設定された場合は、データ・タイプ値のバイナリー表現に基づいて標準バイナリー比較が実行されます。この値を「false」に設定すると、パフォーマンスは上がりますが、誤った結果がもたらされる可能性があります。このプロパティーのデフォルト値は「true」です。

flattenTables

オプション。データベース表をフラット化されたビューで生成するかどうかを示します。値を true にすると、表の追加の列として STRUCT または ARRAY のサブエレメントが公開されます。デフォルト値は false です。

- IMS Explorer は、コピーブックをインポートする際にコピーブックの構造をフラット化します。IMS カタログ内ではコピーブック自体は変更されないままですが、その特定の接続についての各表の構造に関する情報は変更されます。
- flattenTables プロパティーを指定すると、複合構造内のフィールドを直接照会できます。複合構造のフラット化のサポートについて詳しくは、 709 ページの『複合構造のフラット化のサポート』を参照してください。

制約事項**:** flattenTables 接続プロパティーでは、静的配列と静的構造のみがサポートされます。動的配列は変更されません。

t2OutputBufferSize

オプション。タイプ 2 接続の場合の SELECT 操作からの結果をバイト単位で表した出力バッファーのサイズ。

t2OutputBufferSize の最小値は 500000 です。500000 より小さい値が設定された場合、このプロパティー値は 500000 に調整されます。最大限界はありません。デフォルト値は 1280000 です。

treatInvalidDecimalAsNull

オプション。Java アプリケーションで無効と見なされる特定の 10 進値 (無効な符号ビットが設定された PACKEDDECIMAL および ZONEDDECIMAL など) をヌルとして解釈するかどうかを指示します。デフォルトでは、このプロパティーは「false」であるため、Java アプリケーションが無効値を処理しているときには変換例外がスローされます。

currentSchema

オプション。動的に準備される SQL ステートメントで非修飾データベース・オブジェクトを修飾するために使用される、デフォルトのスキーマ名を指定します。

dbViewLocation

オプション。databaseView メタデータ・クラスへの絶対パスを

指定します。このプロパティーを使用して、プロジェクト・パスに配置されていないメタデータ・クラスを組み込むことができます。

dpsbOnCommit
オプション。コミットの発生時に PSB の割り振りを解除するには、このプロパティーを **true** に設定します。

推奨事項: 統合接続プールを使用する管理対象環境の場合を除き、このプロパティーを **true** に設定しないでください。

fetchSize
オプション。さらに多くの行が必要な場合に、データベースから取得する行数に関するヒントをクライアントに示します。このプロパティーに指定された数値は、現行接続でリトリーブされたデータのみに影響します。指定された値が 0 の場合、利用可能なすべての行が返されます。

管理対象接続および管理対象外接続のいずれも、このプロパティーのデフォルト値は 0 です。

llField
オプション。このプロパティーを **true** に設定すると、LL フィールドのデータは結果セットに通常の列として表示されます。LL フィールドの値を変更して、可変長セグメントのインスタンス長を変更できます。

maxRows
オプション。照会結果セットで返す最大行数を指定します。デフォルト値は 0 で、結果セットで利用可能なすべての行を返します。

traceFile
オプション。接続するトレース・ファイルの名前を指定します。

traceFileAppend
オプション。指定のトレース・ファイルが存在する場合、このプロパティーを true に設定すると、既存のトレース・ファイルを上書きせずに、そのトレース・ファイルに新規接続のトレース・データを追加する必要があることを指定します。

traceFile に値が指定されていない場合、このプロパティーは無視されます。

traceDirectory
オプション。トレース・ファイルが入っているファイル・システム・ディレクトリーを指定します。デフォルトでは、このパスはアプリケーションを実行するディレクトリーです。

traceFile に値が指定されていない場合、このプロパティーは無視されます。

traceLevel

オプション。接続に使用可能なトレースを指定します。このプロパティーの有効値は、IMSDataSource クラスの Java API 文書で定義されています。

デフォルトでは、すべてのトレースが使用不可になっています。

traceFile に値が指定されていない場合、このプロパティーは無視されます。

トレース・レベル	**traceLevel** パッケージの値	**IMSDataSource** 内の **traceLevel** 定数フィールド	**traceLevel** の **10** 進値
すべて	com.ibm.ims.db.opendb.*	TRACE_ALL	-1
DL/I	com.ibm.ims.db.opendb.dli.*	TRACE_DLI	28
DRDA	com.ibm.ims.db.opendb.drda.*	TRACE_DRDA	1
JDBC	com.ibm.ims.db.opendb.jdbc.*	TRACE_JDBC	32
Java EE	com.ibm.ims.opendb.spi.* com.ibm.ims.db.opendb.cci.*	TRACE_JEE	192

value　接続プロパティーの有効な値。

sslConnection および **loginTimeout** プロパティーを設定するには、java.util.Properties オブジェクトを使用します。例えば、以下のサンプル・コードは、SSL を使用可能にし、timeout 値を 10 秒に設定する方法を示しています。

```
Properties props = new Properties();
props.put("sslConnection", "true");
props.put("timeout", "10");
```

3. タイプ 4 コネクティビティーの場合は、接続 URL、パラメーター、または java.util.Properties オブジェクトのうちのいずれかの方法で、ユーザー ID とパスワードを指定する必要があります。 パラメーターによって接続用のユーザー ID とパスワードを設定するには、*user* および *password* を指定する getConnection メソッドの形式を使用します。以下に例を示します。

```
String url =
"jdbc:ims://tst.svl.ibm.com:8888/class://BMP2.BMP2DatabaseView";
String user = "MyUserID";
String password = "MyPassword";
Connection conn = DriverManager.getConnection(url, user, password);
```

java.util.Properties オブジェクトによって接続用のユーザー ID とパスワードを設定するには、java.util.Properties オブジェクトを指定する getConnection メソッドの形式を使用します。以下に例を示します。

```
Properties props = new Properties();
props.put( "user", "MyUserID" );
props.put( "password", "MyPassword" );
String url =
"jdbc:ims://tst.svl.ibm.com:8888/class://BMP2.BMP2DatabaseView";
Connection conn = DriverManager.getConnection(url, props);
```

4. アプリケーションが接続を終了した後に、Connection インターフェースで close メソッドを使用して接続を閉じます。

以下のサンプル・コードでは、DriverManager インターフェースを使用して IMS Universal JDBC ドライバー・アプリケーションから IMS データベースへのタイプ 4 接続を作成する方法を示しています。

```
Connection conn = null;

// Create Properties object
Properties props = new Properties();

// Enable SSL for connection
props.put("sslConnection", "true");

// Set datastoreName for connection
props.put( "datastoreName", "IMS1" );

// Set timeout for connection
props.put("loginTimeout", "10");

// Set user ID for connection
props.put( "user", "myUserID" );

// Set password for connection
props.put( "password", "myPassword" );

// Set URL for the data source
Class.forName("com.ibm.ims.jdbc.IMSDriver");

// Create connection
conn = DriverManager.getConnection
("jdbc:ims://tst.svl.ibm.com:8888/class://BMP2.BMP2DatabaseView",
props);
```

別の方法として、URL で接続プロパティーを指定することもできます。以下に例を示します。

```
String url=""jdbc:ims://tst.svl.ibm.com:8888/class://"
   + "BMP2.BMP2DatabaseView:datastoreName=IMS1;"
   + "loginTimeout=10;sslConnection=true;user=myUserID;password=myPassword;";
Connection conn = DriverManager.getConnection(url);
```

関連タスク:

820 ページの『SSL サポート用の IMS Universal ドライバーの構成』

関連情報:

https://imsinsiders.wordpress.com/2016/04/18/how-to-enable-the-ims-jdbc-trace

IMS Universal JDBC ドライバーのサンプル・アプリケーション

次の Java サンプル・アプリケーションは、IMS Universal JDBC ドライバーを使用した JDBC アプリケーションの基本的なプログラミングの流れを示しています。

以下の例では、IMS データベースに接続し、SQL SELECT 照会を使用して患者名のリストをリトリーブし、SQL UPDATE 照会を使用して患者情報を変更しています。

```
package client;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
```

```
import com.ibm.ims.jdbc.IMSDataSource;

public class JDBCSample {


    public static void main(String[] args)
       throws SQLException {
        IMSDataSource ds = new IMSDataSource();
        ds.setDatabaseName("MYPSB");
        ds.setDatastoreName("IMS1");
        ds.setDatastoreServer("ec0123.my.host.com");
        ds.setPortNumber(5555);
        ds.setDriverType(IMSDataSource.DRIVER_TYPE_4);
        ds.setUser("myUserId");
        ds.setPassword("myPassword");

        Connection conn = null;

        try {
            conn = ds.getConnection();

            Statement st = conn.createStatement();

            // List all of the patient names in the
            // SURG ward in the ALEXANDRIA hospital
            ResultSet rs = st.executeQuery("SELECT patname from " +
                    "pcb01.hospital, ward, patient " +
                    "where hospital.hospname = 'ALEXANDRIA' " +
                    "and ward.wardname = 'SURG'");
            while (rs.next()) {
                System.out.println(rs.getString("patname"));
            }

            // Update the name of the patient with patient
            // number 0222 in ward 04 in the hospital
            // with code R1210010000A
            int updatedRecords = st.executeUpdate("UPDATE PCB01.PATIENT " +
               "SET PATNAME='UPDATED NAME' WHERE PATNUM='0222' " +
               "AND HOSPITAL_HOSPCODE='R121001000A' AND WARD_WARDNO='04'");
            System.out.println("Updated " + updatedRecords + " Record(s)");

            conn.commit();
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
            if (!conn.isClosed()) {
                conn.rollback();
                conn.close();
            }
        }
    }
}
```

IMS Universal JDBC ドライバーを使用した IMS データベースにアクセスする SQL 照会の作成

IMS Universal JDBC ドライバーを使用して、IMS データベースに接続し、SQL 照会を作成します。

IMS カタログは、アプリケーション・プログラムにメタデータを提供します。SQL 照会を作成することで、カタログ・データベースで使用可能なメタデータ情報に基づいて、IMS データにアクセスするできます。

IMS システムで IMS カタログを使用していない場合は、代わりに IMS Enterprise Suite Explorer for Developmentを使用できます。

IMS Explorer for Developmentは、Java データベースのメタデータ・クラスを生成します。このクラスには、IMS Universal ドライバー が実行時に使用する PSB および DBD メタデータ・クラスが含まれます。

メタデータには、セグメント、セグメント名、セグメントの階層、フィールド、フィールド・タイプ、フィールド名、フィールド・オフセット、およびフィールド長など、IMS データベースに関する情報が入っています。

メタデータは、プログラム仕様ブロック (PSB) の割り振り、DL/I 呼び出しの発行、データ形式変更の実行、および SQL 照会の DL/I 呼び出しへの変換を行うために、IMS Universal JDBC ドライバーによって使用されます。

次の表は、IMS データベース・エレメントとリレーショナル・データベース・エレメントとの間のマッピングを要約しています。

表 *101. IMS* データベース・エレメントとリレーショナル・データベース・エレメントとの間のマッピング

IMS の階層データベース・エレメント	同等のリレーショナル・データベース・エレメント
セグメント名	テーブル名
セグメント・インスタンス	テーブルの行
セグメント・フィールド名	列名
セグメントのユニーク・キー	テーブルの主キー
外部キー・フィールド	テーブルの外部キー

関連概念:

IMS Explorer for Development の概要

IMS JDBC ドライバーでサポートされている SQL キーワード

IMS Universal JDBC ドライバーが提供する SQL サポートは、リレーショナル・データベース管理システム用の SQL-92 標準をベースとしています。

PCB、セグメント、またはフィールドの名前として SQL キーワードを使用する場合、JDBC アプリケーション・プログラムは、SQL 照会を試行するとエラーを throw します。これらのキーワードでは大文字小文字は区別されません。

IMS JDBC ドライバーでサポートされる SQL キーワードは、以下のとおりです。

ABS
ACOS
ALL
ALTER
AND
AS
ASC
ASIN
ATAN

ATAN2
AVG
BETWEEN
CEIL
CEILING
COS
COSH
COT
COUNT
CREATE
DEGREES
DELETE
DESC
DISTINCT
DROP
EXP
FETCH
FIRST
FLOOR
FROM
GROUP BY
INNER
INSERT
INTO
JOIN
LN
LOG
LOG10
MAX
MIN
MOD
NULL
ON
ONLY
OR
ORDER BY
POWER
RADIANS
ROW
ROWS
SELECT
SET
SIGN
SIN
SINH
SQRT
SUM
TAN

TANH
UPDATE
VALUES
WHERE

関連資料:

765 ページの『IMS Universal JDBC ドライバーにより制限されるポータブル SQL キーワード』

IMS JDBC ドライバーでサポートされている SQL 集約関数

IMS Universal JDBC ドライバーでは、SQL 集約関数および関連キーワードがサポートされています。

- AS
- AVG
- COUNT
- GROUP BY
- MAX
- MIN
- ORDER BY
  - ASC
  - DESC
- SUM

集約関数および ORDER BY 節と GROUP BY 節での ResultSet タイプは、常に TYPE_SCROLL_INSENSITIVE となります。

次の表は、集約関数で受け入れられるフィールドのデータ型と、ResultSet での結果のデータ型を示しています。

表 *102.* サポートされている *SQL* 集約関数とそれらがサポートするデータ型

機能	引数の型	結果の型
SUM および AVG	バイト	Long
	Short	Long
	Integer	Long
	Long	Long
	BigDecimal	倍精度浮動小数点
	単精度浮動小数点	倍精度浮動小数点
	倍精度浮動小数点	倍精度浮動小数点
MIN および MAX	BIT、BLOB、または BINARY 以外の任意の型	引数の型と同じ
COUNT	任意の型	Long

集約関数によって生成される列名

集約関数での ResultSet 列名は、下線文字 (_) で区切られた集約関数名とフィールド名の組み合わせです。例えば、ステートメント `SELECT MAX(age)` の場合は、結果

は列名 MAX_age となります。以降のすべての参照において、この列名を使用します (例えば `resultSet.getInt("MAX_age")` など)。

集約関数の引数フィールドはテーブル修飾され、ResultSet 列名は下線文字 (_) で区切られた集約関数名、テーブル名、および列名の組み合わせとなります。例えば、`SELECT MAX(Employee.age)` の場合は、結果は列名 MAX_Employee_age となります。

AS 節の使用

AS キーワードを使用して、結果セット内の集約関数列、または SELECT ステートメントの他の任意のフィールドの名前を変更できます。 AS キーワードを使用して FROM 節内のテーブルの名前を変更することはできません。 AS キーワードを使用して列の名前を変更した場合は、列の参照にこの新しい名前を使用する必要があります。例えば、`SELECT MAX(age) AS oldest` を指定した場合は、その集約関数列に対するそれ以降の参照は `resultSet.getInt("oldest")` とします。

IMS Universal JDBC ドライバーを使用しており、AS 節で名前変更された列を使用して SELECT 照会を指定した場合、結果の ResultSet 内のフィールドは AS による変更名でのみ参照できます。ただし、SELECT 照会の残りの部分 (WHERE、ORDER BY、および GROUP BY 節内) では、元の列名または AS による変更名のどちらでも使用できます。

ORDER BY 節および GROUP BY 節の使用

重要: GROUP BY 節または ORDER BY 節で指定されるフィールド名は、SELECT ステートメントで指定されるフィールド名と正確に一致する必要があります。

IMS Universal JDBC ドライバーを使用する場合、ORDER BY 節および GROUP BY 節を使用する以下のような照会は有効です。

```
SELECT HOSPNAME, COUNT(PATNAME) AS PatCount FROM PCB01.HOSPITAL, PATIENT
GROUP BY HOSPNAME ORDER BY HOSPNAME
```

```
SELECT HOSPNAME, COUNT(DISTINCT PATNAME) AS PatCount FROM PCB01.HOSPITAL,
PATIENT GROUP BY HOSPNAME ORDER BY HOSPNAME
```

COUNT 関数との DISTINCT の使用

IMS Universal JDBC ドライバーを使用する場合は、COUNT 集約関数を DISTINCT キーワードで修飾できます。例えば、次の照会は、昇順でリストされたすべての病院名を、その病院の個別の患者名の数とともに返します。 COUNT 集約関数は、列名 COUNT_DISTINCT_PATNAME を生成します。

```
SELECT HOSPNAME, COUNT(DISTINCT PATNAME)FROM PCB01.HOSPITAL, PATIENT
GROUP BY HOSPNAME ORDER BY HOSPNAME
```

IMS Universal JDBC ドライバーにより制限されるポータブル SQL キーワード

次の SQL キーワードのいずれかを PCB 名、セグメント名、またはフィールド名として使用している場合、JDBC アプリケーションで SQL 照会を実行しようとするとエラーを受け取ります。代わりに、IMS Enterprise Suite Explorer for Development の別名割り当て機能を使用します。これらのキーワードでは、大/小文字の区別はありません。

以下の表に示すキーワードは、予約済みの SQL キーワードです。

ABORT から **CROSS** まで	**CURRENT** から **IS** まで	**JOIN** から **REAL** まで	**REFERENCES** から **WORK** まで
ABORT	CURRENT	JOIN	REFERENCES
ANALYZE	CURSOR	LAST	RESET
AND	DECIMAL	LEADING	REVOKE
ALL	DECLARE	LEFT	RIGHT
ALLOCATE	DEFAULT	LIKE	ROLLBACK
ALTER	DELETE	LISTEN	ROW
AND	DESC	LOAD	ROWS
ANY	DISTINCT	LOCAL	SELECT
ARE	DO	LOCK	SET
AS	DOUBLE	MAX	SETOF
ASC	DROP	MIN	SHOW
ASSERTION	END	MOVE	SMALLINT
AT	EXECUTE	NAMES	SUBSTRING
AVG	EXISTS	NATIONAL	SUM
BEGIN	EXPLAIN	NATURAL	TABLE
BETWEEN	EXTRACT	NCHAR	TO
BINARY	EXTEND	NEW	TRAILING
BIT	FALSE	NO	TRANSACTION
BOOLEAN	FETCH	NONE	TRIM
BOTH	FIRST	NOT	TRUE
BY	FLOAT	NOTIFY	UNION
CASCADE	FOR	NULL	UNIQUE
CAST	FOREIGN	NUMERIC	UNLISTEN
CHAR	FROM	ON	UNTIL
CHARACTER	FULL	ONLY	UPDATE
CHECK	GRANT	OR	USER
CLOSE	GROUP	ORDER	USING
CLUSTER	HAVING	OUTER	VACUUM
COLLATE	IN	PARTIAL	VALUES
COLUMN	INNER	POSITION	VARCHAR
COMMIT	INSERT	PRECISON	VARYING
CONSTRAINT	INT	PRIMARY	VERBOSE
COPY	INTERVAL	PRIVILEGES	VIEW
COUNT	INTERVAL	PROCEDURE	WHERE
CREATE	INTO	PUBLIC	WITH
CROSS	IS	REAL	WORK

関連資料:

762 ページの『IMS JDBC ドライバーでサポートされている SQL キーワード』

IMS Universal JDBC ドライバーを使用して IMS リソースを変更する DDL ステートメントの作成

IMS システム内のアクティブな DBD または PSB を作成または変更する Java アプリケーションを作成できます。

1. Universal ドライバーを使用して、IMS システムへの接続を作成します。
2. ステートメントを作成し、DDL クエリーを実行します。
3. Commit DDL というクエリーを実行します。 以下に例を示します。

```
ds = FVTConnectionFactory.getIMSDataSource(alias, driverType,
host, port, userName,

password, url);
con = ds.getConnection();



Statement st = con.createStatement();
st.executeUpdate("CREATE DATABASE MYDB ACCESS SHSAM CCSID 'Cp1047'

VERSION '2.0'");
st.executeUpdate("CREATE TABLE MYTABLE (COLUMN1 DECIMAL(5,2)

INTERNALNAME COLUMN1 TYPE C BYTES 10 START 1) AMBIGUOUS INSERT LAST IN MYDB");
st.executeUpdate("CREATE TABLESPACE tb1 IN MYDB BLOCK PRIMARY

32768");
st.executeUpdate("COMMIT DDL");
```

IMS Universal JDBC ドライバーを使用した SQL ステートメントの使用法

次の使用規則は、IMS Universal JDBC ドライバーを使用する、IMS に渡される SQL ステートメントに適用されます。

外部キー・フィールド:

リレーショナル・データベースでは、表間の外部キー関係を作成することによって、論理的に階層を構築できます。 IMS では、階層は明示的であり、データベース定義自体の一部です。 IMS Universal JDBC ドライバーでは、リレーショナルな意味でこれらの明示的階層を言い表す外部キー の概念を導入しています。これによって IMS 用の SQL 構文は標準 SQL と同等なものになります。

IMS Universal JDBC ドライバーを使用して IMS データベースにアクセスする場合、階層パス内でルート表ではないすべての表には、データベースのルートにまで至るそのすべての親セグメントのユニーク・キーが仮想的に含まれています。これらのキーは、外部キー・フィールド と呼ばれます。

副次索引を持つセグメントでは、副次索引もそのセグメントの主キーになります。セグメントに対応する外部キーは、副次索引の名前から派生します。

制約事項: 副次索引は SELECT ステートメント内では参照できません。副次索引を参照できるのは、WHERE 節内のみです。

外部キー・フィールドの目的は、リレーショナル・データベースの外部キーと同様に、参照整合性を維持することです。これにより、階層パス内にある特定の表および列に対して、SQL SELECT、INSERT、UPDATE、および DELETE 照会を作成できます。

要確認: 外部キーは、IMS Universal JDBC ドライバーによって内部的に維持されます。このキーは、IMS データベース内に物理的に保管されているわけではありません。

副次索引を持たない病院データベースの例

例えば、病院データベースでは、HOSPITAL、WARD、および PATIENT 表は同じ階層パス上にあります。 JDBC アプリケーションでは、表には次の列が含まれて表示されます。

HOSPITAL 表

列は次のとおりです。

- HOSPNAME
- HOSPCODE (主キー)

WARD 表

列は次のとおりです。

- WARDNO (主キー)
- WARDNAME
- PATCOUNT
- NURCOUNT
- DOCCOUNT
- HOSPITAL_HOSPCODE (HOSPITAL 表の HOSPCODE 列を参照する外部キー・フィールド)

PATIENT 表

列は次のとおりです。

- PATNUM (主キー)
- PATNAME
- WARD_WARDNO (WARD 表の WARDNO 列を参照する外部キー・フィールド)
- HOSPITAL_HOSPCODE (HOSPITAL 表の HOSPCODE 列を参照する外部キー・フィールド)

次の照会は、前のデータベースの例に基づいて、SQL SELECT ステートメントで外部キーを使用する方法を示しています。次のステートメントは、階層パスで HOSPITAL および WARD の下の子セグメントから派生した PATIENT 表のすべての列をリトリーブします。

```
SELECT * FROM PCB01.PATIENT
WHERE HOSPITAL_HOSPCODE = 'H5140070000H'
   AND WARD_WARDNO = '0023'
```

次の例は、外部キーを使用した INSERT ステートメントを示しています。

```
INSERT INTO PCB01.PATIENT (PATNUM, PATNAME,
WARD_WARDNO, HOSPITAL_HOSPCODE)
VALUES ('00345', 'John Doe', '0023', 'H514007000H')
```

次のステートメントは、WARD 表から病院コードとすべての病棟名をリトリーブします。これらのステートメントはすべて同等です。

```
SELECT HOSPITAL.HOSPCODE, WARD.WARDNAME
FROM PCB01.HOSPITAL, PCB01.WARD

SELECT HOSPITAL_HOSPCODE, WARD.WARDNAME
FROM PCB01.WARD

SELECT WARD.HOSPITAL_HOSPCODE, WARD.WARDNAME
FROM PCB01.WARD

SELECT HOSPITAL_HOSPCODE, WARDNAME
FROM PCB01.WARD
```

次のステートメントは、HOSPITAL 表に HOSPITAL_HOSPCODE 列が存在しないため、失敗します。

```
SELECT HOSPITAL_HOSPCODE FROM PCB01.HOSPITAL
```

副次索引を持つ病院データベースの例

次の例は、HOSPITAL 表に副次索引がある場合に、前の例がどのように変化するかを示しています。

HOSPITAL 表

列は次のとおりです。

- HOSPNAME (副次索引と主キー)
- HOSPCODE

WARD 表

列は次のとおりです。

- WARDNO (主キー)
- WARDNAME
- PATCOUNT
- NURCOUNT
- DOCCOUNT
- HOSPITAL_HOSPNAME (HOSPITAL 表の HOSPNAME 副次索引を参照する外部キー・フィールド)

PATIENT 表

列は次のとおりです。

- PATNUM (主キー)
- PATNAME
- WARD_WARDNO (WARD 表の WARDNO 列を参照する外部キー・フィールド)
- HOSPITAL_HOSPNAME (HOSPITAL 表の HOSPNAME 副次索引を参照する外部キー・フィールド)

CREATE ステートメントの使用法**:**

CREATE ステートメントは、IMS 内のリソースを作成するために使用されます。

各 CREATE コマンドには、個別に独自のキーワード・リストを使用できます。以下の CREATE コマンドを発行できます。

- CREATE DATABASE (アプリケーション・プログラミング API)
- CREATE TABLE (アプリケーション・プログラミング API)
- CREATE TABLESPACE (アプリケーション・プログラミング API)
- CREATE PROGRAMVIEW (アプリケーション・プログラミング API)

例

CREATE DATABASE ステートメントは、以下のように発行します。

```
CREATE DATABASE MYDB ACCESS HIDAM VSAM LIKE BASEDB CCSID 'UTF-8'
DATA CAPTURE CHANGES (EXIT1

NOCASCADE  DATA INPOS PATH LOG, EXIT2 NOCASCADE  DATA INPOS PATH LOG)
VERSION '1.8' DATXEXITNO
```

ALTER ステートメントの使用法**:**

ALTER ステートメントは、IMS 内のリソースを変更するために使用されます。

各 ALTER コマンドには、個別に独自のキーワード・リストを使用できます。以下の ALTER コマンドを発行できます。

- ALTER DATABASE (アプリケーション・プログラミング API)
- ALTER TABLE (アプリケーション・プログラミング API)
- ALTER TABLESPACE (アプリケーション・プログラミング API)

例

ALTER DATABASE ステートメントは、以下のように発行します。

```
ALTER DATABASE MYDB CCSID 'Cp1047' VERSION '1.8' DATXEXITYES
```

注**:** この ALTER ステートメントは、指定されたキーワードだけを変更します。他の当初に定義されたキーワードは、ALTER ステートメントで明示的に指定された場合を除いて、どれも変更されません。

DROP ステートメントの使用法**:**

DROP ステートメントは、IMS 内のリソースを削除するために使用します。

各 DROP コマンドには、個別に独自のキーワード・リストを使用できます。以下の DROP コマンドを発行できます。

- DROP DATABASE (アプリケーション・プログラミング API)
- DROP PROGRAMVIEW (アプリケーション・プログラミング API)
- DROP TABLE (アプリケーション・プログラミング API)
- DROP TABLESPACE (アプリケーション・プログラミング API)

例

DROP DATABASE ステートメントは、以下のように発行します。

```
DROP DATABASE MYDB
```

SELECT ステートメントの使用法**:**

SELECT ステートメントは、1 つ以上の表からデータをリトリーブするために使用します。結果は、表形式の結果セットで返されます。

IMS Universal JDBC ドライバーで SELECT ステートメントを使用する場合は、以下のことに注意してください。

- 複数の表から選択を実行する場合に、これらの表の 1 つ以上に同じ列名が存在する場合は、列を表名で修飾する必要があります。そうしないと、あいまいさのエラーが発生します。
- FROM 節では、データを選択するすべての表をリストする必要があります。FROM 節でリストされた表は、IMS データベースの同じ階層パスに存在している必要があります。
- IMS JDBC ドライバーを使用する Java アプリケーションでは、PSB に接続します。 PSB 内には複数のデータベース PCB があるため、照会では PSB 内のどの PCB を使用するかを指定する必要があります。使用する PCB を指定するには、SQL ステートメントの FROM 節で参照されるセグメントを、セグメント名の前に PCB 名を付けて必ず修飾します。 PSB に 1 つしか PCB が含まれていない場合は、PCB 名を省略しても構いません。

有効な **IMS Universal JDBC** ドライバー **SELECT** 照会の例

指定した列を選択

次のステートメントは、WARD 表と PATIENT 表から、それぞれ病棟名と患者名をリトリーブします。

```
SELECT WARD.WARDNAME,PATIENT.PATNAME
FROM PCB01.WARD, PATIENT
```

* 記号を指定してすべての列を選択

次のステートメントは、PATIENT 表のすべての列をリトリーブします。

```
SELECT *
FROM PCB01.PATIENT
```

次のステートメントは、HOSPITAL 表から病院名、および WARD 表からすべての列をリトリーブします。

```
SELECT HOSPITAL.HOSPNAME, WARD.*
FROM PCB01.HOSPITAL, PCB01.WARD
```

DISTINCT を指定して選択

次のステートメントは、PATIENT 表からすべての個別の患者名をリトリーブします。

```
SELECT DISTINCT PATNAME
FROM PCB01.PATIENT
```

ORDER BY を指定して選択

ORDER BY 節は、行をソートするために使用します。デフォルトでは、結果は昇順の番号順またはアルファベット順でソートされます。次のステートメントは、すべての個別の病院名をアルファベット順でソートしてリトリーブします。

```
SELECT DISTINCT HOSPNAME FROM PCB01.HOSPITAL
   ORDER BY HOSPNAME
```

次のステートメントは、すべての病棟名をアルファベット順でソートし、各病棟の患者番号を昇順の番号順でソートしてリトリーブします。 2 つの WARDNAME 値が ORDER BY 比較で同じである場合、対応する PATCOUNT 値によって同等比較を行います (この場合は、対応する PATCOUNT 値が小さい数値である行が先に表示されます)。

```
SELECT WARDNAME, PATCOUNT FROM PCB01.WARD
   ORDER BY WARDNAME, PATCOUNT
```

照会結果を降順の番号順または逆のアルファベット順でソートする場合は、DESC 修飾子を使用します。次のステートメントは、すべての患者名を逆のアルファベット順でリトリーブします。

```
SELECT PATNAME FROM PCB01.PATIENT
   ORDER BY PATNAME DESC
```

照会結果を昇順の番号順または逆のアルファベット順で明示的にソートする場合は、ASC 修飾子を使用します。次のステートメントは、すべての病棟名を昇順のアルファベット順でソートし、各病棟の患者番号を降順の番号順でソートしてリトリーブします。

```
SELECT WARDNAME, PATCOUNT FROM PCB01.WARD
   ORDER BY WARDNAME ASC, PATCOUNT DESC
```

GROUP BY を指定して選択

GROUP BY 節は、個別の列値によってグループ化される、集約関数の結果セットを返すために使用します。次のステートメントは、個別の病棟名によってグループ化された、病院内の各病棟のすべての医師の集合の合計を返します。

```
SELECT WARDNAME, SUM(DOCCOUNT)
FROM PCB01.WARD
WHERE HOSPITAL_HOSPCODE = 'H5140070000H
   GROUP BY WARDNAME
```

次のステートメントは、病院名、病棟名、および各病院の各病棟のすべての患者数を、個別の病院名でグループ化し、病棟名でサブグループ化して返します。

```
SELECT HOSPNAME, WARDNAME, COUNT(PATNAME)
FROM PCB01.HOSPITAL, WARD, PATIENT
   GROUP BY HOSPNAME, WARDNAME
```

AS 節の使用

結果セットの集約関数列または SELECT ステートメントの他のフィールドを名前変更するには、AS 節を使用します。次のステートメントは、PATIENT 表の個別の患者の集約数に、別名「PATIENTCOUNT」を付けて返します。

```
SELECT COUNT(DISTINCT PATNAME)
   AS PATIENTCOUNT
FROM PCB01.PATIENT
```

次のステートメントは、すべての病院の個別の病棟の集約数に別名「WARDCOUNT」を付け、病院名をアルファベット順でソートし、さらに個別の病院名 (名前変更した列の別名「HOSPITALNAME」) でグループ化して返します。

```
SELECT HOSPNAME AS HOSPITALNAME, COUNT(DISTINCT WARDNAME)
   AS WARDCOUNT
FROM PCB01.HOSPITAL, WARD
   GROUP BY HOSPNAME
   ORDER BY HOSPNAME
```

INSERT ステートメントの使用法**:**

INSERT ステートメントは、表に新規行を挿入するために使用します。

外部キー・フィールドにより、IMS Universal JDBC ドライバーは、標準の SQL 処理を使用して階層パス内に挿入する新規レコード (またはセグメント・インスタンス) を正しく配置できます。これはリレーショナル・データベースの外部キーと似ています。非ルート・レベルで表にレコードを挿入する場合は、表のすべての外部キー・フィールドの値を指定する必要があります。

有効な **IMS Universal JDBC** ドライバー **INSERT** ステートメントの例

ルートにデータを挿入

次のステートメントは、新規 HOSPITAL レコードを挿入します。

```
INSERT INTO PCB01.HOSPITAL (HOSPCODE, HOSPNAME)
VALUES ('R1210050000A', 'O''MALLEY CLINIC')
```

階層パス内の指定された表にデータを挿入

非ルート・レベルで表にレコードを挿入する場合は、表のすべての外部キー・フィールドの値を指定する必要があります。次のステートメントは、新規 ILLNESS レコードを特定の HOSPITAL、WARD、および PATIENT 表に挿入します。この例では、ILLNESS 表には、HOSPITAL_HOSPCODE、WARD_WARDNO、および PATIENT_PATNUM の 3 つの外部キーがあります。新規レコードは、HOSPITAL 表に値が 'H5140070000H' の HOSPCODE が存在し、WARD 表に値 '01' が存在し、および PATIENT 表に 'R1210050000A' の PATNUM 値が存在する場合にのみ挿入されます。

```
INSERT INTO PCB01.ILLNESS (HOSPITAL_HOSPCODE, WARD_WARDNO,
   ILLNAME, PATIENT_PATNUM)
VALUES ('H5140070000H', '01', 'COLD', 'R1210050000A')
```

次のステートメントは、新規 WARD レコードを特定の HOSPITAL 表に挿入します。この例では、WARD 表には外部キー HOSPITAL_HOSPCODE があります。新規レコードは、HOSPITAL 表に、値が 'H5140070000H' の HOSPCODE が存在する場合にのみ挿入されます。

```
INSERT INTO PCB01.WARD (WARDNO, HOSPITAL_HOSPCODE, WARDNAME)
VALUES ('0001', 'H5140070000H', 'EMGY')
```

サブフィールドがある検索可能フィールドにデータを挿入

検索可能フィールドがサブフィールドで構成されている場合、すべてのサブフィールドの値を設定してデータを挿入することで、検索可能フィールドに完全に値を入力できます。

無効な **IMS Universal JDBC** ドライバー **INSERT** ステートメントの例

外部キー・フィールドを指定しないで非ルート・レベルでレコードを挿入

このステートメントでは、WARD_WARDNO 外部キー・フィールドが欠落

しています。この照会は、すべての外部キーに有効な値を指定しなければならない参照整合性の制約に違反しているため、失敗します。

```
INSERT INTO PCB01.PATIENT (HOSPITAL_HOSPCODE, PATNAME, PATNUM)
VALUES ('HW3201', 'JOHN O''CONNER', 'Z800')
```

UPDATE ステートメントの使用法**:**

UPDATE ステートメントは、表のデータを変更するために使用します。

有効な **IMS Universal JDBC** ドライバー **UPDATE** ステートメントの例

レコードの **1** つの列を更新

次のステートメントは、ルートを更新します。

```
UPDATE HOSPITAL SET HOSPNAME = 'MISSION CREEK'
WHERE HOSPITAL.HOSPCODE = 'H001007'
```

階層パスの指定されたレコードの複数の列を更新

外部キーにより、更新するレコード (またはセグメント・インスタンス) を明確に識別することで、IMS Universal JDBC ドライバーは参照整合性を維持できます。次のステートメントは、特定の HOSPITAL の下の WARD レコードを更新します。この例では、WARD 表には外部キー HOSPITAL_HOSPCODE があります。レコードは、HOSPITAL 表に、値が 'H5140070000H' の HOSPCODE が存在する場合にのみ更新されます。

```
UPDATE WARD SET WARDNAME = 'EMGY',
   DOCCOUNT = '2', NURCOUNT = '4'
WHERE HOSPITAL_HOSPCODE = 'H5140070000H'
   AND WARDNO = '01'
```

無効な **IMS Universal JDBC** ドライバー **UPDATE** ステートメントの例

外部キー・フィールドの更新

外部キー・フィールドでの UPDATE の実行は、IMS Universal JDBC ドライバーには無効です。例えば、次の UPDATE 照会は失敗します。

```
UPDATE WARD SET WARDNAME = 'EMGY',
   HOSPITAL_HOSPCODE = 'H5140070000H'
WHERE WARDNO = '01'
```

DELETE ステートメントの使用法**:**

DELETE ステートメントは、表の行を削除するために使用します。 DELETE 操作は、すべての子セグメントにカスケードされます。

有効な **IMS Universal JDBC** ドライバー **DELETE** ステートメントの例

データベース全体を削除

次のステートメントは、HOSPITAL データベースを削除します。

```
DELETE FROM pcb01.HOSPITAL
```

ルートを削除

次のステートメントは、ルート・セグメント・インスタンス、および階層パス内のそのすべての子を削除します。

```
DELETE FROM pcb01.HOSPITAL
WHERE HOSPCODE = 'H5140070000H'
```

単一レコードを削除

外部キーにより、削除するレコード (またはセグメント・インスタンス) を明確に識別することで、IMS Universal JDBC ドライバーは参照整合性を維持できます。次のステートメントは、WARD 表から単一レコードを削除します。この例では、WARD 表には外部キー HOSPITAL_HOSPCODE があります。 WARD レコードは、HOSPITAL 表に値 'H5140070000H' の HOSPCODE が存在し、WARD 表に値 '0001' の WARDNO が存在する場合にのみ削除されます。

```
DELETE FROM pcb01.WARD
WHERE HOSPITAL_HOSPCODE = 'H5140070000H'
   AND WARDNO = '0001'
```

複数のレコードを削除

次のステートメントは、PATIENT 表から複数のレコードを削除します。この例では、PATIENT 表には HOSPITAL_HOSPCODE および WARD_WARDNO の 2 つの外部キーがあります。 PATIENT レコードは、HOSPITAL 表に値 'H5140070000H' の HOSPCODE が存在し、WARD 表に値 '0001' の WARDNO が存在し、および PATIENT 表に '0007' より大きい PATNUM の値が存在する場合にのみ削除されます。

```
DELETE FROM pcb01.PATIENT
WHERE PATNUM > '0007'
   AND HOSPITAL_HOSPCODE = 'H5140070000H'
   AND WARD_WARDNO = '0001'
```

次のステートメントは、データベース全体のすべての WARD セグメント・インスタンスを削除します。

```
DELETE FROM pcb01.WARD
```

WHERE 節の使用法**:**

SQL SELECT、UPDATE、および DELETE ステートメントで、WHERE 節はデータを条件に従って選択するために使用できます。

IMS Universal JDBC ドライバーで WHERE 節を使用する場合は、FROM 節でリストされているいずれかの表にある列を使用します。

推奨事項**:** 列を表名で修飾してください。列を表名で修飾しないと、FROM 節で結合された複数の表にその同じ列名が存在すると、あいまいさの問題が発生する可能性があります。

IMS JDBC ドライバーは、データベースを照会するときに、SQL 照会の WHERE 節を、セグメント検索引数 (SSA) リストに変換します。 SSA 規則では、WHERE 節に指定できる条件のタイプが制限されています。以下の制限が適用されます。

- 一般に、列を他の列とではなく、値と比較します。外部キーの導入により、一方の列が外部キーで他方の列が参照先の主キーである場合の、ある列と他の列との比較は有効です。以下に例を示します。

  ```
  WHERE HOSPITAL_HOSPCODE = HOSPITAL.HOSPCODE
  ```

  個々の修飾ステートメントで、列名と値との間には以下の演算子を使用できます。

  = 等しい

!= 等しくない

> より大きい

>= より大か等しい

< より小さい

<= 以下

例えば、次の WHERE 節は 2 つの列を比較しようとしているため失敗します。

```
WHERE PAYMENTS.PATNUM=PAYMENTS.AMOUNT
```

次の例は、WHERE 節が列と値とを比較しているため有効です。

```
WHERE PAYMENTS.PATNUM='A415'
```

- 括弧は使用しないでください。修飾ステートメントは左から右に評価されます。演算子の評価の順序は、セグメント検索引数の IMS 評価の順序となります。
- 表のすべての修飾ステートメントを隣接させてリストします。例えば、次の有効な WHERE 節では、同じ PATIENT 表からの修飾された列が隣接してリストされています。

```
WHERE PATIENT.PATNAME='BOB' OR PATIENT.PATNUM='A342' AND WARD.WARDNO='52'
```

次の無効な WHERE 節は、HOSPITAL 表からの列が WARD 表からの列と分離しているので失敗します。

```
WHERE HOSPITAL.HOSPNAME='Santa Teresa' AND WARD.WARDNO='52'
OR WARD.WARDNAME='CARD' AND HOSPITAL.HOSPCODE='90'
```

- OR 演算子は、同じ表からの列が含まれる修飾ステートメントの間でのみ使用できます。異なる表の間で OR 演算子を使用することはできません。異なる表の修飾ステートメントを結合するには、AND 演算子を使用します。例えば、次の無効な WHERE 節は失敗します。

```
WHERE WARD.WARDNO='03' OR PATIENT.PATNUM='A415'
```

ただし、次の WHERE 節は、OR 演算子が同じ表の 2 つの修飾ステートメントの間で使用されているため有効です。

```
WHERE PATIENT.PATNUM='A409' OR PATIENT.PATNAME='Sandy'
```

- Prepared ステートメントを使用する場合は、疑問符 (?) 文字を使用できます。これには、後で値が入ります。例えば、以下の WHERE 節は有効です。

```
WHERE PAYMENTS.AMOUNT>?
```

- 仮想外部キーである列を修飾ステートメント内で使用する場合、データベースへのアクセスがランダムのときは、以下の規則に従う必要があります。
  - ルート・セグメントの最初の子セグメント (レベル 2) のいずれかからの仮想外部キーのみを指定してください。階層内でそれより下位レベルにある仮想外部キーを指定することはできません。
  - その仮想外部キーを使用する複数の修飾ステートメントを結合するには、OR 演算子を使用します。
  - その仮想外部キーのみを使用する修飾ステートメントを指定します。例えば、その表または別の表から別の列を指定するために、OR 演算子や AND 演算子を使用することはできません。

例えば、以下の WHERE 文節は有効です。

```
WHERE HOSPITAL_HOSPCODE='ARS100100D' OR HOSPITAL_HOSPCODE='ARS100100D'
```

WHERE 節サブフィールドのサポート*:*

IMS JDBC ドライバーを使用して SQL ステートメントを渡す場合、フィールドが検索可能でサブフィールドによって完全に定義されている限り、WHERE 節を使用してどのフィールドのサブフィールドでもリストできます。

例えば、DBD 定義フィールドが ADDRESS という名前であり、長さは 30 バイトであるとします。 COBOL コピーブックでは、このフィールドは以下のコードに示すように、CITY、STATE、および ZIPCODE サブフィールドに細分化されています。

```
01 ADDRESS
   02 CITY PIC X(10)
   02 STATE PIC X(10)
   03 ZIP PIC X(10)
```

サブフィールドのサポートがない場合、WHERE 節の ADDRESS 値は手動で埋め込み、次のように入力する必要があります。

```
WHERE ADDRESS = 'san jose  ca        95141     '
```

サブフィールドのサポートがあれば、次のように WHERE 節を入力できます。

```
WHERE CITY = 'san jose'
   AND STATE = 'ca'
   AND ZIPCODE = '95141'
```

IMS JDBC ドライバーは、IMS に SQL 照会を送信する前に、個々のサブフィールドを変換してそれらを ADDRESS フィールドにバンドルします。

WHERE 節サブフィールド・サポートには、次の使用規則および制限が適用されます。

- サブフィールドにはパラメーター・マーカーがサポートされます。例えば、Prepared ステートメントの場合、次の WHERE 節の入力は有効です。

  ```
  WHERE CITY = ? AND STATE = ? AND ZIPCODE = ?
  ```

- サブフィールドでサポートされている唯一の関係演算子は、「=」(等号演算子)です。
- サブフィールドを接続するための唯一のブール演算子は、「AND」です。次の WHERE 節の入力は、サブフィールドが「AND」演算子のみを使用して接続されているので有効です。

  ```
  WHERE HOSPCODE=? OR CITY = ? AND STATE = ? AND ZIPCODE = ?
  ```

- 特定の検索可能フィールドのすべてのサブフィールドは、WHERE 節で指定する必要があります。フィールドのどのサブフィールドも省略できません。例えば、次の WHERE 節の入力は、STATE サブフィールドが指定されていないため無効です。

  ```
  WHERE CITY = ? AND ZIPCODE = ?
  ```

- WHERE 節でサブフィールドを指定する場合、検索可能フィールドのすべてのサブフィールドは、互いに隣接させてリストする必要があります。例えば、次の WHERE 節の入力は、サブフィールドが隣接してリストされていないため無効です。

  ```
  WHERE CITY = ? AND STATE = ? OR HOSPCODE=? AND ZIPCODE = ?
  ```

- WHERE 節には、複数の検索可能フィールドのサブフィールドを入力できます。例えば、PATNAME フィールドが LASTNAME サブフィールドと FIRSTNAME サブフィールドに細分化されている場合、次のように ADDRESS と PATNAME に対してサブフィールドを指定できます。

```
WHERE CITY = ? AND STATE = ? AND ZIPCODE = ?
   OR LASTNAME = ? AND FIRSTNAME = ?
```

- WHERE 節で複数の表を対象としてサブフィールドを指定する場合、個々の表の検索可能フィールドのすべてのサブフィールドをまとめてリストし、それから次の表のサブフィールドをリストする必要があります。例えば、HOSPITAL 表に ADDRESS フィールド、および PATIENT 表に PATNAME フィールドがある場合、次の WHERE 節の入力は、HOSPITAL についてすべての ADDRESS サブフィールドがリストされていないため無効です。

```
WHERE HOSPITAL.CITY = ? AND HOSPITAL.ZIPCODE = ?
   AND PATIENT.LASTNAME = ? AND PATIENT.FIRSTNAME = ?
```

IMS Universal JDBC ドライバーを使用して IMS データベースにアクセスする DL/I 呼び出しの作成

SQL 照会のサポートに加えて、IMS Universal JDBC ドライバーは、DL/I オブジェクトへのキャストもサポートしています。

通常、IMS Universal JDBC ドライバーは、アプリケーション・プログラムが IMS データを取得する SQL 照会を作成するためのインターフェースを提供します。しかし、JDBC インターフェースから DL/I オブジェクトをキャストすることもできます。新規のアプリケーション・コンテキストで既知の優れた DL/I 呼び出しを再利用したい場合、あるいは単純な照会で最大のパフォーマンスを得たい場合は、このアプローチを使用することをお勧めします。

1. JDBC および PSB 接続オブジェクトを作成する。
2. IMS データへの接続を取得する。
3. IMS Universal DL/I ドライバーから PSB ハンドルを取得するための JDBC 接続をキャストする。
4. 既存の PSB 接続オブジェクトから PCB オブジェクトを取得する。
5. PSB オブジェクトからのデータ接続を割り振る。
6. セグメント検索引数 (SSA) リストを作成してサブミットする。
7. 返されるデータを入れるパス・オブジェクトおよびパス・セット・オブジェクトを作成する。
8. パス・セット・データを処理する。

この例は、DL/I ドライバーにキャストしてデータを入手する方法を示しています。この例では、IMS Universal Database リソース・アダプターを使用して JEE サーバーで作成された JNDI データ・ソースから JDBC 接続にアクセスしています。

```
import java.sql.Connection;

import javax.naming.InitialContext;
import javax.sql.DataSource;

import com.ibm.ims.dli.PCB;
import com.ibm.ims.dli.PSB;
import com.ibm.ims.dli.Path;
```

```
import com.ibm.ims.dli.PathSet;
import com.ibm.ims.dli.SSAList;



public class JDBCToDLI {
 public static void main(String args[]){

  Connection conn = null; // This is a JDBC Connection
  // This is the equivalent connection object to the JDBC connection for the IMS Java DL/I API
  PSB psb = null;
  try{
   // Lookup the JNDI DataSource that contains the IMS connection information.
   // The JNDI DataSource would be defined in the JEE
   // server with the IMS Universal Database Resource Adapters
   InitialContext ic = new InitialContext();
   DataSource ds = (DataSource) ic.lookup("myJNDIName");

   // Get a JDBC Connection from the DataSource
   conn = ds.getConnection();

   // Cast the JDBC Connection to the IMS ConnectImpl in order to retrieve
   // a handle to the PSB in the IMS Java DL/I API
   psb = ((com.ibm.ims.jdbc.ConnectionImpl) conn).getPSB();


   //Get a PCB using the PSB you just created
   PCB ivp1pcb = psb.getPCB("PHONEAP");
   //Allocate PSB1 to establish a connection to the data
   psb.allocate();
   System.out.println("PSB for IVPDB1 Allocated");

   //Do work on PSB1
   SSAList ssaList = ivp1pcb.getSSAList("PhoneBook"); //Create an SSA list to use in a DLI call.
          //This SSA list qualifies the entire
          //PhoneBook segment. PhoneBook is an
          //alias name for segment A1111111 which
          //is specificed in the database view
          //(DFSIVPDBView.java).

   //Create a path object for later use
   Path path = null;
   PathSet ps = ivp1pcb.batchRetrieve(ssaList);  //This statement uses the PCB object created
          //above to do a batch retrieval of
          //all the segment instances of PhoneBook.
          //The data returned will be placed in
          //a PathSet which is a collection of
          //Path's containing the data you requested
   System.out.println("Batch retrieved all segment instances of Phone Book");

   System.out.println("FIRSTNAME¥tLASTNAME¥tEXTENSION¥tZIPCODE");
   System.out.println("---------------------------------------------------------------------");

   /*
   * The following while loop process the PathSet by checking that there is
   * a next element (ps.hasNext), then it prints out the three fields that are defined in the
   * database view (FIRSTNAME, LASTNAME, EXTENSION) for segment PhoneBook. This will continue
   * until  there are no more elements in the PathSet
   */
   while(ps.hasNext()){
    path = ps.next();
    System.out.println(path.getString("FIRSTNAME").trim()+"¥t¥t"+
    path.getString("LASTNAME").trim()+"¥t¥t"+
    path.getString("EXTENSION").trim()+"¥t"+
    path.getString("ZIPCODE").trim());
   }

      //INSERT a segment into the PhoneBook
      path = ssaList.getPathForInsert("PhoneBook");
      path.setString("LASTNAME", "LAST15");
      path.setString("FIRSTNAME", "FIRST15");
      path.setString("EXTENSION", "8-111-1515");
      path.setString("ZIPCODE", "D15/R15");
      ivp1pcb.insert(path, ssaList);
      System.out.println("¥nInserted New Phone Book Entry with LASTNAME equal to LAST15");

      //Batch retrieve all segment instances of the PhoneBook (A1111111) segment
      ps = ivp1pcb.batchRetrieve(ssaList);
      System.out.println("¥nBatch Retrieved all segment instances of Phone Book and verify LAST15 was inserted");
```

```
    System.out.println("FIRSTNAME¥tLASTNAME¥tEXTENSION¥tZIPCODE");
    System.out.println("-------------------------------------------------------------------");
    while(ps.hasNext()){
     path = ps.next();
     System.out.println(path.getString("FIRSTNAME").trim()+"¥t¥t"+
     path.getString("LASTNAME").trim()+"¥t¥t"+
     path.getString("EXTENSION").trim()+"¥t"+
     path.getString("ZIPCODE").trim());
    }

    //UPDATE FIRSTNAME to NEWNAME where LASTNAME equals LAST15
    ssaList.addInitialQualification(1, "LASTNAME", SSAList.EQUALS, "LAST15");
    if(ivp1pcb.getUnique(path, ssaList, true)){
     path.setString("FIRSTNAME", "NEWNAME");
     if(16448==ivp1pcb.replace(path)){
      System.out.println("¥nUpdated FIRSTNAME for segments with LASTNAME of LAST15");
     }
    }

    ssaList.removeAllQualificationStatements(1);
    //Batch retrieve all segment instances of the PhoneBook (A1111111) segment
    ps = ivp1pcb.batchRetrieve(ssaList);
    System.out.println("¥nBatch Retrieved all segment instances of Phone Book and " +
      "¥nverify that the FISTNAME was updated for the entry LAST15");

    System.out.println("FIRSTNAME¥tLASTNAME¥tEXTENSION¥tZIPCODE");
    System.out.println("-------------------------------------------------------------------");
    while(ps.hasNext()){
     path = ps.next();
     System.out.println(path.getString("FIRSTNAME").trim()+"¥t¥t"+
     path.getString("LASTNAME").trim()+"¥t¥t"+
     path.getString("EXTENSION").trim()+"¥t"+
     path.getString("ZIPCODE").trim());
    }

    //DELETE all segments where LASTNAME equals LAST15
    ssaList.addInitialQualification(1, "LASTNAME", SSAList.EQUALS, "LAST15");
    if(ivp1pcb.batchDelete(ssaList)==1){
     System.out.println("¥nSegment with LASTNAME equal to LAST15 has been deleted");
    }

    ssaList.removeAllQualificationStatements(1);

    //Batch retrieve all segment instances of the PhoneBook (A1111111) segment
    ps = ivp1pcb.batchRetrieve(ssaList);
    System.out.println("¥nBatch Retrieved all segment instances of Phone Book and " +
      "¥nverify that the segment with LASTNAME of LAST15 has been deleted");

    System.out.println("FIRSTNAME¥tLASTNAME¥tEXTENSION¥tZIPCODE");
    System.out.println("-------------------------------------------------------------------");
    while(ps.hasNext()){
     path = ps.next();
     System.out.println(path.getString("FIRSTNAME").trim()+"¥t¥t"+
     path.getString("LASTNAME").trim()+"¥t¥t"+
     path.getString("EXTENSION").trim()+"¥t"+
     path.getString("ZIPCODE").trim());
    }

    //Commit the work
    psb.commit();
    System.out.println("¥nPSB Committed");
    //Deallocate the PSB
    psb.deallocate();
    System.out.println("PSB deallocated");
    //Close the socket connection
    psb.close();
    System.out.println("Connection Closed");
    System.out.println("Open Database IVP Completed");
} catch(Exception e) {
 e.printStackTrace();
 try {
     psb.deallocate();
     psb.close();
 }catch(Exception e1){
  e1.printStackTrace();
```

```
    }
  }
 }
}
```

関連概念:

792 ページの『IMS Universal DL/I ドライバーを使用したプログラミング』

XML 用の IMS Universal JDBC ドライバーのサポート

IMS Universal JDBC ドライバーを使用して XML データを IMS データベースに保管したり、XML データを IMS データベースからリトリーブしたりするためのアプリケーションを作成できます。これは、タイプ 4 とタイプ 2 の両方のドライバーでサポートされます。

XML 用の IMS Universal JDBC ドライバーのサポートを使用して、以下の操作を実行できます。

- SQL SELECT ステートメントによって、IMS データベースから XML データを文字ラージ・オブジェクト (CLOB) としてリトリーブする。
- PreparedStatement.setClob メソッドまたは PreparedStatement.setCharacterStream メソッドを使用して、SQL INSERT ステートメントによって XML データを IMS データベースに保管する。

IMS Universal JDBC ドライバーを使用して XML データを保管またはリトリーブする構文は、XML データが物理的にどのように IMS データベースに保管されるかには関係ありません。インターフェースにとっては、データの保管モードが分解保管モードか原形保管モードのいずれか一方なのか、または両方のモードが使用されているのか、あるいはデータを保管する IMS データベースが既存のものか新規のものかは重要ではありません。

XML 用の IMS Universal JDBC ドライバーのサポートを使用するには、プログラム仕様ブロック (PSB) に対応するランタイム Java メタデータ・クラスを生成する必要があります。この Java メタデータ・クラスで、XML データを保管およびリトリーブするための IMS データベースで列フィールドを定義し、データの構造を記述する XML スキーマを識別する必要があります。

IMS バージョン 12 以降には、新規の DBD ソース・パラメーター (FIELD ステートメントの DATATYPE=XML パラメーターと DFSMARSH ステートメントの OVERFLOW パラメーター) が含まれています。これらを使用して、XML に含まれるフィールドおよびオーバーフロー・セグメントを定義することができます。IMS システムで IMS カタログ・データベースを使用している場合は、IMS Enterprise Suite Explorer for Development は、静的メタデータ・クラスを生成する代わりに、カタログに接続して必要なメタデータを動的にリトリーブすることができます。IMS システムで IMS カタログ・データベースを使用していない場合は、これらのフィールド定義は、IMS Explorer for Development で作成される静的 Java メタデータ・クラスに組み込まれます。

Java メタデータ・クラスの XML データ・タイプ列フィールドの定義

XML 用の IMS Universal JDBC ドライバー・サポートを使用するには、XML データの保管とリトリーブに使用する XML データ・タイプ列フィールドを定義する必要があります。

注: 15 以降では、XML データ・タイプ定義用の新しい DBD 生成パラメーターをサポートしています。FIELD ステートメントの DATATYPE=XML パラメーターと、DFSMARSH ステートメントの OVERFLOW セグメント定義です。これらのパラメーターを使用する場合、Java メタデータ・クラスには XML 定義がすでに含まれているので、クラスを変更する必要はありません。IMS カタログ・データベースを使用する場合は、メタデータは、静的メタデータ・クラスでなく、データ接続で使用可能になります。

Java メタデータ・クラスの XML データ・タイプ列フィールドを定義する手順は、以下のとおりです。

1. IMS Enterprise Suite Explorer for Development を使用して、Java メタデータ・クラスを生成します。
2. IMS Enterprise Suite DLIModel ユーティリティー・プラグインを使用してデータベース用の XML スキーマを生成するか、DBD とデータ・タイプのマッピングに基づいて手動で XML スキーマを作成します。
3. 生成した Java メタデータ・クラスを変更して、XML データ・タイプ列フィールドを指定します。 分解保管モードで XML データの保管またはリトリーブを行う場合は、以下の DLITypeInfo コンストラクター構文を使用して XML データ・タイプ列フィールドを定義してください。同じセグメント内で 1 つ以上の XML データ・タイプ列フィールドを定義できます。

   ```
   public DLITypeInfo(String fieldName,
               String XMLSchemaName,
                DLITypeInfo.XML);
   ```

4. データベース接続のセットアップ中に、Java メタデータ・クラスの名前を IMS Universal JDBC ドライバーに渡します。

以下の例は、分解モードの場合に Java メタデータ・クラスの XML 列のデータ・タイプ・フィールドを定義する方法を示しています。この例では、「BMP255-PCB01.xsd」XML スキーマに関連付けられた「HOSPXML」という XML データ・タイプ列フィールドを定義しています。「B.xsd」XMLスキーマに関連付けられた、「HXML」という別の XML データ・タイプ列フィールドも定義しています。

```
// The following describes Segment: HOSPITAL ("HOSPITAL") in PCB: PCB01 ("PCB01")
    static DLITypeInfo[] PCB01HOSPITALArray= {
        new DLITypeInfo("HOSPLL", DLITypeInfo.CHAR, 1, 2, "HOSPLL"),
        new DLITypeInfo("HOSPCODE", DLITypeInfo.CHAR, 3, 12,
           "HOSPCODE", DLITypeInfo.UNIQUE_KEY),
        new DLITypeInfo("HOSPNAME", DLITypeInfo.CHAR, 15, 17, "HOSPNAME"),
        new DLITypeInfo("HOSPXML", "BMP255-PCB01.xsd", DLITypeInfo.XML),
        new DLITypeInfo("HXML", "B.xsd", DLITypeInfo.XML)
    };
```

IMS Universal JDBC ドライバーを使用した XML データの保管

IMS Universal JDBC ドライバーを使用すると、SQL INSERT ステートメントにより XML データを IMS データベースに保管することができます。

IMS Universal JDBC ドライバー・アプリケーションに XML データを保管する手順は、以下のとおりです。

1. *http://www.ibm.com/ims/schema-resolver/file/path* 環境変数を設定して、入力 XML データ構造を記述した XML スキーマ・ファイル (.xsd) が入っているファイル・パスを指定します。 以下の例は、プログラムで環境変数を設定する方法を示しています。この例では、ファイル・パス `uxml/samples` は、XML スキーマ・ファイルへの相対パスを示します。絶対ファイル・パスを指定することもできます。

   ```
   System.setProperty("http://www.ibm.com/ims/schema-resolver/file/path",
      "uxml/samples");
   ```

2. XML データ・ソースを指定します。 外部ソース (ファイルなど) から XML データを読み取る場合は、java.io.Reader オブジェクトを作成して入力 XML データをラップする必要があります。以下の例は、`hospwashington.xml` という外部ファイルをラップするための InputStreamReader オブジェクトの作成方法を示しています。 InputStreamReader オブジェクトは、入力ファイルから読み取ったバイトを ASCII エンコードから Unicode に変換します。

   ```
   String doc = "hospwashington.xml";
   InputStream fileStream = getClass().getResourceAsStream(doc);
   if (fileStream == null) {
       throw new FileNotFoundException("Insert Document: '" + doc + "' was
       not found in classpath");
   }
   InputStreamReader fileReader = new InputStreamReader(fileStream, "ASCII");
   ```

3. XML データを挿入します。

   a. SQL INSERT 呼び出しを表す java.sql.preparedStatement オブジェクトを作成します。 この SQL INSERT ステートメントに、XML データを保管する XML 列の名前を指定する必要があります。列名は、Java メタデータ・クラスに定義されている名前と一致していなければなりません。

      以下の例は、java.sql.Connection インスタンス conn を使用して HOSPITAL セグメントの hospxml 列にデータを挿入するための preparedStatement オブジェクトの作成方法を示しています。

      ```
      String s = "INSERT INTO pcb01.HOSPITAL (hospxml) VALUES (?)"
      PreparedStatement ps = conn.prepareStatement(s);
      ```

   b. preparedStatement オブジェクトに挿入する XML データの値を設定します。

      以下の表では、XML 列へのデータの挿入に使用できるメソッドと、対応する入力データ・タイプを示します。

表 *103. XML* 列の更新用のメソッドとデータ・タイプ

メソッド	入力データ・タイプ
PreparedStatement.setCharacterStream	Reader
PreparedStatement.setClob	Clob

分解保管モードでは、XML データは EBCDIC エンコードで保管されま

す。原形保管モードでは、デフォルトのエンコード方式は Unicode です。

以下のサンプル・コードは、病院データベースに XML データを挿入する方法を示しています。

```
package uxml.samples;

import java.io.*;
import java.sql.Clob;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.PreparedStatement;
import com.ibm.ims.jdbc.IMSDataSource;

public class StoreXMLSamples{

    public static void main(String argv[]) throws SQLException,IOException {
        IMSDataSource ds = new IMSDataSource();
        ds.setDatabaseName("class://uxml.samples.BMP255NewSyntaxDatabaseView");
        ds.setDatastoreName("IMS1");
        ds.setDatastoreServer("yourhost.yourdomain.com");
        ds.setPortNumber(5555);
        ds.setDriverType(IMSDataSource.DRIVER_TYPE_4);
        ds.setUser("myUserID");
        ds.setPassword("myPass");

        // Specify file path of XML schema
        System.setProperty("http://www.ibm.com/ims/schema-resolver/file/path",
           "uxml/samples");

        Connection conn = null;

        try {
          conn = ds.getConnection();
          Statement st = conn.createStatement();
         String doc = "hospwashington.xml";
         StoreXMLSamples storeSample = new StoreXMLSamples();
         InputStream fileStream =
             storeSample.getClass().getResourceAsStream(doc);
         if (fileStream == null) {
             throw new FileNotFoundException("Insert Document: '" +
                 doc + "' was not found in classpath");
         }

          // Convert XML document from ASCII to Unicode
          InputStreamReader fileReader =
             new InputStreamReader(fileStream, "ASCII");

         PreparedStatement ps =
             conn.prepareStatement("INSERT INTO pcb01.HOSPITAL" +
               " (hospxml) VALUES (?)");

            ps.setCharacterStream(1, fileReader, -1);
            int rows = ps.executeUpdate();
            System.out.println("Inserted");
            conn.commit();
           conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
            if (!conn.isClosed()) {
                conn.rollback();
                conn.close();
```

```
            }
        }
    }
}
```

IMS Universal JDBC ドライバーを使用した XML データのリトリーブ

IMS Universal JDBC ドライバーを使用すると、SQL SELECT ステートメントで、IMS データベースから XML データを文字ラージ・オブジェクト (CLOB) としてリトリーブすることができます。

IMS Universal JDBC ドライバー・アプリケーションの XML データをリトリーブする手順は、以下のとおりです。

1. *http://www.ibm.com/ims/schema-resolver/file/path* 環境変数を設定して、入力 XML データ構造を記述した XML スキーマ・ファイル (.xsd) が入っているファイル・パスを指定します。 以下の例は、プログラムで環境変数を設定する方法を示しています。この例では、ファイル・パス `uxml/samples` は、XML スキーマ・ファイルへの相対パスを示します。絶対ファイル・パスを指定することもできます。

   ```
   System.setProperty("http://www.ibm.com/ims/schema-resolver/file/path",
      "uxml/samples");
   ```

2. XML データをリトリーブするための SQL SELECT ステートメントを指定して実行します。 SQL SELECT ステートメント内のデータベース表に、XML データをリトリーブするための XML 列が含まれている必要があります。列名を SQL SELECT ステートメントに明示的に指定する場合、その列名は Java メタデータ・クラスに定義されている名前と一致していなければなりません。

   以下の例は、HOSPITAL セグメントの hospxml 列をリトリーブする SQL SELECT 呼び出しで java.sql.resultSet オブジェクトを取得する方法を示しています。この例では、st は java.sql.Statement インスタンスです。

   ```
   ResultSet rs = st.executeQuery("SELECT hospxml FROM PCB01.HOSPITAL");
   ```

3. リトリーブ呼び出しの実行後に、resultSet オブジェクトから XML データを読み取ります。 XML データは、resultSet 内の java.sql.Clob オブジェクトに保管されています。

以下のサンプル・コードは、病院データベースから XML データをリトリーブする方法を示しています。

```
package uxml.samples;

import java.io.*;
import java.sql.Clob;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import com.ibm.ims.jdbc.IMSDataSource;

public class RetrieveXMLSamples{

    public static void main(String argv[]) throws SQLException,IOException {
        IMSDataSource ds = new IMSDataSource();
        ds.setDatabaseName("class://uxml.samples.BMP255NewSyntaxDatabaseView");
        ds.setDatastoreName("IMS1");
```

```
        ds.setDatastoreServer("yourhost.yourdomain.com");
        ds.setPortNumber(5555);
        ds.setDriverType(IMSDataSource.DRIVER_TYPE_4);
        ds.setUser("myUserId");
        ds.setPassword("myPass");

        // Specify file path of XML schema
        System.setProperty("http://www.ibm.com/ims/schema-resolver/file/path",
           "uxml/samples");

        Connection conn = null;

        try {
            conn = ds.getConnection();

            Statement st = conn.createStatement();

        ResultSet rs = st.executeQuery("SELECT hospxml FROM PCB01.HOSPITAL");

        StringWriter sw = new StringWriter();

            while (rs.next()) {

                Clob clob = rs.getClob(1);
                Reader reader = clob.getCharacterStream();
                char[] buffer = new char[1000];
                int read = reader.read(buffer);
                while (read != -1) {
                 sw.write(buffer,0,read);
                    read = reader.read(buffer);
                }
            }

           String result = sw.toString();
           System.out.println(result);
           System.out.println();

           conn.commit();
           conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
            if (!conn.isClosed()) {
                conn.rollback();
                conn.close();
            }
        }
    }
}
```

JDBC のデータ変換サポート

IMS JDBC ドライバーには、クライアント・アプリケーション用のデータ変換機能があります。ライブラリーは、IMS カタログ・データベースまたは Java データベース・メタデータ・クラスから情報を受け取ると、データを別のデータ型に内部的に変換できます。IMS Universal DL/I ドライバーには、カスタム・データ・タイプを変換するための拡張可能なユーザー・データ・タイプ・コンバーターもあります。

サポートされている JDBC データ型

次の表では、各 JDBC データ型に対応する Java データ型をリストしています。

表 *104.* サポートされている *JDBC* データ型

JDBC データ・タイプ	**Java** データ型	長さ
ARRAY	java.lang.Array	アプリケーションで定義
BIGINT	long	8 バイト
BINARY	byte[]	1 - 32 KB
BIT	ブール	1 バイト
CHAR	java.lang.String	1 - 32 KB
CLOB	java.sql.Clob	アプリケーションで定義
日付	java.sql.Date	アプリケーションで定義
DOUBLE	double	8 バイト
FLOAT	float	4 バイト
INTEGER	int	4 バイト
PACKEDDECIMAL	java.math.BigDecimal	1 から 10 バイト
SMALLINT	short	2 バイト
STRUCT	java.lang.Struct	アプリケーションで定義
TIME	java.sql.Time	アプリケーションで定義
TIMESTAMP	java.sql.Timestamp	アプリケーションで定義
TINYINT	byte	1 バイト
ZONEDDECIMAL	java.math.BigDecimal	1 から 19 バイト

データ型のリトリーブおよび変換のメソッド

IMS Universal JDBC ドライバーでは、ResultSet インターフェース (java.sql.ResultSet) を使用してデータをリトリーブし、データベース・メタデータで定義されているデータ型から Java アプリケーションで必要なデータ型に変換することができます。 同様に、IMS Universal DL/I ドライバーでは、Path インターフェースを使用して、データをリトリーブし、Java データ型に変換することができます。

以下の表では、特定の Java データ型のデータ・アクセスに使用できる get メソッドを示しています。IMS Universal JDBC ドライバーの場合は ResultSet インターフェースであり、IMS Universal DL/I ドライバーの場合は Path インターフェースです。

「切り捨てまたはデータ欠落なし」列は、特定の getXXX メソッドでアクセスするように設計されているデータ型を示しています。この列のデータ型をそれに対応するメソッドで使用する場合は、切り捨てまたはデータ欠落は生じません。「有効、データ保全性なし」列のデータ型は、他のすべての有効な呼び出しを示しています。ただし、特定の getXXX メソッドを使用してアクセスした場合のデータ保全性は保証できません。データ型がどちらの列にも属していない場合、そのデータ型に特定の getXXX メソッドを使用すると、例外が出される結果になります。

表 105. データ型をリトリーブする *ResultSet.getXXX* および *Path.getXXX* メソッド

ResultSet.getXXX メソッドまたは **Path.getXXX** メソッド	データ型 **(**リストされていないものは例外が出される結果になる**)**	
	切り捨てまたはデータの損失無し	データ保全性が保証外の正当なメソッド
getByte	TINYINT UTINYINT	SMALLINT INTEGER BIGINT FLOAT DOUBLE BIT CHAR VARCHAR PACKEDDECIMAL[1] ZONEDDECIMAL[1]
getShort	SMALLINT USMALLINT	TINYINT INTEGER BIGINT FLOAT DOUBLE BIT CHAR VARCHAR PACKEDDECIMAL[1] ZONEDDECIMAL[1]
getInt	INTEGER UINTEGER	TINYINT SMALLINT BIGINT FLOAT DOUBLE BIT CHAR VARCHAR PACKEDDECIMAL[1] ZONEDDECIMAL[1]
getLong	BIGINT UBIGINT	TINYINT SMALLINT INTEGER FLOAT DOUBLE BIT CHAR VARCHAR PACKEDDECIMAL[1] ZONEDDECIMAL[1]

表 105. データ型をリトリーブする *ResultSet.getXXX* および *Path.getXXX* メソッド (続き)

ResultSet.getXXX メソッドまたは **Path.getXXX** メソッド	データ型 **(**リストされていないものは例外が出される結果になる**)**	
	切り捨てまたはデータの損失無し	データ保全性が保証外の正当なメソッド
getFloat	FLOAT	TINYINT SMALLINT INTEGER BIGINT DOUBLE BIT CHAR VARCHAR PACKEDDECIMAL[1] ZONEDDECIMAL[1]
getDouble	DOUBLE	TINYINT SMALLINT INTEGER BIGINT FLOAT BIT CHAR VARCHAR PACKEDDECIMAL[1] ZONEDDECIMAL[1]
getBoolean	BIT	TINYINT SMALLINT INTEGER BIGINT FLOAT DOUBLE CHAR VARCHAR PACKEDDECIMAL[1] ZONEDDECIMAL[1]
getString	CHAR VARCHAR	TINYINT SMALLINT INTEGER BIGINT FLOAT DOUBLE BIT PACKEDDECIMAL[1] ZONEDDECIMAL[1] BINARY DATE TIME TIMESTAMP

表 105. データ型をリトリーブする *ResultSet.getXXX* および *Path.getXXX* メソッド (続き)

ResultSet.getXXX メソッドまたは **Path.getXXX** メソッド	データ型 **(**リストされていないものは例外が出される結果になる**)**	
	切り捨てまたはデータの損失無し	データ保全性が保証外の正当なメソッド
getBigDecimal	BINARY[3] PACKEDDECIMAL[1] ZONEDDECIMAL[1]	TINYINT SMALLINT INTEGER BIGINT FLOAT DOUBLE BIT CHAR VARCHAR
getClob	CLOB[2]	その他はすべて例外の結果
getBytes	BINARY	その他はすべて例外の結果
getDate	日付	CHAR VARCHAR TYIMESTAMP
getTime	TIME	CHAR VARCHAR TIMESTAMP
getTimestamp	TIMESTAMP	CHAR VARCHAR DATE TIME

注**:**

1. PACKEDDECIMAL および ZONEDDECIMAL は、IMS Universal JDBC ドライバーおよび IMS Universal DL/I ドライバー用の拡張データ型です。その他のデータ型は、SQL92 で定義されている標準 SQL データ型です。制約事項**:** PACKEDDECIMAL および ZONEDDECIMAL データ型は、SIGN LEADING モードおよび SIGN SEPARATE モードはサポートしません。これらの 2 つのデータ・タイプについては、符号情報が常に Sign Trailing (符号末尾) 方式で保管されます。
2. CLOB データ型は、XML データのリトリーブおよび保管についてのみサポートされます。
3. BINARY データ型は、バイナリー形式コンバーターと一緒に使用される10 進データでのみ有効です。

フィールド・タイプが PACKEDDECIMAL または ZONEDDECIMAL のいずれかの場合、型修飾子は、フィールドのレイアウトを表す COBOL PICTURE ストリングです。 9、P、V、および S の有効な組み合わせを含むすべての COBOL PICTURE ストリングはサポートされます。 PICTURE ストリングの拡張は、自動的に処理されます。例えば、「9(5)」は、有効な PICTURE ストリングです。ゾーン 10 進数の場合、小数点も PICTURE ストリングで使用できます。`PIC 9(06)V99 COMP` および `PIC 9(06)V99 COMP-4` は、BINARY 10 進データに対して有効な PICTURE 文節です。

フィールドに DATE、TIME、または TIMESTAMP データが含まれている場合、データ形式は型修飾子で示されます。例えば、*ddMMyyyy* という型修飾子は、データが次のような形式であることを意味します。

```
11122015 is December 11, 2015
```

DATE および TIME 型の場合、java.text.SimpleDateFormat クラスのすべてのフォーマット・オプションがサポートされます。

TIMESTAMP 型の場合、フォーマット・オプション「f」が、ナノ秒用に使用できます。 TIMESTAMP には、最大で 9 つの「f」を入れることができ、ミリ秒になると「S」オプションで置き換えられます。その場合には、「fff」はミリ秒の精度を示します。 TIMESTAMP の形式例は、以下のとおりです。

```
yyyy-mm-dd hh:mm:ss.fffffffff
```

Java データ型にマップする COBOL コピーブックの型

IMS のデータは強く型付けされていないので、COBOL コピーブックの型を使用して、IMS データを Java データ型にマップすることができます。

次の表では、COBOL コピーブックの型と、DLIDatabaseView クラスの DLITypeInfo 定数、および Java データ型の両方とのマップを説明しています。

表 *106. COBOL* フォーマットから *DLITypeInfo* 定数および *Java* データ型へのマッピング

コピーブックのフォーマット	**DLITypeInfo** 定数	**Java** データ型
PIC X	CHAR	`java.lang.String`
PIC 9 BINARY[1]	『PICTURE 節に基づく DLITypeInfo 定数および Java データ型』を参照[2]	『PICTURE 節に基づく DLITypeInfo 定数および Java データ型』を参照[2]
COMP-1	FLOAT	float
COMP-2	DOUBLE	double
PIC 9 COMP-3[3]	PACKEDDECIMAL	`java.math.BigDecimal`
PIC 9 DISPLAY[4]	ZONEDDECIMAL	`java.math.BigDecimal`

注:

1. BINARY データ項目の同義語は、COMP および COMP-4 です。COMP または COMP-4 を指定した `PIC 9(06)V99` ステートメントは、BINARY 10 進データに使用されます。
2. BINARY データ項目の場合、DLITypeInfo 定数および Java データ型は、PICTURE 節の桁数によって異なります。表『PICTURE 節に基づく DLITypeInfo 定数および Java データ型』で、PICTURE 節の長さに基づく型について説明しています。
3. PACKED-DECIMAL は、COMP-3 の同義語です。
4. USAGE 節が、グループ・レベルおよびエレメント・レベルのいずれにおいても指定されていない場合、DISPLAY が想定されます。

次の表では、PICTURE 節に基づく DLITypeInfo 定数および Java データ型を示しています。

表 *107. PICTURE* 節に基づく *DLITypeInfo* 定数および *Java* データ型

PICTURE 節内での桁	使用ストレージ	**DLITypeInfo** 定数	**Java** データ型
1 から 2	1 バイト	TINYINT UTINYINT	byte
1 から 4	2 バイト	SMALLINT USMALLINT	short
5 から 9	4 バイト	INTEGER UINTEGER	int
10 から 18	8 バイト	BIGINT UBIGINT	long

次の表は、DLITypeInfo 定数にマップする特定のコピーブック・フォーマットの例を示しています。

表 *108. DLITypeInfo* 定数にマップするコピーブック・フォーマット

コピーブックのフォーマット	**DLITypeInfo** 定数
PIC X(25)	CHAR
PIC 9(02) COMP	UTINYINT
PIC S9(04) COMP	SMALLINT
PIC 9(04) COMP	USMALLINT
PIC S9(06) COMP-4	INTEGER
PIC 9(06) COMP-4	UINTEGER
PIC 9(06)V99 COMP または COMP-4	BINARY
PIC S9(12) BINARY	BIGINT
PIC 9(12) BINARY	UBIGINT
COMP-1	FLOAT
COMP-2	DOUBLE
PIC S9(06)V99	ZONEDDECIMAL
PIC 9(06).99	ZONEDDECIMAL
PIC S9(06)V99 COMP-3	PACKEDDECIMAL

IMS Universal DL/I ドライバーを使用したプログラミング

非管理対象環境において、Java クライアントから IMS データベースに直接アクセスするために詳細な照会を作成する必要がある場合は、IMS Universal DL/I ドライバーを使用します。

階層データベースとリレーショナル・データベースの根本的な違いのために、JDBC API はフルセットの IMS データベース機能へのアクセスを提供しない場合があります。 IMS Universal DL/I ドライバーは、IMS でアプリケーションを作成する他のプログラミング言語で使用されている従来の IMS DL/I データベース呼び出しインターフェースと密接に関連しており、JDBC API よりも IMS データベース機能に対する低レベルのアクセスを提供します。 IMS Universal DL/I ドライバーを使用すると、セグメント検索引数 (SSA) を作成し、プログラム連絡ブロック (PCB)

オブジェクトのメソッドを使用して、セグメントの読み取り、挿入、更新、削除、またはバッチ操作を行うことができます。セグメント階層での完全なナビゲーション制御を手に入れることができます。

IMS Universal ドライバーを使用した Java アプリケーションを作成する準備

IMS Universal ドライバーを使用する Java アプリケーション・プログラムでは、Java Development Kit (JDK) 7.0 が必要です。JMP 領域および JBP 領域で実行される Java プログラムは、JDK 7.0 以降が必要です。IMS Universal ドライバーを使用する Java アプリケーション・プログラムは、IMS データベースと対話するために、データベース・メタデータへのアクセス権限を持っている必要があります。IMS カタログ・データベースで直接このメタデータにアクセスするか、IMS Enterprise Suite Explorer for Development を使用して Java メタデータ・クラスとしてこのメタデータを生成することができます。

IMS Universal DL/I ドライバー・アプリケーションを作成するための基本ステップ

一般に、IMS Universal DL/I ドライバーを使用してアプリケーション・プログラムを作成するには、以下のタスクを実行する必要があります。

IMS Universal DL/I ドライバー・アプリケーションを作成するには、以下のステップを実行します。

1. IMS Universal DL/I ドライバー・クラス、インターフェース、およびメソッドを含む `com.ibm.ims.dli` パッケージをインポートします。
2. IMS データベース・サブシステムに接続します。
3. 1 つ以上の PCB が含まれているプログラム仕様ブロック (PSB) を取得します。
4. PCB ハンドルをリトリーブします。これは IMS データベースのアプリケーションのビューを定義し、データベース情報をリトリーブ、挿入、更新、および削除するためのデータベース呼び出しを発行できます。
5. データベース階層で 1 つ以上のセグメントの非修飾セグメント検索引数リスト (SSAList) を取得します。
6. DL/I 呼び出しのターゲットとなるセグメントを指定する修飾ステートメントを追加します。
7. データをリトリーブする場合は、返されるセグメント・フィールドにマークを付けます。
8. IMS データベースに対する DL/I 呼び出しを実行します。
9. DL/I プログラミング・インターフェースから返されたエラーを処理します。
10. IMS データベース・サブシステムから切断します。

関連タスク:

802 ページの『IMS Universal DL/I ドライバー・アプリケーションでのデータのリトリーブ』

関連資料:

710 ページの『実行時 Java メタデータ・クラスの生成』

IMS Universal DL/I ドライバー・サポート用の Java パッケージ

IMS Universal DL/I ドライバー・メソッドを呼び出す前に、これらのメソッドを含む各種の Java パッケージのすべてまたは一部にアクセスする必要があります。

これは、パッケージまたは特定のクラスをインポートするか、あるいは完全修飾クラス名を使用して実行できます。 IMS Universal DL/I ドライバー・アプリケーションでは、次のパッケージまたはクラスが必要となる場合があります。

com.ibm.ims.dli
IMS Universal DL/I ドライバー・用のコア・クラス、インターフェース、およびメソッドが含まれます。

com.ibm.ims.base
DL/I または IMS によって返されるエラーの例外クラスが含まれます。

関連資料:

Java API 文書 (Javadoc) (アプリケーション・プログラミング API)

IMS Universal DL/I ドライバーを使用した IMS データベースへの接続

IMS Universal DL/I ドライバー・アプリケーションから DL/I 呼び出しを実行する前に、IMS データベースに接続する必要があります。

IMS Universal DL/I ドライバー・アプリケーションは、`com.ibm.ims.dli` パッケージの一部である PSB インターフェースを使用して IMS データベースとの接続を確立できます。IMSConnectionSpec インスタンスを使用して接続プロパティーを受け渡します。

IMS Universal DL/I ドライバーを使用して IMS データベースに接続するには、以下のようにします。

1. IMSConnectionSpecFactory クラスで createIMSConnectionSpec メソッドを呼び出して、IMSConnectionSpec インスタンスを作成します。
2. IMSConnectionSpec インスタンスに対して以下の接続プロパティーを設定します。

   DatastoreName

   アクセスする IMS データ・ストアの名前。

   - タイプ 4 コネクティビティーを使用する場合、**DatastoreName** プロパティーは、ODBM に定義されているデータ・ストアの名前と一致したものであるかまたはブランクである必要があります。データ・ストア名は、DATASTORE(NAME=*name*) パラメーターまたは DATASTORE(NAME=*name*, ALIAS(NAME=*aliasname*)) パラメーターのいずれかを使用して、ODBM CSLDCxxx PROCLIB メンバー内で定義されます。別名が指定される場合、**datastoreName** プロパティーの値として *aliasname* を指定する必要があります。 **DatastoreName** の値が空白 (または指定されていない) 場合、ODBM に定義されているすべてのデータ・ストアでデータ共用が使用可能となっていると見なされるため、IMS Connect は使用可能な任意の ODBM のインスタンスに接続します。

- タイプ 2 コネクティビティーを使用する場合は、**DatastoreName** プロパティーを IMS サブシステムの別名に設定します。Java 従属領域のランタイムの場合は、これは設定する必要はありません。

DatabaseName

ターゲット IMS データベースを表すデータベース・メタデータの場所。

メタデータが IMS カタログに保管されているのか、IMS Enterprise Suite Explorer for Development によって生成された静的メタデータ・クラスとして保管されているのかに応じて、**DatabaseName** プロパティーを以下の 2 つの方法のいずれかで指定することができます。

- IMS システムで IMS カタログを使用している場合、**DatabaseName** プロパティーは、アプリケーションがターゲット IMS データベースへのアクセスに使用する PSB の名前です。
- IMS Explorer for Development を使用している場合、**databaseName** プロパティーは、IMS Explorer for Development によって生成された Java メタデータ・クラスの完全修飾名です。URL には、`class://` の接頭部を付ける必要があります (例えば、`class://com.foo.BMP255DatabaseView`)。

J2C 接続ファクトリー環境では、リソース・アダプターに指定されたデフォルト値に影響を与えずに、個別の接続の **DatabaseName** プロパティーをオーバーライドできます。

MetadataURL

ターゲット IMS データベースを表すデータベース・メタデータの場所。

このプロパティーは推奨されていません。代わりに **DatabaseName** を使用してください。

MetadataURL プロパティーは、IMS Enterprise Suite Explorer for Developmentによって生成された Java メタデータ・クラスの完全修飾名です。 URL には、`class://` の接頭部を付ける必要があります (例えば、`class://com.foo.BMP255DatabaseView`)。

J2C 接続ファクトリー環境では、リソース・アダプターに指定されたデフォルト値に影響を与えずに、個別の接続の **MetadataURL** プロパティーをオーバーライドできます。

PortNumber

IMS Connect と通信するために使用される TCP/IP サーバーのポート番号。ポート番号は、IMS Connect 構成 PROCLIB メンバーの ODACCESS ステートメントで DRDAPORT パラメーターを使用して定義されます。デフォルトのポート番号は 8888 です。タイプ 2 コネクティビティーの使用時にはこのプロパティーを設定しないでください。

DatastoreServer

データ・ストア・サーバー (IMS Connect) の名前または IP アドレス。ホスト名 (例えば、dev123.svl.ibm.com) または IP アドレス (例え

ば、192.166.0.2) のいずれかを指定できます。タイプ 2 コネクティビティーの使用時にはこのプロパティーを設定しないでください。

DriverType
使用するドライバー・コネクティビティーのタイプ (値はタイプ 4 コネクティビティーに対しては `IMSConnectionSpec.DRIVER_TYPE_4`、タイプ 2 コネクティビティーに対しては `IMSConnectionSpec.DRIVER_TYPE_2` である必要があります)。

allMetadata

オプション。このプロパティーが `true` に設定された場合、DatabaseMetadata インターフェースは、IMS カタログにあるすべてのリソースに関する情報を返します。このプロパティーが `false` に設定された場合、DatabaseMetadata インターフェースは、割り振り済み PSB に関する情報を返します。このプロパティーのデフォルト値は `false` です。

sslConnection
オプション。この接続がデータ暗号化用に Secure Sockets Layer (SSL) を使用するかどうかを示します。 SSL を使用可能にするにはこのプロパティーを「true」に設定します。使用しない場合は「false」を設定します。タイプ 2 コネクティビティーの使用時にはこのプロパティーを設定しないでください。

sslKeyStoreType
オプション。セキュア・ソケット接続の確立に必要な暗号オブジェクトを含むファイルの形式を指定します。有効な値は、「JKS」および「PKCS12」です。この値は、**sslConnection** が「true」に設定されていて、**sslKeyStoreType** が指定されていない場合にのみ使用します。 **sslKeyStoreType** パラメーターのデフォルトは「JKS」です。

sslSecureSocketProtocol
オプション。新しい接続の暗号通信プロトコルを指定します。サーバーがサポートしているプロトコルで、最高レベルのセキュリティーを提供するプロトコルを指定します。有効な値は、「SSL」、「SSLv3」、「TLSv1.1」、および「TLSv1.2」です。この値は、**sslConnection** が「true」に設定されている場合のみ使用します。**sslConnection** が「true」に設定されていて、**sslSecureSocketProtocol** が指定されていない場合、デフォルト・プロトコルは JRE とサーバーによって実行時に決定されます。

sslTrustStoreLocation
オプション。新しい接続の暗号トラストストア・ファイルのロケーションを指定します。この値は、**sslConnection** が true に設定されている場合のみ使用します。

sslTrustStorePassword
オプション。暗号トラストストア・ファイルにアクセスするためのパスワードを指定します。この値は、**sslConnection** が true に設定されている場合のみ使用します。

sslKeyStoreLocation
オプション。新しい接続の暗号鍵ストア・ファイルのロケーションを指定します。この値は、**sslConnection** が true に設定されている場合のみ使用します。

sslKeyStorePassword
オプション。暗号鍵ストア・ファイルにアクセスするためのパスワードを指定します。この値は、**sslConnection** が true に設定されている場合のみ使用します。

loginTimeout
オプション。接続初期設定またはサーバー要求時にドライバーがサーバーからの応答を待つ秒数を指定します。この時間が過ぎるとタイムアウトになります。このプロパティーには、秒数として負でない整数を指定します。タイムアウトの長さを無制限にする場合は、このプロパティーを 0 に設定してください。タイプ 2 コネクティビティーの使用時にはこのプロパティーを設定しないでください。

user RACF 管理者によって提供された IMS Connect に接続するためのユーザー名。タイプ 2 コネクティビティーの使用時にはこのプロパティーを設定しないでください。

password
RACF 管理者によって提供された IMS Connect に接続するためのパスワード。タイプ 2 コネクティビティーの使用時にはこのプロパティーを設定しないでください。

dbViewLocation
オプション。databaseView メタデータ・クラスへの絶対パスを指定します。このプロパティーを使用して、プロジェクト・パスに配置されていないメタデータ・クラスを組み込むことができます。

treatInvalidDecimalAsNull
オプション。Java アプリケーションで無効と見なされる特定の 10 進値 (無効な符号ビットが設定された PACKEDDECIMAL および ZONEDDECIMAL など) をヌルとして解釈するかどうかを指示します。デフォルトでは、このプロパティーは「false」であるため、Java アプリケーションが無効値を処理しているときには変換例外がスローされます。

3. 接続要求プロパティーを PSBFactory クラスに受け渡して、PSB インスタンスを作成します。 PSB インスタンスが正常に作成されると、データベースへの接続が確立されます。
4. IMS Universal DL/I ドライバー・アプリケーションから IMS データベースへの接続が終了した場合は、PSB インスタンスで close メソッドを呼び出して、データベースとの接続を閉じる必要があります。

例: タイプ 4 接続

以下のサンプル・コードは、IMS Universal DL/I ドライバー・アプリケーションから IMS データベースへのタイプ 4 接続を作成する方法を示しています。

```
IMSConnectionSpec connSpec = IMSConnectionSpecFactory.createIMSConnectionSpec();
connSpec.setDatastoreName("SYS1");
connSpec.setDatastoreServer("9.876.543.21");
connSpec.setPortNumber(8888);
connSpec.setDatabaseName("class://testdb.jdbo.HospitalDatabaseView");
connSpec.setSSLConnection(true);
connSpec.setLoginTimeout(10);
connSpec.setUser("usr");
connSpec.setPassword("usrpwd");
connSpec.setDriverType(IMSConnectionSpec.DRIVER_TYPE_4);
PSB psb = PSBFactory.createPSB(connSpec);
```

関連タスク:

820 ページの『SSL サポート用の IMS Universal ドライバーの構成』

関連資料:

Java API 文書 (Javadoc) (アプリケーション・プログラミング API)

DL/I 操作を実行するための IMS Universal DL/I ドライバー・インターフェース

従来の IMS アプリケーションでは、DL/I 呼び出しを実行して、データの挿入、更新、削除、またはリトリーブを行います。 IMS Universal DL/I ドライバー・アプリケーションで同じ機能を実行するには、メソッドを呼び出します。

メソッドは以下のインターフェースで定義されます。

- プログラム仕様ブロック (PSB) インターフェースは、IMS データベースに接続するために使用します。 PSB インターフェースは、PSB に含まれる任意のプログラム連絡ブロック (PCB) へのハンドルを取得するために使用します。 PCB ハンドルは、PCB によって参照される特定のデータベースにアクセスするために使用します。
- PCB インターフェースは、IMS データベース内でのカーソル位置を表します。 PCB インターフェースは、Get Unique (GU)、Get Next (GN)、Get Next Within Parent (GNP)、Insert (ISRT)、Replace (REPL)、および Delete (DLET) を含む、DL/I メッセージ呼び出し機能をサポートします。 PCB インターフェースは、セグメントリトリーブ引数の非修飾リストをリトリーブすることができ、バッチ・リトリーブ、更新、および削除操作を実行できます。また、PCB インターフェースは、最新の DL/I 呼び出しに関連付けられているアプリケーション・インターフェース・ブロック (AIB) を返すためにも使用できます。
- SSAList インターフェースは、特定のデータベース呼び出しのターゲットとなるセグメントを指定するために使用される、セグメント検索引数 (SSA) のリストを表します。 SSAList インターフェースは、SSA を構成し、SSA 用のコマンド・コードおよびロック・クラスを設定するために使用します。初期修飾ステートメントを設定し、セグメント・フィールドの値に基づいて追加の修飾子を付加することで、DL/I 呼び出しでターゲットとなるセグメントを制限できます。また、データベース・リトリーブ呼び出しから返されるフィールドを指定することもできます。
- Path インターフェースは、DL/I リトリーブまたは更新操作を目的としたデータベース・レコードを表します。 Path インターフェースは、ルート・セグメントに最も近い最高位レベルのセグメントから始まり最下位レベルのセグメントに至るまでの、特定のデータベース階層パスにあるすべてのセグメント・インスタン

スの連結と見なすことができます。 Path インターフェースは、階層パスに配置されている任意のセグメント・フィールドの値を設定したりリトリーブしたりできます。

- PathSet インターフェースは、バッチ・リトリーブ操作によって返された Path オブジェクトの集合へのアクセスを提供します。
- AIB インターフェースおよびデータベース PCB (DBPCB) インターフェースは、DL/I 呼び出しの結果として IMS によって返された有用な情報を返します。
- GSAMPCB インターフェースは、GSAM PCB を表し、本質的には GSAM データベースでのカーソル位置です。このインターフェースは、DL/I 呼び出しに類似した呼び出しを指定して、GSAM データベースへのデータ・アクセスを提供します。
- RSA インターフェースは、GSAM データベースでのカーソル位置へのキーとなる、GSAM データベース・レコード検索引数を表します。

関連資料:

Java API 文書 (Javadoc) (アプリケーション・プログラミング API)

SSAList インターフェースを使用したセグメント検索引数の指定

SSAList インターフェースは、特定のデータベース呼び出しでターゲットとなるセグメントの指定に使用される、一連のセグメント検索引数のリストを表します。

SSAList インターフェースは、リスト内に各セグメント検索引数 (SSA) を構成し、SSA 用のコマンド・コードおよびロック・クラスを設定するために使用します。SSAList 内の各 SSA は、非修飾または修飾のどちらでも構いません。

さらに、アプリケーションでは、markFieldForRetrieval メソッドまたは markAllFieldsForRetrieval メソッドを使用して、データベース・リトリーブ呼び出しから返されるセグメント・フィールドを指定できます。 IMS のデフォルトに従って、SSAList で指定されている最下位レベルのセグメントのすべてフィールドには、最初からリトリーブ用のマークが付けられています。

- 非バッチ DL/I データ・リトリーブまたは更新操作の場合は、getPathForRetrieveReplace メソッドを使用します。
- DL/I 挿入呼び出しの場合は、getPathForInsert メソッドを使用します。
- バッチ更新操作の場合は、getPathForBatchUpdate メソッドを使用します。

以下の例は、SSAList インターフェースを使用したセグメント検索引数の指定方法を示しています。この例は、病院データベースに基づいています。

非修飾 SSAList の作成

この例は、セグメント「DOCTOR」のすべてのフィールドで構成される Path を返します。

```
SSAList ssaList = pcb.getSSAList("HOSPITAL","DOCTOR");
Path path = ssaList.getPathForRetrieveReplace();
pcb.getUnique(path, ssaList, false);
```

前の例では、ssaList は、最高位のセグメント (「HOSPITAL」) から最下位のセグメント (「DOCTOR」) までの階層パスのすべてのセグメントを表しています。ssaList のロックは次のようになります。

```
HOSPITALb
WARDbbbbb
PATIENTbb
ILLNESSbb
TREATMNTb
DOCTORbbb
```

修飾 SSAList の作成

SSAList は、リトリーブまたは更新する階層パスのセグメントをフィルター操作するために修飾されます。修飾 SSAList を作成するための一般的なステップは、次のとおりです。

1. getSSAList メソッドを使用して PCB から非修飾 SSAList を取得します。
2. addInititalQualification メソッドを使用して、getSSAList メソッドから返される SSAList のセグメントに対して、初期検索条件を指定します。 SSAList に示される各セグメントに対して、そのセグメントの初期修飾を指定する 1 つの呼び出しを行うことができます。セグメントは、名前を使用するか、または SSAList 内のそのセグメントを表す 1 ベース・オフセットの SSA を使用して参照できます。 1 つのセグメントに対して複数の addInitialQualification ステートメントを使用している場合は、例外が throw されます。addInitialQualification メソッドの関係演算子 (**relationalOp**) パラメーターは、セグメントが修飾されるために満たすべき条件基準を示します。有効な関係演算子は次のとおりです。
   - EQUALS
   - GREATER_OR_EQUAL
   - GREATER_THAN
   - LESS_OR_EQUAL
   - LESS_THAN
   - NOT_EQUAL
3. 追加の検索基準を指定するには、appendQualification メソッドを使用します。各セグメントに対して、appendQualification メソッドへの複数の呼び出しを行い、複数の修飾ステートメントを追加できます。 appendQualification メソッドのブール演算子 (**booleanOp**) パラメーターは、この修飾が論理的に前の修飾と接続される方法を示します。有効なブール演算子は次のとおりです。
   - AND
   - OR
   - INDEPENDENT_AND
4. DL/I コマンド・コードおよびロック・クラスを設定して、SSAList を修飾することもできます。サポートされる DL/I コマンド・コードには、以下のものがあります。
   - CC_A: A コマンド・コード (明確な位置決め)。
   - CC_C: C コマンド・コード (連結キー)。 addConcatenatedKey メソッドを使用して、連結キーをセグメントに追加します。

- CC_D: D コマンド・コード (パス CALL)
- CC_F: F コマンド・コード (最初のオカレンス)
- CC_G: G コマンド・コード (ランダム化の回避)。
- CC_L: L コマンド・コード (最後のオカレンス)
- CC_N: N コマンド・コード (パス CALL 無視)
- CC_O: O コマンド・コード (フィールド名またはセグメントの位置と長さを含む)
- CC_P: P コマンド・コード (親子関係の設定)
- CC_U: U コマンド・コード (このレベルでの位置を保持)
- CC_V: V コマンド・コード (このレベルおよびすべての上位レベルでの位置を保持)

ロック・クラスを使用して、プログラムがコミット・ポイントに到達するまで他のプログラムがセグメントを更新するのを防ぐことができます。セグメントにロック・クラスを追加するには addLockClass メソッドを使用します。サポートされるロック・クラスの文字は「A」から「J」です。ロック・クラスの動作は、そのロック・クラス文字に「Q」コマンド・コードを使用するのと同じです。

次のサンプル・コードでは、単一の初期修飾ステートメントで修飾された SSAList を指定および使用してデータをリトリーブする方法を示しています。

```
SSAList ssaList = pcb.getSSAList("HOSPITAL","DOCTOR");
ssaList.addInitialQualification("PATIENT","PATNAME",SSAList.EQUALS,"ANDREA SMITH");
ssaList.markFieldForRetrieval("ILLNESS","ILLNAME",true);
ssaList.markFieldForRetrieval("TREATMNT","TREATMNT",true);
Path path = ssaList.getPathForRetrieveReplace();
pcb.getUnique(path, ssaList, false);
```

上記のサンプル・コードの場合、ssaList は次のようになります。

```
HOSPITALb
WARDbbbbb
PATIENTb(PATNAMEbEQANDREAbSMITHbbbbb)
ILLNESSb*D
TREATMNT*D
```

上記のサンプル・コードでは、PATNAME が「ANDREA SMITH」であるすべてのレコードについて、以下の情報がリトリーブされます。

- ILLNESS セグメントの ILLNAME フィールド。
- TREATMNT セグメントの TREATMNT フィールド。
- DOCTOR セグメントのすべてのフィールド (デフォルトでは、IMS は SSAList で指定されている最下位レベルのセグメントのすべてフィールドを返します)。

次にサンプル・コードでは、複数の修飾ステートメントで修飾された SSAList を指定してデータをリトリーブする方法を示しています。

```
SSAList ssaList = pcb.getSSAList("HOSPITAL","WARD");
ssaList.addInitialQualification("WARD","NURCOUNT",SSAList.GREATER_THAN,4);
ssaList.appendQualification("WARD",SSAList.AND,"DOCCOUNT",SSAList.GREATER_THAN, 2);
```

上記のサンプル・コードの場合、ssaList は次のようになります。

```
HOSPITALb
WARDbbbb(NURCOUNTGT4&DOCCOUNTGT2;)
```

次の例では、コマンド・コード CC_L (「最後のオカレンス」を意味) が指定された修飾 SSAList を指定して、「SANTA TERESA」病院に最後に入院した患者を検索する方法を示しています。

```
SSAList ssaList = pcb.getSSAList("HOSPITAL","PATIENT");
ssaList.addInitialQualification
   ("HOSPITAL","HOSPNAME",SSAList.EQUALS,"SANTA TERESA");
ssaList.addCommandCode("PATIENT",SSAList.CC_L);
Path path = ssaList.getPathForRetrieveReplace();
pcb.getUnique(path,ssaList,false);
```

上記のサンプル・コードの場合、ssaList は次のようになります。

```
HOSPITAL(HOSPNAMEEQSANTAbTERESAbbbbb)
WARDbbbbb
PATIENTb*L
```

SSAList のデバッグ

SSAList のセグメント検索引数が正しいかどうかを識別することでデバッグする必要がある場合は、今後のデバッグのために buildSSAListInBytes メソッドを使用して、DL/I 形式で SSAList を作成します。

```
byte[][] ssaListInBytes = ssaList.buildSSAListInBytes();
```

返されるバイト配列の各セグメント検索引数を繰り返し、これをプリントしてセグメント検索引数が正当なものであることを確認します。

関連概念:

207 ページの『セグメント検索引数 (SSA)』

関連資料:

275 ページの『SSA コーディング形式』

711 ページの『病院データベースの例』

コマンド・コードの参照情報 (アプリケーション・プログラミング API)

IMS Universal DL/I ドライバー・アプリケーションでのデータのリトリーブ

IMS Universal DL/I ドライバーは、DL/I セマンティクスを反映するデータ・リトリーブのサポートを提供します。

データベースからセグメントをリトリーブするための一般的なステップを以下に示します。

1. データベースを表す PCB インスタンスから SSAList インスタンスを取得します。
2. オプションとして、SSAList インスタンスに修飾ステートメントを追加することができます。
3. リトリーブするセグメント・フィールドを指定します。 markFieldForRetrieval メソッドを使用して単一のフィールドにマークを付けるか、または markAllFieldsForRetrieval メソッドを使用してセグメントのすべてのフィールドにマークを付けます。 IMS のデフォルトに従って、SSAList インスタンスで指定されている最下位レベルのセグメントのすべてフィールドには、最初からリトリーブ用のマークが付けられています。 SSAList インスタンスで指定された

最下位レベル・セグメントの 1 つまたは複数のフィールドにリトリーブ用のマークが付けられている場合、明示的にマークが付けられたフィールドのみがリトリーブされます。

4. 前のステップの SSAList インスタンスを使用して getPathForRetrieveReplace メソッドを呼び出し、Path インスタンスを取得します。リトリーブ呼び出しを行うと、結果の Path オブジェクトには、リトリーブ用にマークが付けられたすべてのフィールドが入ります。
5. PCB インターフェースから次のメソッドのいずれかを使用して、DL/I リトリーブ操作を呼び出します。

DL/I リトリーブ・メソッド用の **Java API**	使用法
getUnique	特定の固有セグメントをリトリーブします。このメソッドは、DL/I Get Unique (GU) データベース呼び出しと同じ機能を提供します。 isHoldCall パラメーターが true に設定されている場合、この呼び出しは DL/I Get Hold Unique (GHU) データベース呼び出しと同じように動作します。
getNext	パス内の次のセグメントをリトリーブします。このメソッドは、DL/I Get Next (GN) データベース呼び出しと同じ機能を提供します。 isHoldCall パラメーターが true に設定されている場合、この呼び出しは DL/I Get Hold Next (GHN) データベース呼び出しと同じように動作します。
getNextWithinParent	同じ親内の次のセグメントをリトリーブします。このメソッドは、DL/I Get Next Within Parent (GNP) データベース呼び出しと同じ機能を提供します。 isHoldCall パラメーターが true に設定されている場合、この呼び出しは DL/I Get Hold Next Within Parent (GHNP) データベース呼び出しと同じように動作します。
batchRetrieve	1 回の呼び出しで複数のセグメントをリトリーブします。このメソッドの使用法について詳しくは、『Java API for DL/I アプリケーションでのバッチ・データ・リトリーブ』を参照してください。

6. リトリーブ呼び出しが実行された後に、Path オブジェクトから、リトリーブ対象のフィールドの値を読み取ります。

IMS Universal DL/I ドライバーのデータ・リトリーブの例

次のコード・フラグメントは、getUnique メソッドおよび getNext メソッドを使用して、病院データベースから病院名 (HOSPNAME)、病棟名 (WARDNAME)、患者数 (PATCOUNT)、看護士数 (NURCOUNT)、および医師数 (DOCCOUNT) フィールドをリトリーブする方法を示しています。

```
import com.ibm.ims.dli.*;


public class HospitalDLIReadClient {

   public static void main(String[] args) {
     PSB psb = null;
     PCB pcb = null;
     SSAList ssaList = null;
     Path path = null;
     PathSet pathSet = null;

     try {
        // establish a database connection
        IMSConnectionSpec connSpec
           = IMSConnectionSpecFactory.createIMSConnectionSpec();
        connSpec.setDatastoreName("IMS1");
        connSpec.setDatastoreServer("ecdev123.svl.ibm.com");
        connSpec.setPortNumber(5555);
        connSpec.setMetadataURL("class://BMP266.BMP266DatabaseView");
        connSpec.setUser("usr");
        connSpec.setPassword("password");
    connSpec.setDriverType(IMSConnectionSpec.DRIVER_TYPE_4);
        psb = PSBFactory.createPSB(connSpec);
        System.out.println("**** Created a connection to the IMS database");

        pcb = psb.getPCB("PCb01");
        System.out.println("**** Created PCB object");

        // specify the segment search arguments
        ssaList = pcb.getSSAList("HOSPITAL", "WARD");
        // add the initial qualification
        ssaList.addInitialQualification("HOSPITAL", "HOSPCODE",
           SSAList.GREATER_OR_EQUAL, 444);
        // specify the fields to retrieve
        ssaList.markFieldForRetrieval("HOSPITAL", "HOSPNAME", true);
        ssaList.markAllFieldsForRetrieval("WARD", true);
        ssaList.markFieldForRetrieval("WARD", "WARDNO", false);
        System.out.println("**** Created SSAList object");

        // obtain a Path containing the segments that match the SSAList criteria
        path = ssaList.getPathForRetrieveReplace();
        System.out.println("**** Created Path object");

        // issue a DL/I GU call to retrieve the first segment on the Path
       if (pcb.getUnique(path, ssaList, true) {
          System.out.println("HOSPNAME: "+ path.getString("HOSPITAL", "HOSPNAME"));
          System.out.println("WARDNAME: "+ path.getString("WARD", "WARDNAME"));
          System.out.println("PATCOUNT: "+ path.getInt("WARD", "PATCOUNT"));
          System.out.println("NURCOUNT: "+ path.getInt("WARD", "NURCOUNT"));
          System.out.println("DOCCOUNT: "+ path.getShort("WARD", "DOCCOUNT"));
       }

       // issue multiple DL/I GN calls until there are no more segments to retrieve
       while (pcb.getNext(pat, ssaList, true) {
          System.out.println("HOSPNAME: "+ path.getString("HOSPITAL", "HOSPNAME"));
          System.out.println("WARDNAME: "+ path.getString("WARD", "WARDNAME"));
          System.out.println("PATCOUNT: "+ path.getInt("WARD", "PATCOUNT"));
          System.out.println("NURCOUNT: "+ path.getInt("WARD", "NURCOUNT"));
          System.out.println("DOCCOUNT: "+ path.getShort("WARD", "DOCCOUNT"));
       }

       // close the database connection
       psb.close();
       System.out.println("**** Disconnected from IMS database");

     } catch (DLIException e) {
```

```
            System.out.println(e);
            System.exit(0);
        }
    }
}
```

関連概念:

799 ページの『SSAList インターフェースを使用したセグメント検索引数の指定』

関連タスク:

793 ページの『IMS Universal DL/I ドライバー・アプリケーションを作成するための基本ステップ』

『IMS Universal DL/I ドライバー・アプリケーションでのバッチ・データ・リトリーブ』

関連資料:

787 ページの『データ型のリトリーブおよび変換のメソッド』

IMS Universal DL/I ドライバー・アプリケーションでのバッチ・データ・リトリーブ

1 回の呼び出しで複数のセグメントをリトリーブするには、batchRetrieve メソッドを使用します。

クライアント・アプリケーションで複数の GU および GN 呼び出しを行う代わりに、IMS がすべての GU および GN 処理を実行し、1 回のバッチ・ネットワーク操作でクライアントに結果を返します。フェッチ・サイズ・プロパティーによって、各バッチ・ネットワーク操作で返されるデータの量が決まります。

バッチ・データ・リトリーブを実行するには、以下のようにします。

1. データベースを表す PCB インスタンスから SSAList インスタンスを取得します。
2. オプションとして、SSAList インスタンスに修飾ステートメントを追加することができます。
3. リトリーブするセグメント・フィールドを指定します。 markFieldForRetrieval メソッドを使用して単一のフィールドにマークを付けるか、または markAllFieldsForRetrieval メソッドを使用してセグメントのすべてのフィールドにマークを付けます。 IMS のデフォルトに従って、SSAList インスタンスで指定されている最下位レベルのセグメントのすべてフィールドには、最初からリトリーブ用のマークが付けられています。
4. オプションで、フェッチ・サイズ・プロパティーを設定します。 フェッチ・サイズを指定すると、1 回のバッチ操作でデータベースから取り出すレコードの数に関して、IMS Universal DL/I ドライバーに示唆を与えることができます。詳しくは、『フェッチ・サイズの設定による照会パフォーマンスの向上』を参照してください。
5. パラメーターとして上記の SSAList インスタンスを指定して、batchRetrieve メソッドを呼び出します。 batchRetrieve メソッドは、SSAList で指定された基準を満たすレコードのリストが含まれている、PathSet を返します。
6. リトリーブ呼び出しが実行された後に、Path オブジェクトから、リトリーブ対象のフィールドの値を読み取ります。

IMS Universal DL/I ドライバーのバッチ・データ・リトリーブの例

次のコード・フラグメントは、batchRetrieveメソッドを使用して、病院データベースから病院名 (HOSPNAME)、病棟名 (WARDNAME)、患者数 (PATCOUNT)、看護士数 (NURCOUNT)、および医師数 (DOCCOUNT) フィールドをリトリーブする方法を示しています。

```
import com.ibm.ims.dli.*;


public class HospitalDLIReadClient {

   public static void main(String[] args) {
     PSB psb = null;
     PCB pcb = null;
     SSAList ssaList = null;
     Path path = null;
     PathSet pathSet = null;

     try {
        // establish a database connection
        IMSConnectionSpec connSpec
           = IMSConnectionSpecFactory.createIMSConnectionSpec();
        connSpec.setDatastoreName("IMS1");
        connSpec.setDatastoreServer("ecdev123.svl.ibm.com");
        connSpec.setPortNumber(5555);
        connSpec.setMetadataURL("class://BMP266.BMP266DatabaseView");
        connSpec.setUser("usr");
        connSpec.setPassword("password");
    connSpec.setDriverType(IMSConnectionSpec.DRIVER_TYPE_4);

        psb = PSBFactory.createPSB(connSpec);
        System.out.println("**** Created a connection to the IMS database");

        pcb = psb.getPCB("PCb01");
        System.out.println("**** Created PCB object");

        // specify the segment search arguments
        ssaList = pcb.getSSAList("HOSPITAL", "WARD");
        // add the initial qualification
        ssaList.addInitialQualification("HOSPITAL", "HOSPCODE",
           SSAList.GREATER_OR_EQUAL, 444);
        // specify the fields to retrieve
        ssaList.markFieldForRetrieval("HOSPITAL", "HOSPNAME", true);
        ssaList.markAllFieldsForRetrieval("WARD", true);
        ssaList.markFieldForRetrieval("WARD", "WARDNO", false);
        System.out.println("**** Created SSAList object");

        // issue the database call to perform a batch retrieve operation
        pathSet = pcb.batchRetrieve(ssaList);
        System.out.println("**** Batch Retrieve returned without exception");
        System.out.println("**** Created PathSet object");

        while(pathSet.hasNext()){
           path = pathSet.next();

           System.out.println("HOSPNAME: "+ path.getString("HOSPITAL", "HOSPNAME"));
           System.out.println("WARDNAME: "+ path.getString("WARD", "WARDNAME"));
           System.out.println("PATCOUNT: "+ path.getInt("WARD", "PATCOUNT"));
           System.out.println("NURCOUNT: "+ path.getInt("WARD", "NURCOUNT"));
           System.out.println("DOCCOUNT: "+ path.getShort("WARD", "DOCCOUNT"));
        }
        System.out.println("**** Fetched all rows from PathSet");

        // close the database connection
        psb.close();
```

```
            System.out.println("**** Disconnected from IMS database");
        } catch (DLIException e) {
            System.out.println(e);
            System.exit(0);
        }
    }
}
```

関連概念:

799 ページの『SSAList インターフェースを使用したセグメント検索引数の指定』

関連資料:

787 ページの『データ型のリトリーブおよび変換のメソッド』

フェッチ・サイズの設定による照会パフォーマンスの向上:

バッチ・リトリーブ・モードでリトリーブするレコードの数を設定して、照会パフォーマンスを最適化することができます。

IMS Universal DL/I ドライバーには、SSAList で指定されたセグメント検索引数の基準に一致する 1 つ以上のセグメントを含む Path インスタンスによって、行のリストが表されます。フェッチ・サイズ は、ネットワーク呼び出しごとに IMS データベースから物理的にリトリーブされる行数です。これは内部的に自動設定されます。また、PCB インターフェースから setFetchSize メソッドを使用してフェッチ・サイズを設定することもできます。フェッチ・サイズを設定すると、単一の要求で同時に複数の行を返すことができるため、次行のリトリーブを要求する各アプリケーションは、必ずしもネットワーク要求を出すことにはなりません。フェッチ・サイズが *n* で、IMS Universal DL/I ドライバー・アプリケーションがバッチ・リトリーブ操作時に、前の行数 *n* より多くの行を要求する場合、アプリケーションに代わって別のネットワーク呼び出しが発行され、セグメントリトリーブ引数の基準に一致する次の行数 *n* の行がリトリーブされます。

データ型のリトリーブおよび変換のメソッド

IMS Universal JDBC ドライバーでは、ResultSet インターフェース (java.sql.ResultSet) を使用してデータをリトリーブし、データベース・メタデータで定義されているデータ型から Java アプリケーションで必要なデータ型に変換することができます。 同様に、IMS Universal DL/I ドライバーでは、Path インターフェースを使用して、データをリトリーブし、Java データ型に変換することができます。

以下の表では、特定の Java データ型のデータ・アクセスに使用できる get メソッドを示しています。IMS Universal JDBC ドライバーの場合は ResultSet インターフェースであり、IMS Universal DL/I ドライバーの場合は Path インターフェースです。

「切り捨てまたはデータ欠落なし」列は、特定の getXXX メソッドでアクセスするように設計されているデータ型を示しています。この列のデータ型をそれに対応するメソッドで使用する場合は、切り捨てまたはデータ欠落は生じません。「有効、データ保全性なし」列のデータ型は、他のすべての有効な呼び出しを示しています。ただし、特定の getXXX メソッドを使用してアクセスした場合のデータ保全性

は保証できません。データ型がどちらの列にも属していない場合、そのデータ型に特定の getXXX メソッドを使用すると、例外が出される結果になります。

表 *109.* データ型をリトリーブする *ResultSet.getXXX* および *Path.getXXX* メソッド

ResultSet.getXXX メソッドまたは **Path.getXXX** メソッド	データ型 **(**リストされていないものは例外が出される結果になる**)**	
	切り捨てまたはデータの損失無し	データ保全性が保証外の正当なメソッド
getByte	TINYINT UTINYINT	SMALLINT INTEGER BIGINT FLOAT DOUBLE BIT CHAR VARCHAR PACKEDDECIMAL[1] ZONEDDECIMAL[1]
getShort	SMALLINT USMALLINT	TINYINT INTEGER BIGINT FLOAT DOUBLE BIT CHAR VARCHAR PACKEDDECIMAL[1] ZONEDDECIMAL[1]
getInt	INTEGER UINTEGER	TINYINT SMALLINT BIGINT FLOAT DOUBLE BIT CHAR VARCHAR PACKEDDECIMAL[1] ZONEDDECIMAL[1]
getLong	BIGINT UBIGINT	TINYINT SMALLINT INTEGER FLOAT DOUBLE BIT CHAR VARCHAR PACKEDDECIMAL[1] ZONEDDECIMAL[1]

表 109. データ型をリトリーブする *ResultSet.getXXX* および *Path.getXXX* メソッド (続き)

ResultSet.getXXX メソッドまたは **Path.getXXX** メソッド	データ型 **(**リストされていないものは例外が出される結果になる**)**	
	切り捨てまたはデータの損失無し	データ保全性が保証外の正当なメソッド
getFloat	FLOAT	TINYINT SMALLINT INTEGER BIGINT DOUBLE BIT CHAR VARCHAR PACKEDDECIMAL[1] ZONEDDECIMAL[1]
getDouble	DOUBLE	TINYINT SMALLINT INTEGER BIGINT FLOAT BIT CHAR VARCHAR PACKEDDECIMAL[1] ZONEDDECIMAL[1]
getBoolean	BIT	TINYINT SMALLINT INTEGER BIGINT FLOAT DOUBLE CHAR VARCHAR PACKEDDECIMAL[1] ZONEDDECIMAL[1]
getString	CHAR VARCHAR	TINYINT SMALLINT INTEGER BIGINT FLOAT DOUBLE BIT PACKEDDECIMAL[1] ZONEDDECIMAL[1] BINARY DATE TIME TIMESTAMP

表 109. データ型をリトリーブする *ResultSet.getXXX* および *Path.getXXX* メソッド (続き)

ResultSet.getXXX メソッドまたは **Path.getXXX** メソッド	データ型 **(**リストされていないものは例外が出される結果になる**)**	
	切り捨てまたはデータの損失無し	データ保全性が保証外の正当なメソッド
getBigDecimal	BINARY[3] PACKEDDECIMAL[1] ZONEDDECIMAL[1]	TINYINT SMALLINT INTEGER BIGINT FLOAT DOUBLE BIT CHAR VARCHAR
getClob	CLOB[2]	その他はすべて例外の結果
getBytes	BINARY	その他はすべて例外の結果
getDate	日付	CHAR VARCHAR TYIMESTAMP
getTime	TIME	CHAR VARCHAR TIMESTAMP
getTimestamp	TIMESTAMP	CHAR VARCHAR DATE TIME

注**:**

1. PACKEDDECIMAL および ZONEDDECIMAL は、IMS Universal JDBC ドライバーおよび IMS Universal DL/I ドライバー用の拡張データ型です。その他のデータ型は、SQL92 で定義されている標準 SQL データ型です。制約事項**:** PACKEDDECIMAL および ZONEDDECIMAL データ型は、SIGN LEADING モードおよび SIGN SEPARATE モードはサポートしません。これらの 2 つのデータ・タイプについては、符号情報が常に Sign Trailing (符号末尾) 方式で保管されます。
2. CLOB データ型は、XML データのリトリーブおよび保管についてのみサポートされます。
3. BINARY データ型は、バイナリー形式コンバーターと一緒に使用される10 進データでのみ有効です。

フィールド・タイプが PACKEDDECIMAL または ZONEDDECIMAL のいずれかの場合、型修飾子は、フィールドのレイアウトを表す COBOL PICTURE ストリングです。 9、P、V、および S の有効な組み合わせを含むすべての COBOL PICTURE ストリングはサポートされます。 PICTURE ストリングの拡張は、自動的に処理されます。例えば、「9(5)」は、有効な PICTURE ストリングです。ゾーン 10 進数の場合、小数点も PICTURE ストリングで使用できます。`PIC 9(06)V99 COMP` および `PIC 9(06)V99 COMP-4` は、BINARY 10 進データに対して有効な PICTURE 文節です。

フィールドに DATE、TIME、または TIMESTAMP データが含まれている場合、データ形式は型修飾子で示されます。例えば、*ddMMyyyy* という型修飾子は、データが次のような形式であることを意味します。

```
11122015 is December 11, 2015
```

DATE および TIME 型の場合、java.text.SimpleDateFormat クラスのすべてのフォーマット・オプションがサポートされます。

TIMESTAMP 型の場合、フォーマット・オプション「f」が、ナノ秒用に使用できます。 TIMESTAMP には、最大で 9 つの「f」を入れることができ、ミリ秒になると「S」オプションで置き換えられます。その場合には、「fff」はミリ秒の精度を示します。 TIMESTAMP の形式例は、以下のとおりです。

```
yyyy-mm-dd hh:mm:ss.fffffffff
```

IMS Universal DL/I ドライバー・アプリケーションでのデータの作成および挿入

データベースに新規セグメントを挿入するには、PCB インターフェースで create メソッドまたは insert メソッドを使用します。

IMS Universal DL/I ドライバーでは、insert メソッドおよび create メソッドにより、DL/I ISRT 呼び出しと類似の機能が提供されます。 insert メソッドは、DL/I 操作の結果を示す IMS 状況コードを返します。これに対し、create メソッドは、作成されたセグメントの数を返します (これは常時 1 になります)。キー・フィールドが設定されていない場合は、例外が throw されます。

新規セグメントをデータベースに追加するための、一般的なステップを以下に示します。

1. データベースを表す PCB インスタンスから SSAList インスタンスを取得します。
2. オプションとして、SSAList に修飾ステートメントを追加できます。
3. 前のステップの SSAList インスタンスを使用して getPathForInsert メソッドを呼び出し、Path インスタンスを取得します。 getPathForInsert メソッドは、パラメーターとして SSAList にある既存のセグメント名を使用します。パラメーターは、新規セグメントのセグメント・タイプの名前を示します。例えば、patient という新規セグメントを追加する場合、セグメント名 PATIENT をパラメーターとして渡します。
4. 上記のステップの Path インスタンスを使用して、新規セグメントのフィールド値を設定します。
5. insert メソッドまたは create メソッドを呼び出して、新規セグメントを追加します。

IMS Universal DL/I ドライバーでの create および insert の例

以下のコード・フラグメントは、create メソッドおよび insert メソッドを使用してデータベースに新規 patient セグメントおよび illness セグメントを追加する方法を示しています。ここで、hospital 名は「SANTA TERESA」であり、ward 名は「GENERAL」です。

```
SSAList ssaList = pcb.getSSAList("HOSPITAL","PATIENT");
ssaList.addInitialQualification("HOSPITAL","HOSPNAME",
SSAList.EQUALS,"SANTA TERESA");
ssaList.addInitialQualification("WARD","WARDNAME",
SSAList.EQUALS,"GENERAL");

Path path = ssaList.getPathForInsert("PATIENT");
path.setString("PATIENT", "PATNUM", "0088");
path.setString("PATIENT", "PATNAME", "JACK KIRBY");
int i = pcb.create(path, ssaList); // returns i = 1 if successful
System.out.println(i);

SSAList ssaList2 = pcb.getSSAList("HOSPITAL","ILLNESS");
ssaList2.addInitialQualification("HOSPITAL","HOSPNAME",
   SSAList.EQUALS,"SANTA TERESA");
ssaList2.addInitialQualification("WARD","WARDNAME",
   SSAList.EQUALS,"GENERAL");
ssaList2.addInitialQualification("PATIENT","PATNUM",
   SSAList.EQUALS,"0088");

Path path2 = ssaList2.getPathForInsert("ILLNESS");
path2.setString("ILLNAME", "APPENDICITIS");
short status = pcb.insert(path2, ssaList2);
```

以下のコーディング例は、上と同じ処理を単一の呼び出しで実行する別の方法を示しています。

```
SSAList ssaList = pcb.getSSAList("HOSPITAL","ILLNESS");
ssaList.addInitialQualification("HOSPITAL","HOSPNAME",
   SSAList.EQUALS,"SANTA TERESA");
ssaList.addInitialQualification("WARD","WARDNAME",
   SSAList.EQUALS,"GENERAL");
ssaList.addCommandCode("PATIENT", SSAList.CC_D);

Path path = ssaList.getPathForInsert("PATIENT");
path.setString("PATIENT", "PATNUM", "0088");
path.setString("PATIENT", "PATNAME", "JACK KIRBY");
path.setString("ILLNAME", "APPENDICITIS");

int i = pcb.create(path, ssaList); // returns i = 1 if successful
```

重要: データベースへの変更を持続的なものとするために、アプリケーションでは PSB を割り振り解除する前に PSB.commit メソッドを呼び出す必要があります。そうしないと、変更は最後に commit が呼び出された時点までロールバックされることになります。

関連概念:

799 ページの『SSAList インターフェースを使用したセグメント検索引数の指定』

IMS Universal DL/I ドライバー・アプリケーションでのデータの更新

データベース内の既存のセグメントを更新するには、PCB インターフェースで replace メソッドを使用します。

IMS Universal DL/I ドライバーでは、replace メソッドにより、DL/I REPL 呼び出しと類似の機能が提供されます。 replace メソッドは、DL/I 操作の結果を示す IMS 状況コードを返します。

データベース内の既存のセグメントを更新するための一般的なステップを以下に示します。

1. データベースを表す PCB インスタンスから SSAList インスタンスを取得します。
2. オプションとして、SSAList インスタンスに修飾ステートメントを追加することができます。詳しくは、『SSAList インターフェースを使用したセグメント検索引数の指定』を参照してください。
3. 前のステップの SSAList インスタンスを使用して getPathForRetrieveReplace メソッドを呼び出し、Path インスタンスを取得します。
4. 前のステップの Path インスタンスを使用して、セグメントを更新するようにフィールド値を設定します。
5. 置換呼び出しを発行する前に Hold 操作を実行します。 Hold 操作は、getUnique、getNext、または getNextWithinParent メソッド呼び出しのいずれでも構いません。
6. replace メソッドを呼び出してセグメントを更新します。

IMS Universal DL/I ドライバーによる更新の例

以下のコード・フラグメントは、replace メソッドを使用して患者レコードの患者名を更新する方法を示しています。ここで、患者名は「ANDREA SMITH」、病棟名は「SURG」、および病院名は「ALEXANDRIA」です。

```
SSAList ssaList = pcb.getSSAList("HOSPITAL","PATIENT");
ssaList.addInitialQualification("HOSPITAL","HOSPNAME",SSAList.EQUALS,"ALEXANDRIA");
ssaList.addInitialQualification("WARD","WARDNAME",SSAList.EQUALS,"SURG");
ssaList.addInitialQualification("PATIENT","PATNAME",SSAList.EQUALS,"ANDREA SMITH");

Path path = ssaList.getPathForRetrieveReplace();
if(pcb.getUnique(path, ssaList, true)){
   path.setString("PATNAME", "ANDREA TAYLOR");
   pcb.replace(path);
}
while(pcb.getNext(path, ssaList, true){
   path.setString("PATNAME", "ANDREA TAYLOR");
   pcb.replace(path);
}
```

注: データベースへの変更を持続的なものとするために、アプリケーションでは PSB を割り振り解除する前に commit メソッドを呼び出す必要があります。そうしないと、変更は最後に commit メソッドが呼び出された時点までロールバックされることになります。

関連概念:

799 ページの『SSAList インターフェースを使用したセグメント検索引数の指定』

IMS Universal DL/I ドライバー・アプリケーションでのバッチ・データ更新の実行

データベースの複数の既存セグメントを 1 回の呼び出しで更新するには、PCB インターフェースで batchUpdate メソッドを使用します。

データベース内の複数の既存セグメントを 1 回の呼び出しで更新するための一般的なステップを以下に示します。

1. データベースを表す PCB インスタンスから SSAList インスタンスを取得します。
2. オプションとして、SSAList メソッドに修飾ステートメントを追加できます。詳しくは、『SSAList インターフェースを使用したセグメント検索引数の指定』を参照してください。
3. 前のステップの SSAList インスタンスを使用して getPathForBatchUpdate メソッドを呼び出し、Path インスタンスを取得します。
4. 前のステップの Path インスタンスを使用して、セグメントを更新するようにフィールド値を設定します。
5. batchUpdate メソッドを呼び出してセグメントを更新します。

以下のコード・フラグメントは、batchUpdate メソッドを使用して患者の名前を更新する方法を示しています。 SSAList インスタンスは、患者名「ANDREA SMITH」、病棟名「SURG」、および病院名「ALEXANDRIA」のレコードだけを更新するように設定されています。 PATIENT セグメントおよびその子セグメントを含むパスを取得するため、getPathForBatchUpdate メソッドが呼び出されます。最後に、患者名フィールドの値を「ANDREA TAYLOR」に変更するため、batchUpdate メソッドが呼び出されます。

```
SSAList ssaList = pcb.getSSAList("HOSPITAL","PATIENT");
ssaList.addInitialQualification("HOSPITAL","HOSPNAME",SSAList.EQUALS,"ALEXANDRIA");
ssaList.addInitialQualification("WARD","WARDNAME",SSAList.EQUALS,"SURG");
ssaList.addInitialQualification("PATIENT","PATNAME",SSAList.EQUALS,"ANDREA SMITH");
Path path = ssaList.getPathForBatchUpdate("PATIENT");
path.setString("PATNAME", "ANDREA TAYLOR");
pcb.batchUpdate(path, ssaList);
```

重要: データベースへの変更を持続的なものとするために、アプリケーションでは PSB を割り振り解除する前に commit メソッドを呼び出す必要があります。そうしないと、変更は最後に commit メソッドが呼び出された時点までロールバックされることになります。

関連概念:

799 ページの『SSAList インターフェースを使用したセグメント検索引数の指定』

IMS Universal DL/I ドライバー・アプリケーションでのデータの削除

データベースの既存のセグメントを削除するには、PCB インターフェースで delete メソッドを使用します。

IMS Universal DL/I ドライバーでは、delete メソッドにより、DL/I DLET 呼び出しと類似の機能が提供されます。削除呼び出しの前に HOLD 操作を実行する必要があります。セグメントを削除すると、セグメントのすべての子セグメントも削除されます。 delete メソッドは、DL/I 操作の結果を示す IMS 状況コードを返します。

データベース内の既存のセグメントを削除するための一般的なステップを以下に示します。

1. データベースを表す PCB インスタンスから非修飾 SSAList インスタンスを取得します。

2. オプションとして、SSAList インスタンスに修飾ステートメントを追加することができます。 詳しくは、『SSAList インターフェースを使用したセグメント検索引数の指定』を参照してください。
3. ステップ 1 および 2 の SSAList インスタンスを使用して getPathForRetrieveReplace メソッドを呼び出し、Path インスタンスを取得します。
4. 置換呼び出しを発行する前に Hold 操作を実行します。 Hold 操作は、getUnique、getNext、または getNextWithinParent メソッド呼び出しのいずれでも構いません。
5. ステップ 3 でリトリーブした Path のすべてのセグメントを削除するか、またはセグメントのサブセットを削除できます。
   - Path のすべてのセグメントを削除するには、引数を指定しないで PCB.delete メソッドを呼び出します。
   - ステップ 3 でリトリーブした Path でデータベースから複数のセグメントが返されており、Path の一部のセグメントだけしか削除したくない場合は、SSAList 引数を指定した PCB.delete メソッドを使用し、削除を開始するセグメントの非修飾 SSAList に渡します。引数として修飾 SSAList を指定すると、例外が throw されます。

IMS Universal DL/I ドライバーでの delete の例

以下のコード・フラグメントは、Path のすべてのセグメントを削除する方法を示しています。引数を指定しないで delete メソッドを呼び出すと、すべての PATIENT セグメントおよびその従属セグメント (ILLNESS、TREATMNT、DOCTOR、BILLING) が削除されます。ここで、patient 名は「ANDREA SMITH」、ward 名は「SURG」、hospital 名は「ALEXANDRIA」、および patient 番号は「PatientNo7」です。

```
SSAList ssaList = pcb.getSSAList("HOSPITAL","ILLNESS");
ssaList.addInitialQualification("HOSPITAL","HOSPNAME",SSAList.EQUALS,"ALEXANDRIA");
ssaList.addInitialQualification("WARD","WARDNAME",SSAList.EQUALS,"SURG");
ssaList.addInitialQualification("PATIENT","PATNAME",SSAList.EQUALS,"ANDREA SMITH");
ssaList.addCommandCode("PATIENT", SSAList.CC_D);
Path path = ssaList.getPathForRetrieveReplace();
if (pcb.getUnique(path, ssaList, true)) {
   if (path.getString("PATIENT", "PATNUM").equals("PatientNo7")) {
      pcb.delete();
   }
}
while (pcb.getNext(path, ssaList, true)) {
   if (path.getString("PATIENT", "PATNUM").equals("PatientNo7")) {
      pcb.delete();
   }
}
```

以下のコード・フラグメントは、非修飾 SSAList を指定して delete を使用する方法を示しています。非修飾 SSAList を指定して delete メソッドを呼び出すと、すべての ILLNESS セグメントおよびその従属セグメント (TREATMNT、DOCTOR) が削除されます。ここで、patient 名は「ANDREA SMITH」、ward 名は「SURGICAL」、hospital 名は「ALEXANDRIA」、および patient 番号は「PatientNo7」です。

```
SSAList ssaList = pcb.getSSAList("HOSPITAL","ILLNESS");
ssaList.addInitialQualification("HOSPITAL","HOSPNAME",SSAList.EQUALS,"ALEXANDRIA");
ssaList.addInitialQualification("WARD","WARDNAME",SSAList.EQUALS,"SURGICAL");
ssaList.addInitialQualification("PATIENT","PATNAME",SSAList.EQUALS,"ANDREA SMITH");
ssaList.markAllFieldsForRetrieval("PATIENT", true);
Path path = ssaList.getPathForRetrieveReplace();
SSAList illnessSSAList = pcb.getSSAList("ILLNESS");
if (pcb.getUnique(path, ssaList, true)) {
   if (path.getString("PATIENT", "PATNUM").equals("PatientNo7")) {
      pcb.delete(illnessSSAList);
   }
}
while (pcb.getNext(path, ssaList, true)) {
   if (path.getString("PATIENT", "PATNUM").equals("PatientNo7")) {
      pcb.delete(illnessSSAList);
   }
}
```

重要: データベースへの変更を持続的なものとするために、アプリケーションでは PSB を割り振り解除する前に commit メソッドを呼び出す必要があります。そうしないと、変更は最後に commit メソッドが呼び出された時点までロールバックされることになります。

関連概念:

799 ページの『SSAList インターフェースを使用したセグメント検索引数の指定』

IMS Universal DL/I ドライバー・アプリケーションでのバッチ・データ削除の実行

データベース内の複数の既存セグメントを 1 回の呼び出しで削除するには、PCB インターフェースで batchDelete メソッドを使用します。

データベース内の複数の既存セグメントを 1 回の呼び出しで削除するための一般的なステップを以下に示します。

1. データベースを表す PCB インスタンスから非修飾 SSAList インスタンスを取得します。
2. オプションとして、SSAList に修飾ステートメントを追加できます。 詳しくは、『SSAList インターフェースを使用したセグメント検索引数の指定』を参照してください。
3. batchDelete メソッドを呼び出して、前のステップで SSAList によって指定したセグメントを削除します。

以下のコード・フラグメントは、batchDelete メソッドを使用して患者のレコードを削除する方法を示しています。 SSAList インスタンスは削除操作を制限するように設定されており、患者名「ANDREA SMITH」、病棟名「SURG」、および病院名「ALEXANDRIA」のレコードだけを削除します。

```
SSAList ssaList = pcb.getSSAList("HOSPITAL","PATIENT");
ssaList.addInitialQualification("HOSPITAL","HOSPNAME",SSAList.EQUALS,"ALEXANDRIA");
ssaList.addInitialQualification("WARD","WARDNAME",SSAList.EQUALS,"SURG");
ssaList.addInitialQualification("PATIENT","PATNAME",SSAList.EQUALS,"ANDREA SMITH");
pcb.batchDelete(ssaList);
```

重要: データベースへの変更を持続的なものとするために、アプリケーションでは PSB を割り振り解除する前に PSB.commit メソッドを呼び出す必要があります。そうしないと、変更は最後に commit が呼び出された時点までロールバックされることになります。

関連概念:

799 ページの『SSAList インターフェースを使用したセグメント検索引数の指定』

com.ibm.ims.dli.AIB インターフェースを使用した PCB 状況コードおよび関連情報の検査

IMS Universal ドライバーによるデータ・アクセス呼び出しの後で、PCB 状況コード、戻りコード、理由コード、エラー・コード拡張、および関連情報を検査するには、IMS Universal DL/I ドライバーによって提供される com.ibm.ims.dli.AIB インターフェースを使用します。

通常、呼び出しが正常に実行されない場合には、IMS Universal ドライバーは例外をスローします。例外を生成しない非ブランク状況コードは、
GD、GE、GB、GA、GK、QC、QD、および CF です。エラーの場合、生成される例外には、PCB 状況コード、戻りコード、理由コード、およびエラー・コード拡張などの、最も適切な情報が含まれます。エラー以外のすべてのケースについて、PCB 状況コードおよび関連情報を検査するには、com.ibm.ims.dli.AIB インターフェースを使用します。

AIB インスタンスには、IMS アプリケーション・インターフェース・ブロックのすべてのデータ属性が含まれます。また、AIB インスタンスには、PCBインスタンスのすべてのデータ属性を含む com.ibm.ims.dli.DBPCB インスタンスへの参照も含まれます。

IMS Universal ドライバー・アプリケーションで IMS へのデータ・アクセス呼び出しを行うと、アプリケーションは com.ibm.ims.dli.PCB インスタンスを使用して内部的に DL/I 呼び出しを行います。アプリケーションが発行する各 DL/I 呼び出しの後に、IMS はその PCB インスタンス用の AIB インスタンスに保管されている DBPCB インスタンスの中に、2 文字の状況コードを入れます。

com.ibm.ims.dli.IMSStatusCodes クラスには、IMS 状況コード用の定数が入ります。このヘルパー・クラスを使用して、DL/I 呼び出しからの状況コードの比較検査を行います。

以下のサンプル・コードは、IMS Universal DL/I ドライバー・アプリケーションから AIB インスタンスにアクセスする方法を示しています。

```
try {
   psb = PSBFactory.createPSB(connSpec);
} catch (DLIException e) {
   AIB aib = e.getAib();
   if (aib != null) {
      String sc = aib.getDBPCB().getStatusCodeChars();
      String retcode = aib.getReturnCodeHex();
      String reascode = aib.getReasonCodeHex();
```

```
      System.out.println("Status code: " + sc + " Return Code: "
         + retcode + " Reason Code: " + reascode);
      }
}
```

以下のサンプル・コードは、JDBC アプリケーションから AIB インスタンスにアクセスする方法を示しています。 Java コードで AIB および DLIException オブジェクトを使用するには、com.ibm.ims.dli パッケージをインポートする必要があることに注意してください。

```
try {
   resultSet.updateString(1, "Harry Houdini");
} catch (SQLException e) {
   Throwable t = e.getCause();
   if (t != null && t instanceof DLIException) {
      com.ibm.ims.dli.DLIException de = (com.ibm.ims.dli.DLIException) t;
      com.ibm.ims.dli.AIB aib = de.getAib();
      if (aib != null) {
         String sc = aib.getDBPCB().getStatusCodeChars();
         String retcode = aib.getReturnCodeHex();
         String reascode = aib.getReasonCodeHex();
         System.out.println("Status code: " + sc + " Return Code: "
            + retcode + " Reason Code: " + reascode);
      }
    }
}
```

関連資料:

DL/I 状況コードの生命 (メッセージおよびコード)

DL/I トランザクションのコミットまたはロールバック

IMS Universal DL/I ドライバーでは、コミットおよびロールバック・メソッドを使用するローカル・トランザクションのサポートが提供されています。

1 つのローカル・トランザクションは、複数のリカバリー単位を持つ 1 つの作業単位で構成されています。 IMS Universal DL/I ドライバー・アプリケーションは、1 つのリカバリー単位内でデータベースに対する変更をコミットまたはロールバックすることができます。 IMS Universal DL/I ドライバーでは、ローカル・トランザクションは、PSB インスタンスまでを有効範囲とします。ローカル・トランザクションを開始するための明示的呼び出しは必要ありません。作業単位はアプリケーションで PSB オブジェクトを割り振ると開始され、PSB.allocate メソッドを呼び出すことでデータベースへの接続を取得します。

作業単位が開始した後に、アプリケーションは DL/I 呼び出しを行ってデータベースにアクセスし、データの作成、置換、挿入、または削除を実行します。アプリケーションは、PSB.commit メソッドを使用して現在のリカバリー単位をコミットします。コミット操作は、データベースに対して作業単位の開始時点から行われた変更、または最後のコミットあるいはロールバック・メソッド呼び出し時点以後に行われた変更で、いずれか最新の方をコミットするようにデータベースに対して指示します。

重要: データベースへの変更を持続的なものとするために、アプリケーションでは PSB を割り振り解除する前に commit メソッドを呼び出す必要があります。そうしないと、変更は最後に commit メソッドが呼び出された時点までロールバックされることになります。

また、アプリケーションでは PSB.rollback メソッドを使用するロールバック操作を呼び出して、リカバリー単位を終了することもできます。ロールバック操作を呼び出すと、作業単位の開始時点から、または最新のコミットあるいはロールバック呼び出し時点以降の、データベースに対して行われたすべての変更を取り消します。

PSB.commit メソッドまたは PSB.rollback メソッドが呼び出され、PSB インスタンスが割り振り解除されていない場合、新規のリカバリー単位が開始されます。 PSB インスタンスが割り振り解除されると、作業単位全体が終了します。現在は作業単位にない PSB.commit メソッドまたは PSB.rollback メソッドが (PSB インスタンスが割り振られる前または割り振り解除された後に) 呼び出されると、例外がスローされます。

単一の PSB を使用するローカル・トランザクション

以下のサンプル・コードは、単一の PSB 用のローカル・トランザクションを示しています。

```
IMSConnectionSpec connSpec = IMSConnectionSpecFactory.createIMSConnectionSpec();
connSpec.setDatastoreName("IMS1");
connSpec.setDatastoreServer("ecdev123.svl.ibm.com");
connSpec.setPortNumber(5555);
connSpec.setMetadataURL("class://BMP266.BMP266DatabaseView");
connSpec.setUser("usr");
connSpec.setPassword("password");
connSpec.setDriverType(IMSConnectionSpec.DRIVER_TYPE_4);

PSB psb = PSBFactory.createPSB(connSpec);
psb.allocate(); // new unit of work begins
PCB pcb = psb.getPCB("PCb01");

SSAList ssa = pcb.getSSAList("HOSPITAL");
Path path = ssa.getPathForInsert("HOSPITAL");
path.setString("HOSPCODE", "R12100200000A");
path.setString("HOSPNAME", "SANTA TERESA");
pcb.insert(path);
psb.commit();  // or use psb.rollback() to undo the insert.
               // The unit of receovery ends.
```

この例では、アプリケーションはデータベースとの接続を行い、PSB を割り振ります。アプリケーションは PCB を取得して、新規 HOSPITAL レコードを挿入するためのパスを指定します。次にアプリケーションは、DL/I 操作を実行して、新規レコードをデータベースに挿入します。この時点で、アプリケーションは挿入操作をコミットし、新規レコードがデータベースに書き込まれます。または、アプリケーションは挿入操作をロールバックして、データベースを挿入呼び出しが行われたよりも前の状態に戻すことができます。これで現在のリカバリー単位は終了します。

複数の PSB があるローカル・トランザクション

アプリケーションによって複数の PSB オブジェクトが割り振られている場合、PSB ごとに別個のローカル・トランザクションを同時に実行できます。以下のサンプル・コードは、2 つの PSB を持つ複数のローカル・トランザクションの例を示しています。

```
IMSConnectionSpec connSpec = IMSConnectionSpecFactory.createIMSConnectionSpec();
connSpec.setDatastoreName("IMS1");
connSpec.setDatastoreServer("ecdev123.svl.ibm.com");
connSpec.setPortNumber(5555);
```

```
connSpec.setMetadataURL("class://BMP266.BMP266DatabaseView");
connSpec.setUser("usr");
connSpec.setPassword("password");
connSpec.setDriverType(IMSConnectionSpec.DRIVER_TYPE_4);

// create a connection to MyDB
PSB psb = PSBFactory.createPSB(connSpec);
psb.allocate(); // new unit of work begins for psb

// create another connection to MyDB.
// Note: This does not need be be a connection to the same database.
PSB psb2 = PSBFactory.createPSB(connSpec);
psb2.allocate();

pcb = psb.getPCB("PCb01");
SSAList ssa = pcb.getSSAList("HOSPITAL");
Path path = ssa.getPathForInsert("HOSPITAL");
path.setString("HOSPCODE", "R1210020000A");
path.setString("HOSPNAME", "SANTA TERESA");
pcb.insert(path);
psb.commit(); // or use psb.rollback() to undo the insert.
              // The unit of recovery for psb ends

pcb2 = psb2.getPCB("PCb01");
SSAList ssa2 = pcb2.getSSAList("HOSPITAL");
Path path2 = ssa2.getPathForInsert("HOSPITAL");
path2.setString("HOSPCODE", "R1210010000A");
path2.setString("HOSPNAME", "ALEXANDRIA");
pcb2.insert(path2);
psb2.rollback(); //or use psb2.commit() to commit the insert.
                 // The unit of recovery for psb2 ends

psb2.deallocate(); // unit of work ends for psb2
psb.deallocate(); // unit of work ends for psb
```

この例では、アプリケーションは同じデータベースに対して 2 つの接続を行います。 1 つの PSB は最初の接続に対して割り振られ、アプリケーションは DL/I 操作を実行して、病院名「SANTA TERESA」の新規 HOSPITAL レコードをデータベースに挿入します。もう 1 つの PSB は 2 つ目の接続に対して割り振られ、病院名「ALEXANDRIA」の新規 HOSPITAL レコードの挿入操作が実行されます。アプリケーションは次に、最初の PSB の変更をコミットして、病院名「SANTA TERESA」の新規レコードをデータベースに書き込みます。アプリケーションは、2 つ目の PSB に対してはロールバック・ステートメントを発行して、病院名「ALEXANDRIA」のレコードの先行挿入操作を取り消します。病院名「SANTA TERESA」の新規 HOSPITAL レコードの 1 つのみが、データベースに挿入されます。

SSL サポート用の IMS Universal ドライバーの構成

タイプ 4 コネクティビティーを使用して、IMS Universal ドライバーは、Java Secure Socket Extension (JSSE) を介した Secure Sockets Layer (SSL) のサポートを提供します。

この情報はタイプ 4 コネクティビティーのみに適用されます。 Java アプリケーションでの SSL サポートは、IMS Universal Database リソース・アダプターを使用したコンテナー管理環境、または IMS Universal JDBC ドライバーおよび IMS Universal DL/I ドライバーを使用したスタンドアロン環境のいずれかで使用できます。

コンテナー管理環境での SSL サポート用の IMS Universal Database リソース・アダプターの構成

IMS Universal Database リソース・アダプター用に、コンテナー管理環境で SSL を有効にするには、WebSphere Application Server 管理コンソールから SSL 証明書およびキー管理設定を構成する必要があります。

前提条件:

- 最初に IBM z/OS Communications Server Application Transparent Transport Layer Security (AT-TLS) をセットアップして、IMS Connect 用の z/OS システムで SSL サポートを有効にする必要があります。
- また、WebSphere Application Server がインストールされているローカル・ファイル・システムに対するクライアント証明書 (.crt) をリトリーブする必要があります。証明書をリトリーブするには、TSO から、`OMVSADM.CERTAUTH.CERT` メンバーを参照します。コンテンツをローカル・ファイル・システムのテキスト・ファイルにコピーし、末尾のスペースをすべて削除します。ファイルに `hostname.crt` という名前を付けます。

SSL サポート用に IMS Universal Database リソース・アダプターを構成するには、以下のようにします。

1. WebSphere Application Server 管理コンソールを開きます。
2. 左ペインから、「**Security**」->「**SSL certificate and key management**」 を展開します。
3. 「**Key stores and certificates**」をクリックします。
4. 「**NodeDefaultTrustStore**」をクリックします。
5. 「**Signer certificates**」をクリックします。
6. 「**Add**」をクリックします。
7. **Alias** フィールドに、IMS Connect で SSL を有効にするために AT-TLS セットアップ時に作成されたサーバー鍵リング・ファイルに、この証明書が関係がある (抽出された) ことが分かるような名前を入力します。
8. **File name** フィールドには、ローカル・ファイル・システムにある .crt ファイルへの完全修飾パスを入力します。
9. 「**OK**」をクリックして、「**Save**」をクリックします。 トラステッド証明書が自動的に選出され、実行時の SSL ハンドシェーク処理時に使用されます。

関連タスク:

IMS Connect 用の AT-TLS SSL のセットアップ (システム定義)

スタンドアロン環境での SSL サポート用の IMS Universal ドライバーの構成

IMS Universal ドライバー用に、スタンドアロン環境で SSL を有効にするには、SSL 鍵ストアを生成して構成する必要があります。

前提条件:

- 最初に IBM z/OS Communications Server Application Transparent Transport Layer Security (AT-TLS) をセットアップして、IMS Connect 用の z/OS システムで SSL サポートを有効にする必要があります。
- また、ローカル・ファイル・システムに対するクライアント証明書 (.crt) をリトリーブする必要があります。証明書をリトリーブするには、TSO から、OMVSADM.CERTAUTH.CERT メンバーを参照します。コンテンツをローカル・ファイル・システムのテキスト・ファイルにコピーし、末尾のスペースをすべて削除します。ファイルに hostname.crt という名前を付けます。

SSL サポート用に IMS Universal DL/I ドライバーまたは IMS Universal JDBC ドライバーを構成するには、以下のようにします。

1. Java SDK で提供されている Java Keytool を使用して、新しい SSL 鍵ストアを生成します。 この鍵ストア・ファイルは、IMS への SSL 接続を作成する際の SSL ハンドシェーク中に、JRE によりトラストストアとして使用されます。鍵ストア (.ks) ファイルをローカル・ファイル・システムに保管し、そのロケーションを記録しておきます。鍵ストア用のパスワードを設定し、それを記録しておきます。

   鍵ストア・ファイルには、リモート通信ピアから受け取る (証明書の形式の) 公開鍵と、ローカル・システムで生成される公開鍵/秘密鍵のペアを含めることができます。 SSL ハンドシェーク中に使用する通信ピアの証明書をリトリーブするために鍵ストアがアクセスされる場合には、鍵ストア・ファイルはトラストストア として呼ばれます。
2. certificate (.crt) ファイルをトラステッド自己署名証明書として鍵ストアにインポートする前に、証明書が改ざんされていないことを確認してください。Keytool ツールを使用してローカル証明書の指紋を表示し、これをホスト上の鍵リング・ファイルから取り出した元の指紋と比較することで、改ざんの有無を確認できます。
3. 完全修飾パスをシステム・プロパティー javax.net.ssl.trustStore の値として鍵ストア・ファイルに設定し、鍵ストア・パスワードをシステム・プロパティー javax.net.ssl.trustStorePassword の値として設定します。 オプションで、SSL 関連の問題をトラブルシューティングするため、システム・プロパティー javax.net.debug=all を設定して、SSL のクライアント・サイドのトレースをオンにできます。このシステム・プロパティーをコマンド行から指定するには、次のように入力します。

   ```
   java -Djavax.net.debug=all -Djavax.net.ssl.trustStore=myTruststore
   -Djavax.net.ssl.trustStorePassword=myTruststorePassword MyApp
   ```

関連タスク:

IMS Connect 用の AT-TLS SSL のセットアップ (システム定義)

IMS Universal ドライバー・アプリケーションのトレース

IMS Universal ドライバーの問題を診断するデータを取得するため、トレース・データを収集できます。

次のいずれかの手順を使用して、トレースを使用可能にします。

JRE logging.properties ファイルの自動トレースをオンにする

推奨される方式は、Java ランタイム環境 (JRE) の logging.properties ファイルで IMS Universal ドライバー・ロガーのトレース・レベルを設定して、トレースを使用可能にすることです。この方式を使用すると、アプリケーションを再コンパイルする必要がありません。ファイルは、¥jre¥lib¥logging.properties の下の JRE のパスに配置されます。推奨されるトレース・レベルは FINEST です。

すべての IMS Universal ドライバー・ロガーに対してトレースを設定するには、logging.properties ファイルに以下の行を追加します。

```
com.ibm.ims.db.opendb = FINEST
```

トレース出力をファイルに送信するには、logging.properties ファイルを次のように変更します。

1. ファイルに次の行が含まれており、他の handlers= 行が (コメント化されている場合を除いて) 含まれていないことを確認します。行の先頭にあるポンド記号 (#) は、その行がコメントであることを示します。

   ```
   handlers = java.util.logging.FileHandler, java.util.logging.ConsoleHandler
   ```

2. 以下の行を追加します。

   ```
   java.util.logging.FileHandler.level = FINEST
   java.util.logging.FileHandler.pattern = c:/UniversalDriverTrace.txt
   java.util.logging.FileHandler.formatter = java.util.logging.SimpleFormatter
   ```

Java EE トレースの構成

トレースは、Java EE アプリケーション・サーバーからオンにできます。WebSphere Application Server では、これは管理コンソールを使用して構成されます。トレースを構成する前に、WebSphere Application Server に IMS Universal Database リソース・アダプターが配置されている必要があります。

IMS Universal Database リソース・アダプターから最も詳細なトレースを取得するには、以下の手順に従います。

1. WebSphere Application Server 管理コンソールを開始します。
2. 「**Troubleshooting**」を選択します。
3. 「**Logs and Trace**」を選択します。
4. 表から、使用するアプリケーション・サーバーを選択します。
5. 「**General Properties**」の下で、「**Diagnostic Trace**」を選択します。
6. 「**Additional Properties**」の下で、「**Change Log Detail Levels**」を選択します。
7. 「**Runtime**」タブを選択します。
   - 「**Save runtime changes to configuration as well**」チェック・ボックスが「**ON**」になっていることを確認します。
   - 「**Change Log Details Levels**」セクションで、「**com.ibm.ims.db.opendb**」コンポーネントを選択します。これによって「**Message and Trace levels**」メニューが表示されます。
   - メッセージ・レベル「**FINEST**」を選択します。
   - 「**Apply**」をクリックします。

8. アプリケーション・サーバーの次回の始動のためにこれらの変更を保管するには、ページの一番上の「**Save**」リンクをクリックします。 WebSphere Application Server を再始動する必要はありません。

トレースのプログラマチックな使用可能化

IMS Universal ドライバー・アプリケーションでプログラマチックにトレースをオンにすることもできます。これはアプリケーションを再コンパイルする必要があります。

1. アプリケーションで java.util.logging パッケージをインポートし、ストリング引数「com.ibm.ims.db.opendb」を指定して Logger.getLogger メソッドを呼び出すことにより、ロガーを作成します。
2. アプリケーションで、Logger.setLevel メソッドを使用して、ロガーのトレースのレベルを設定できます。推奨されるトレース・レベルは Level.FINEST です。

以下のサンプル・コードは、任意の IMS Universal ドライバー・アプリケーションでプログラマチック・トレースを使用可能にする方法を示しています。

```
private static final Logger universalLogger
   = Logger.getLogger("com.ibm.ims.db.opendb");
universalLogger.setLevel(Level.FINEST);
FileHandler fh
   = new FileHandler("C:/UniversalTrace.txt");
fh.setFormatter(new SimpleFormatter());
fh.setLevel(Level.FINEST);
universalLogger.addHandler(fh);
```

第 41 章 Java 従属領域のプログラミング

続くいくつかのトピックを使用して、Java 従属領域で稼働するアプリケーション・プログラムの設計、作成、および保守を行うことができます。

関連概念:

693 ページの『第 38 章 Java 開発用の IMS ソリューションの概要』

IMS Java 従属領域の概要

IMS Java 従属領域は、Java アプリケーション用の Java 仮想マシン (JVM) 環境を提供する、Java メッセージ処理 (JMP) 領域と Java バッチ処理 (JBP) 領域の、2 つのタイプの IMS 従属領域です。

重要: 以下の z/OS 環境から IMS にアクセスするために、Java アプリケーションはメインフレーム上でホストできます。

- JMP 領域と JBP 領域
- WebSphere Application Server for z/OS
- Db2 for z/OS ストアード・プロシージャー
- CICS

Java アプリケーションを IMS 従属領域で実行する必要がある場合は、Java アプリケーションをホストするために JMP 領域または JBP 領域を使用します。

JMP 領域および JBP 領域では、Java、オブジェクト指向 COBOL、オブジェクト指向 PL/I、またはこれらの言語の組み合わせによって作成されたアプリケーションを実行できます。

JMP アプリケーションおよび JBP アプリケーションから IMS メッセージ・キューにアクセスするには、IMS Java 従属領域リソース・アダプターを使用します。JMP アプリケーションおよび JBP アプリケーションから IMS データベースにアクセスする場合にも、IMS Universal ドライバー (IMS Universal JDBC ドライバーおよび IMS Universal DL/I ドライバー) を使用します。

IMS データベースに加え、Db2 for z/OS 用 JDBC ドライバー (JCC ドライバーのバージョン 3.57.91) を使用することにより、JMP および JBP アプリケーションから Db2 for z/OS データベースにアクセスできます。

JMP 領域および JBP 領域は、31 ビットまたは 64 ビットの Java 仮想マシン (JVM) でロードおよび実行することができます。64 ビットと 31 ビットのアドレッシング・モードを切り替えるには、JVM=64 パラメーターまたは JVM=31 パラメーターを使用します。JVM= に値を指定しなかった場合は、31 ビット・アドレッシング・モードが使用されます。

Java メッセージ処理 (JMP) 領域

JMP 領域は、メッセージ処理プログラム (MPP) 領域と類似していますが、JMP 領域では、Java プログラムのみがスケジュール可能です。 Java プログラムに関連付

けられた PSB ソースでは、オプション LANG=JAVA の指定が必要です。 JMP アプリケーションは、JMP アプリケーションに対するキューにメッセージが存在し、IMS が処理するメッセージをスケジュールに入れると開始されます。 JMP アプリケーションは、MPP アプリケーションのように、端末のユーザーまたは他のアプリケーションからサブミットされたトランザクション・コードを使用して実行されます。各トランザクション・コードは、JMP アプリケーションが処理するトランザクションを表します。

また、単一アプリケーションを複数トランザクション・コードから開始することもできます。JMP アプリケーションは、MPP アプリケーションのように、トランザクション処理方法や出力の送信先の指定において柔軟性があります。また JMP アプリケーションは、任意の出力メッセージをメッセージ・キューに返送し、同じトランザクション・コードを使用する次のメッセージを処理します。このプログラムは、同じトランザクション・コードを使用するメッセージが無くなるまで、継続して実行されます。JMP アプリケーションは、以下のような特性を共用しています。

- 小規模なプログラムである。
- 必要な出力を即時に処理できる。
- DB/DC 環境の IMS または DB2 データ、ならびに DCCTL 環境の DB2 データにアクセスできる。

Java バッチ処理 (JBP) 領域

JBP 領域では、オンラインでバッチ・タイプの処理を実行し、出力用の IMS メッセージ・キューにアクセスできる、非メッセージ・ドリブンのバッチ・メッセージ処理 (BMP) アプリケーションのような柔軟なプログラムを実行できます。 JBP アプリケーションは、JCL によって、あるいは TSO からジョブをサブミットして開始されます。 JBP アプリケーションは、IMS メッセージ・キューから入力メッセージを読み取ることができない点を除き、BMP アプリケーションに類似しています。例えば、始動プロシージャーには IN= パラメーターがありません。 BMP アプリケーションと同様、JBP アプリケーションは、シンボリック・チェックポイント呼び出しおよび再始動呼び出しを使用して、異常終了後にアプリケーションを再始動できます。JBP アプリケーションは、DB/DC または DCCTL 環境の IMS または Db2 for z/OS データ、ならびに DCCTL 環境の Db2 for z/OS データにアクセスできます。

関連概念:

701 ページの『第 40 章 IMS Universal ドライバーを使用したプログラミング』

関連タスク:

858 ページの『IBM Enterprise COBOL for z/OS と JMP および JBP アプリケーションとのインターオペラビリティー』

860 ページの『JMP アプリケーションまたは JBP アプリケーションからの Db2 for z/OS データベースへのアクセス』

IMS Java 従属領域リソース・アダプターを使用したプログラミング

IMS は、IMS Java 従属領域で稼働する Java アプリケーションを開発するための、IMS Java 従属領域リソース・アダプターと呼ばれる一連の Java API を提供しています。

IMS Java 従属領域リソース・アダプターは、JMP または JBP 領域で稼働する Java アプリケーション・プログラムに、メッセージ処理プログラム (MPP) および非メッセージ・ドリブン BMP 領域に提供されているものと同様の、以下のような DL/I 機能を提供できます。

- メッセージの読み取りおよび書き込みのための、IMS メッセージ・キューへのアクセス
- プログラム間通信の実行
- コミットおよびロールバック処理
- GSAM データベースへのアクセス
- データベース・リカバリー (CHKP/XRST)

IMS Java 従属領域リソース・アダプターをタイプ 2 IMS Universal JDBC ドライバーまたはタイプ 2 IMS Universal DL/I ドライバーと一緒に使用して、GSAM データベース・アクセスなどのデータベース操作を実行します。

次の図は、JMP または JBP 領域で稼働する Java アプリケーションを示しています。データベース・アクセス要求およびメッセージ処理要求は IMS Java 従属領域リソース・アダプター および タイプ 2 IMS Universal ドライバー に渡され、それらの呼び出しは DL/I 呼び出しに変換されます。

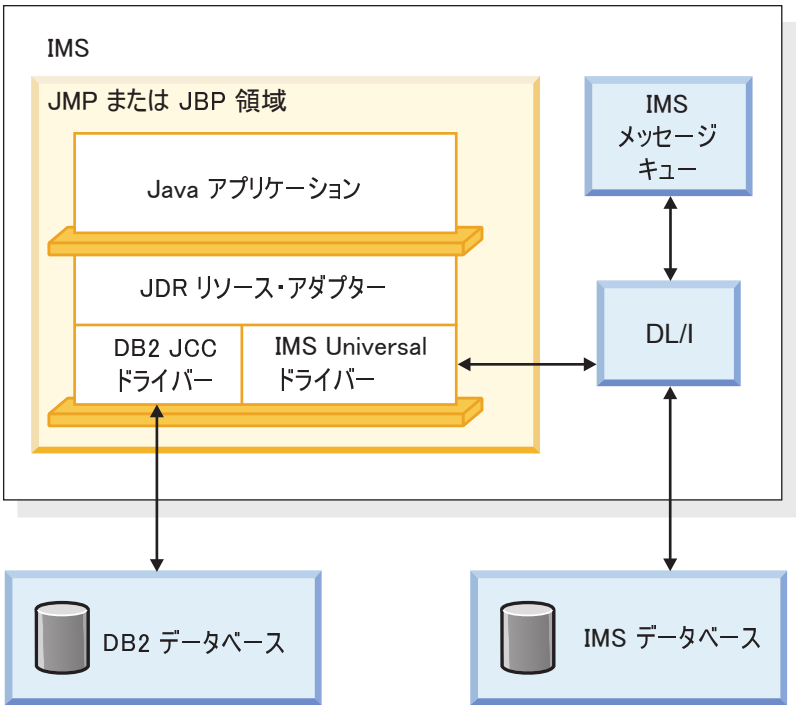


図 *109. IMS Java* 従属領域リソース・アダプターを使用する *JMP* または *JBP* アプリケーション

IMS Java 従属領域リソース・アダプターを使用した Java アプリケーションを作成する準備

IMS Java 従属領域リソース・アダプターは、SMP/E でインストール可能なドライバー (Java オンデマンド機能 FMID を介した `imsutm.jar`) として入手できます。

IMS Java 従属領域リソース・アダプターを使用する Java アプリケーション・プログラムでは、Java Development Kit (JDK) 7.0 以降が必要です。また、IMS Enterprise Suite Explorer for Developmentの使用などの、IMS データベース・メタデータを生成する方法も必要です。 IMS Explorer for Developmentによって作成されるデータベース・メタデータ・クラスのデフォルトのセグメント・エンコードは、cp1047 です。このセグメント・エンコードを変更するには、com.ibm.ims.base.DLIBaseSegment.setDefaultEncoding メソッドを使用します。

関連概念:

115 ページの『第 6 章 メッセージ処理オプションの要件収集』

IMS Java 従属領域リソース・アダプターを使用した JMP アプリケーションの開発

Java メッセージ処理 (JMP) アプリケーションは、IMS メッセージ・キューにアクセスして、メッセージを受信し、処理し、出力メッセージを送信します。

入出力メッセージ・クラスの定義

JMP アプリケーションでメッセージ・キューにアクセスするには、com.ibm.ims.application.IMSFieldMessage クラスをサブクラス化することで、入出力メッセージ・クラスを定義する必要があります。

推奨事項: このサポートが使用可能な場合は、IMS Enterprise Suite Explorer for Developmentを使用して、COBOL サンプル集または PL/I リソースから必要なメタデータ・クラス・ファイルを生成してください。

IMS Java 従属領域リソース・アダプターは、IMSFieldMessage オブジェクトを処理する機能を備えています。

IMSFieldMessage のサブクラス化：入力メッセージのサンプル・コード

このサンプル・コードは、com.ibm.ims.application.IMSFieldMessage クラスをサブクラス化して、メッセージ内のフィールドをプログラムが使用できるようにし、メッセージ内のフィールドの com.ibm.ims.base.DLITypeInfo オブジェクトの配列を作成します。 DLITypeInfo クラスの場合、コードは、最初にフィールド名、続いてデータ・タイプ、位置、最後に配列内の個々のフィールドの長さを識別します。これにより、アプリケーションは IMSFieldMessage クラス階層内でアクセス機能を使用して、データをメッセージ内のそのフォーマットから、アプリケーションが処理できる Java タイプに自動変換できます。ここで定義されるメッセージ固有のフィールドに加えて、IMSFieldMessage クラスには、トランザクション・コードとメッセージ長を判別するのに使用できるアクセス機能があります。

このクラスは、カー・モデルの 2 バイトのタイプ・コードを受信して、入手可能なカー・モデルについてカー・ディーラー・データベースに照会するための、入力メッセージを定義します。

```
package dealership.application;
import com.ibm.ims.db.*;
import com.ibm.ims.base.*;
import com.ibm.ims.application.*;

/* Subclasses IMSFieldMessage to define application's input messages */
public class InputMessage extends IMSFieldMessage {
```

```
    /* Creates array of DLITypeInfo objects for the fields in message */
    final static DLITypeInfo[]fieldInfo={
        new DLITypeInfo("ModelTypeCode", DLITypeInfo.CHAR, 1, 2)
    };

    public InputMessage() {
        super(fieldInfo, 2, false);
    }
}
```

IMSFieldMessage のサブクラス化：出力メッセージのサンプル・コード

次のサンプル・コードは、タイプ・コード照会から入手可能なカー・モデルを表示する出力メッセージを定義するために、com.ibm.ims.application.IMSFieldMessage クラスをサブクラス化する方法を示しています。

このサンプル・コードは、com.ibm.ims.base.DLITypeInfo オブジェクトの配列を作成してから、その配列、バイト配列の長さ、およびブール値 false (非 SPA メッセージを示す) を、IMSFieldMessage コンストラクターに渡します。 DLITypeInfo オブジェクトごとに、最初にフィールドのデータ・タイプ、続いてフィールド名、バイト配列内のフィールドのオフセット、最後にバイト配列の長さを識別する必要があります。

```
package dealership.application;
import com.ibm.ims.db.*;
import com.ibm.ims.base.*;
import com.ibm.ims.application.*;

/*Subclasses IMSFieldMessage to define application's output messages */
public class ModelOutput extends IMSFieldMessage {

s        /* Creates array of DLITypeInfo objects for the fields in message */
    final static DLITypeInfo[] fieldInfo={
        new DLITypeInfo("Type",         DLITypeInfo.CHAR,    1,  2),
        new DLITypeInfo("Make",         DLITypeInfo.CHAR,    3, 10),
        new DLITypeInfo("Model",        DLITypeInfo.CHAR,   13, 10),
        new DLITypeInfo("Year",         DLITypeInfo.DOUBLE, 23,  4),
        new DLITypeInfo("CityMiles",    DLITypeInfo.CHAR,   27,  4),
        new DLITypeInfo("HighwayMiles", DLITypeInfo.CHAR,   31,  4),
        new DLITypeInfo("Horsepower",   DLITypeInfo.CHAR,   35,  4)
    };

    public ModelOutput() {
        super(fieldInfo, 38,false);
    }

}
```

JMP プログラミング・モデル

JMP アプリケーションは、IMS メッセージ・キューからの入力メッセージのリトリーブ、IMS および Db2 for z/OS データベースへのアクセス、トランザクションのコミットまたはロールバック、および出力メッセージの送信を実行できます。

JMP アプリケーションの main メソッドの作成

main メソッド (public static void main(String[] args)) は、すべての JMP および JBP アプリケーションへのプログラム・エントリー・ポイントです。

JMP アプリケーションは、IMS が JMP アプリケーション用のトランザクション・コードを含むメッセージを受信すると開始し、メッセージをスケジュールに入れます。 JMP アプリケーションは通常、処理すべきトランザクション・コードを含むメッセージがなくなると終了します。

JMP アプリケーション main メソッド・コードのサンプル

以下のサンプル・コードは、病院のデータベースにアクセスしてメッセージを送信する JMP アプリケーションの実装方法を示しています。

```
package hospital.ims;

import java.sql.*;
import com.ibm.ims.dli.tm.*;
import com.ibm.ims.dli.DLIException;

public static void main(String args[]) {
  try {
    Application app = null;
    MessageQueue messageQueue = null;
    IOMessage inputMessage  = null;
    IOMessage outputMessage  = null;
    Transaction tran = null;

    app = ApplicationFactory.createApplication();
    inputMessage = app.getIOMessage("class://hospital.ims.InMessage");
    outputMessage = app.getIOMessage("class://hospital.ims.OutMessage");
    messageQueue = app.getMessageQueue();
    tran = app.getTransaction();

    IMSDataSource dataSource = new IMSDataSource();
    dataSource.setMetadataURL("class://hospital.ims.HospitalDBView");
    dataSource.setDriverType(IMSDataSource.DRIVER_TYPE_2);
    dataSource.setDatastoreName("IMS1");

    Connection conn = dataSource.getConnection();
    conn.SetAutoCommit(false);
    Statement st = conn.createStatement();

    String in = new String("");

    // Returns true if a message is read from the queue
    while (messageQueue.getUnique(inputMessage)) {

      in = inputMessage.getString("Message").trim();
      if (!in.equals("")) {

    // Query the database for all hospital names
        ResultSet rs
    = st.executeQuery("SELECT HOSPNAME FROM PCB01.HOSpital");

        while (rs.next()) {

    // Return hospital name in output message
          outputMessage.setString("Message", rs.getString("HOSPNAME"));
          messageQueue.insert(outputMessage,
          MessageQueue.DEFAULT_DESTINATION);

    // Commit this transaction
          tran.commit();
  }
       conn.close();
      }
    }
```

```
  } catch (Exception e) {
     e.printStackTrace();
  }
}
```

JMP アプリケーションからの Db2 for z/OS データへのアクセス

JMP アプリケーションは、IMS データのみにアクセスするときは、データベース接続を一度開くだけで、複数のトランザクションを処理できます。しかし、Db2 for z/OS データにアクセスする JMP アプリケーションは、処理するメッセージごとにデータベース接続の開閉が必要です。

JMP アプリケーションの入力メッセージの処理:

トランザクションは、アプリケーションが入力メッセージを受け取ると開始し、アプリケーションがメッセージ処理の結果をコミットすると終了します。入力メッセージを取得する際、アプリケーションは MessageQueue.getUnique メソッドを呼び出します。

入力メッセージ処理のサンプル・コード

次のサンプル・コードは、JMP アプリケーションにおける入力メッセージの処理方法を示しています。

```
import com.ibm.ims.dli.tm.*;

public static void main(String args[]) {

    conn = dataSource.getConnection(...); //Establish DB connection

    while(messageQueue.getUnique(...)){ //Get input message, which
                                          //starts transaction

        results=statement.executeQuery(...); //Perform DB processing
        ...
        messageQueue.insert(...);     //Send output messages
        ...
    }

    conn.close();                            //Close DB connection

    return;
}
```

JMP アプリケーションにおける IMS 変更のロールバック:

JMP アプリケーションは、トランザクション中に、IMS 変更を何回でもロールバックできます。ロールバックを呼び出すと、すべての出力メッセージが最新のコミットまでバックアウトされます。

JBP アプリケーションでコミットおよびロールバック操作を実行するには、com.ibm.ims.dli.tm.Transaction クラスを使用します。

次のサンプル・コードでは、JMP アプリケーションにおける IMS 変更のロールバック方法を示しています。

```
import com.ibm.ims.dli.tm.*;
import java.sql.*;

public static void main(String args[]) {
```

```
   conn = dataSource.getConnection(...); //Establish DB connection
   Application app = ApplicationFactory.createApplication();
   Transaction tran = app.getTransaction();
   MessageQueue mq = app.getMessageQueue();

   while(mq.getUnique(...)){ //Get input message, which
                 //starts transaction

      results=statement.executeQuery(...); //Perform DB processing
      ...
      mq.insertMessage(...); //Send output messages
      ...
      tran.rollback(); //Roll back output messages

      results=statement.executeQuery(...); //Perform more DB processing
                //(optional)
      ...
      mq.insert(...); //Send more output messages
               //(optional)
   }

   conn.close(); //Close DB connection
}
```

JMP アプリケーションのメッセージ処理に関する追加の考慮事項

ここでの考慮事項は、会話型トランザクション、複数セグメント・メッセージ、繰り返し構造体を持つメッセージ、および複数の入力メッセージを処理する際に、IMS メッセージ・キューにアクセスする JMP アプリケーションに適用されます。

会話型トランザクション:

IMS Java 従属領域リソース・アダプターは、IMS 会話型トランザクションへのアクセスをサポートします。

会話型トランザクション

会話型トランザクションでは、トランザクション全体の処理は一度には行われません。会話型プログラムでは、処理を、端末 - プログラム - 端末と送られる一連の関連した対話に分割します。1 つのトランザクションが複数の部分からなる場合、会話型処理を使用します。これとは対照的に、非会話型プログラムは、端末からメッセージを受信して要求を処理し、メッセージを端末へ返信します。

会話型プログラムでは、端末からメッセージを受信してその端末に応答を返しますが、トランザクションからのデータをスクラッチパッド域 (SPA) に保管します。ユーザーが端末でデータを追加入力する場合に、プログラムは最後のメッセージからのデータを SPA に保持しているので、端末でユーザーが同じデータを再入力しなくても、要求の処理を続行できます。

会話型トランザクションのサンプル

次のサンプル・コードは、会話型トランザクションを処理する JMP アプリケーションの作成方法を示しています。

```
package mytest.jdbo;

import com.ibm.ims.dli.DLIException;
import com.ibm.ims.dli.tm.*;
```

```
public class MyConversationalSample {

  public static void main(String[] args) {
    Transaction tran = null;
    try {
      Application app
         = ApplicationFactory.createApplication();
      IOMessage spaMessage
         = app.getIOMessage("class://mytest.jdbo.SPAMessage");
      IOMessage inputMessage
         = app.getIOMessage("class://mytest.jdbo.InMessage");
      IOMessage outputMessage
         = app.getIOMessage("class://mytest.jdbo.OutMessage");
      MessageQueue msgQueue = app.getMessageQueue();
      tran = app.getTransaction();

      // Read the SPA message
      while (msgQueue.getUnique(spaMessage)) {
        // before reading the application messages.
        if (msgQueue.getNext(inputMessage)) {
          String inField
            = inputMessage.getString("Message").trim();
          try {
            int sum = (new Integer(inField)).intValue();
            spaMessage.setString("Message", "" + sum);
            msgQueue.insert(spaMessage,
              MessageQueue.DEFAULT_DESTINATION);
            outputMessage.setString("Message",
              "The initial value is: " + sum);
            msgQueue.insert(outputMessage,
              MessageQueue.DEFAULT_DESTINATION);
          } catch (NumberFormatException e) {
            if (inField.equalsIgnoreCase("stop")) {
              // End the conversation
              spaMessage.setString("Message",
                "Exit requested, so I am exiting");
spaMessage.setTransactionName(IOMessage.END_CONVERSATION_BLANKS);
msgQueue.insert(spaMessage, MessageQueue.DEFAULT_DESTINATION);
            }
          }
        }
      }
      tran.commit();
    } catch (DLIException e) {
      e.printStackTrace();
      try {
        // Roll back the transaction
        if (tran != null) {
          tran.rollback();
        }
      } catch (DLIException e1) {
        e1.printStackTrace();
      }
    }
   }
}
```

イベントの会話型トランザクション・シーケンス

メッセージが会話型トランザクションである場合、以下の一連のイベントが発生します。

1. IMS は、トランザクション・コードを除去し、それをメッセージ・セグメントの先頭に置きます。メッセージ・セグメントの長さは、システム定義でこのトランザクションに関して定義されている SPA の長さと同じです。トランザクショ

ン・コードは、プログラムが使用できる入力メッセージの最初のセグメントです。端末から取得した 2 から *n* 番目のセグメントから、トランザクション・コードを削除したものが、アプリケーション・プログラムに渡されるメッセージの残りとなります。

2. 会話型プログラムは、応答を準備した後 SPA を IMS に挿入する。次に、プログラムは応答の実際のテキストを、出力メッセージのセグメントとして挿入します。
3. IMS は SPA を保管し、メッセージを入力 LTERM (論理端末) へ経路指定する。
4. 別のプログラムが同じ会話を継続するように SPA 挿入で指定された場合は、すべての応答 (SPA を含む) が次のプログラムへの入力としてメッセージ・キューに保存される。次に、このプログラムは類似の形式のメッセージを受信します。
5. 会話型プログラムを入力交換ごとにスケジューリングする必要がある。その他のプロセスは入力端末のオペレーターが応答を調べて新規入力メッセージを作成している間も、継続することに注意してください。
6. 会話を終了するため、プログラムは SPA のトランザクション・コードのフィールドにブランクを入れ、SPA を IMS に挿入する。 IMS Java 従属領域リソース・アダプターの使用時に会話を終了するには、SPA トランザクション・コードを IOMessage.END_CONVERSATION_BLANKS 定数に設定する。
7. SPA のトランザクション・コードが非会話型プログラムからのトランザクション・コードによって置き換えられ、その SPA が IMS に挿入された場合も、会話は終了することがある。端末での次の入力後に、IMS は、そのメッセージを通常の方法で他方のプログラムのキューに送付します。

関連概念:

会話型トランザクション (「コミュニケーションおよびコネクション」)

複数セグメントのメッセージの処理:

メッセージ・ドリブン・アプリケーションは、複数セグメントの入力メッセージを受け取ることができます。つまり、メッセージ全体をリトリーブするには、メッセージ・キューから複数のメッセージを読み取る必要があります。

以下のサンプル・コードは、複数セグメント・メッセージをリトリーブするための IOMessage および MessageQueue クラスの使用方法を示しています。

```
//Create a message queue
MessageQueue messageQueue = app.getMessageQueue();

//Create the first input message
IOMessage input1
  = app.getIOMessage("class://InputMessage1");

//Create the second input message
IOMessage input2
  = app.getIOMessage("class://InputMessage2");

try {
  //Read the first message from the queue
  messageQueue.getUnique(input1);
  ...
 //Read additional messages from the queue
```

```
   while(messageQueue.getNext(input2)) {
    ...
  } catch (DLIException e) {
    ...
  }
```

繰り返し構造体によるメッセージのコーディングおよびアクセス:

繰り返し構造体のメッセージは、DLITypeInfoList クラスを使用して定義できます。 DLITypeInfoList クラスを使用すると、フィールドの繰り返しリスト、およびそのリストを繰り返すことができる最大回数を指定できます。これらの繰り返し構造体には、繰り返し構造体を含めることができます。

次のサンプル・コードは、Make、Model、および Color の一連の各フィールドがある、サンプルの出力メッセージです。

繰り返し構造体のサンプル出力メッセージ

```
public class ModelOutput extends IMSFieldMessage {
static DLITypeInfo[] modelTypeInfo = {
     new DLITypeInfo("Make", DLITypeInfo.CHAR, 1, 20),
     new DLITypeInfo("Model", DLITypeInfo.CHAR, 21, 20),
     new DLITypeInfo("Color", DLITypeInfo.CHAR, 41, 20),
     };
static DLITypeInfo[] modelTypeInfoList = {
     new DLITypeInfoList("Models", modelTypeInfo, 1, 60, 100),
     };
public ModelOutput() {
   super(modelTypeInfoList, 6004, false);
} }
```

DLITypeInfoList オブジェクト内で定義されているネストされた構造体にアクセスするには、ドット表記法を使用して、繰り返し構造体内にフィールドのフィールドを指定します。例えば、output オブジェクトの 4 番目の「Models」定義の「Color」フィールドは、output メッセージ内の「Models.4.Color」としてアクセスされます。以下のコードは、output メッセージ内の 4 番目の「Color」の値を「Red」に設定します。

```
IOMessage output = app.getIOMessage("class://ModelOutput");
output.setString("Models.4.Color", "Red");
```

複数入力メッセージのフレキシブルな読み取り:

JMP アプリケーションは、さまざまな入力データ・タイプを必要とする複数入力メッセージを処理できます。

例えば以下のカー・ディーラーのサンプル・アプリケーションでは、モデルのリスト作成、モデルの詳細表示、自動車の検索、注文の取り消し、販売の記録といった要求をサポートしています。これらの要求には、それぞれ異なる入力データが必要です。

以下のステップで、メッセージを定義してこれらの要求をサポートする方法、およびアプリケーションからメッセージにアクセスする方法を示します。

1. 基本入力メッセージを定義する。

   基本入力メッセージとは、すべての入力メッセージをリトリーブするために、MessageQueue.getUnique メソッドに渡すメッセージのことです。基本入力メ

ッセージには、アプリケーションが受信する可能性があるすべての入力要求を収めるのに十分な大きさの入出力域がなければなりません。また、すべての入力メッセージに共通のフィールドが、少なくとも 1 つ必要です。この共通フィールドを使用すると、入力要求を判別できます。次のサンプル・コードでは、共通フィールドは CommandCode、各メッセージの最大長は 64 (IMSFieldMessage コンストラクターに渡される数) です。

```
public class InputMessage extends IMSFieldMessage {

    final static DLITypeInfo[] fieldInfo =
    {
       new DLITypeInfo("CommandCode",
          DLITypeInfo.CHAR, 1, 20),
    };

    public InputMessage()
    {
        super(fieldInfo, 64, false);
    }
}
```

2. 各要求ごとに異なる入力メッセージを定義する。

これらの入力メッセージのそれぞれに、その先頭フィールドとして同じ CommandCode フィールドが含まれます。これらの各入力メッセージは、IMSFieldMessage オブジェクトおよび DLITypeInfo 配列を持つ IMSFieldMessage コンストラクターも使用します。 IMSFieldMessage コンストラクターを使用すると、各要求に同じタイプの情報を使用して、基本入力メッセージの内容を再マップできます。したがって、メッセージの入出力域への参照のみで、この領域をコピーする必要はありません。次のサンプル・コードでは、ShowModelDetails、FindACar、および CancelOrder などの要求に対して入力メッセージを作成する方法を示しています。

```
public class ShowModelDetailsInput extends IMSFieldMessage {
   final static DLITypeInfo[] fieldInfo = {
      new DLITypeInfo("CommandCode", DLITypeInfo.CHAR, 1, 20),
      new DLITypeInfo("ModelTypeCode", DLITypeInfo.CHAR, 21, 2)
   };

   public ShowModelDetailsInput(InputMessage inputMessage) {
      super(inputMessage, fieldInfo);
   }
}

public class FindACarInput extends IMSFieldMessage {
   final static DLITypeInfo[] fieldInfo = {
      new DLITypeInfo("CommandCode", DLITypeInfo.CHAR, 1, 20),
      new DLITypeInfo("Make", DLITypeInfo.CHAR, 21, 10),
      new DLITypeInfo("Model", DLITypeInfo.CHAR, 31, 10),
      new DLITypeInfo("Year", DLITypeInfo.CHAR, 41, 4),
      new DLITypeInfo("LowPrice", DLITypeInfo.PACKEDDECIMAL, 45, 5),
      new DLITypeInfo("HighPrice", DLITypeInfo.PACKEDDECIMAL, 50, 5),
      new DLITypeInfo("Color", DLITypeInfo.CHAR, 55, 10),
   };
   public FindACarInput(InputMessage inputMessage) {
      super(inputMessage, fieldInfo);
   }
}

public class CancelOrderInput extends IMSFieldMessage {
   final static DLITypeInfo[] fieldInfo = {
      new DLITypeInfo("CommandCode", DLITypeInfo.CHAR, 1, 20),
      new DLITypeInfo("OrderNumber", DLITypeInfo.CHAR, 21, 6),
```

```
            new DLITypeInfo("DealerNumber", DLITypeInfo.CHAR, 21, 6),
        };
        public CancelOrderInput(InputMessage inputMessage)
        {
            super(inputMessage, fieldInfo);
        }
    }
```

前述のサンプル・コードに関する以下の詳細に注意してください。

- CommandCode フィールドは、コマンド・コードを読み取るすべてのメッセージで定義されます。このフィールドを定義しない場合は、バイト配列内に CommandCode が存在することを考慮して、後続のフィールドのオフセットを調整する必要があります。例えば、CancelOrderInput クラス内の CommandCode の DLITypeInfo 項目は削除できますが、OrderNumber フィールドは依然オフセット 21 から開始する必要があります。
- 基本クラス InputMessage は、そのサブクラスをすべて含めるのに十分な長さである必要があります。この例では、InputMessage クラスは、FindACarInput メソッドのフィールドで必要とされるため、64 バイトです。
- InputMessage の各サブクラスは、InputMessage オブジェクトからそれ自体を作成するためのコンストラクターを備えている必要があります。このコンストラクターは、IMSFieldMessage クラスで、コピー・コンストラクター と呼ばれる新規コンストラクターを使用します。

この設計であれば、アプリケーションは、次のサンプル・コードにあるようなメッセージ読み取りロジックを備えている必要があります。

```
while (messageQueue.getUnique(inputMessage)) {

    string commandCode=inputMessage.getString("CommandCode").trim();

    if (commandCode.equals("ShowModelDetails")) {
        showModelDetails(new ShowModelDetailsInput(inputMessage));

    } else if(commandCode.equals("FindACar")) {
        findACar(new FindACarInput(inputMessage));

    } else {
        //process an error
    }

}
```

IMS Java 従属領域リソース・アダプターを使用した JBP アプリケーションの開発

JBP アプリケーションは、IMS メッセージ・キューから入力メッセージを受信しない点を除き、JMP アプリケーションに類似しています。バッチ・メッセージ処理 (BMP) アプリケーションとは異なり、JBP アプリケーションは、非メッセージ・ドリブン・アプリケーションでなければなりません。

シンボリック・チェックポイントおよび再始動

バッチ・メッセージ処理 (BMP) アプリケーションと同様、JBP アプリケーションは、シンボリック・チェックポイント呼び出しおよび再始動呼び出しを使用して、異常終了後にアプリケーションを再始動できます。 IMS Java 従属領域リソース・

アダプターを使用する際に、シンボリック・チェックポイントおよび再始動を実行するには、以下の com.ibm.ims.dli.tm.Transaction インターフェースのメソッドを使用します。

- Transaction.checkpoint()
- Transaction.restart()

これらのメソッドは、DL/I システム・サービス呼び出しである、(シンボリック) CHKP および XRST に類似の機能を実行します。

JBP アプリケーションは、データベースへの接続、再始動呼び出しの実行、データベース処理の実行、定期的なチェックポイント、およびプログラム終了時のデータベースからの切断を行います。プログラムは、終了前に最終コミットを発行する必要があります。アプリケーションの最初の開始時に、Transaction.restart() メソッドは、アプリケーションに対してシンボリック・チェックポイントと再始動を有効にすることを IMS に通知します。その後、アプリケーションは、定期的に Transaction.checkpoint() 呼び出しを発行して、チェックポイントを取ります。Transaction.checkpoint() メソッドにより、アプリケーションは、チェックポイントを使用して状態を保管する 1 つ以上の他のアプリケーション Java オブジェクトを含む、com.ibm.ims.dli.tm.SaveArea オブジェクトを提供できます。

再始動が必要な場合、最後のチェックポイント ID がデフォルトとなります。Transaction.restart() メソッドは、アプリケーション・オブジェクトがチェックポイント時に挿入されたのと同じ順序で入っている、SaveArea オブジェクトを戻します。 SaveArea オブジェクトが NULL を戻した場合、これは、チェックポイント時に SaveArea オブジェクトに保管されたオブジェクトがなかったことを意味します。

シンボリック・チェックポイント呼び出しおよび再始動呼び出しは、GSAM データ、すなわち z/OS データ・セットで使用されることもあります。基本 z/OS チェックポイントを使用して再始動するには、再始動チェックポイントを特定する必要があります。

シンボリック・チェックポイントおよび再始動の JBP サンプル・コード

シンボリック・チェックポイントおよび再始動についての以下のサンプル JBP アプリケーションは、IMS Java 従属領域リソース・アダプターによるチェックポイントおよび再始動機能サポートの使用例です。

2 つのシンボリック・チェックポイント・メソッド (checkpoint() および checkpoint(SaveArea saveArea) は、(プログラムに何らかの異常終了が発生した場合に) 4 文字定数「LAST」を使用してアプリケーションの再始動を要求します。

```
package samples.dealership.chkp_xrst;

import java.sql.*;
import java.io.*;

import com.ibm.ims.dli.DLIException;
import com.ibm.ims.dli.tm.Application;
import com.ibm.ims.dli.tm.ApplicationFactory;
import com.ibm.ims.dli.tm.SaveArea;
import com.ibm.ims.dli.tm.Transaction;
```

```
import com.ibm.ims.jdbc.IMSDataSource;

public class CheckpointRestartAutoSample {

    private SaveArea saveAreaOut;
    private Connection connection;
    private Transaction transaction;

    // The entry point of the application
    public static void main(String[] args) throws Exception {
        CheckpointRestartAutoSample crSample
          = new CheckpointRestartAutoSample();

        crSample.setup();

        crSample.runSample();

        crSample.closeDown();

    }

    // Set up for the application:
    //  1. Enable trace
    //  2. Creates connection
    void setup() throws Exception {
        this.createConnection();
        Application app = ApplicationFactory.createApplication();
        this.transaction = app.getTransaction();
    }

    void closeDown() throws Exception {
        // close the connection
        connection.close();

        // Commit the IMS DB work
        this.transaction.commit();
    }

    // Creates a connection to the auto dealership database
    void createConnection() throws Exception {
        try {
            IMSDataSource ds = new IMSDataSource();
         ds.setDriverType(IMSDataSource.DRIVER_TYPE_2);
         ds.setMetadataURL("class://samples.dealership.AUTPSB11DatabaseView");

            connection = ds.getConnection();

        } catch (SQLException e) {
            String errorMessage
               = new String("During connection creation: "
                    + e.toString());
            throw new Exception(errorMessage);
        }
    }

    void runSample() throws Exception {
        // the restart call is always the first call in a
        // checkpoint/restart application
        saveAreaOut = this.transaction.restart();

        // if the SaveArea object returned is null it
        // is a normal program start, otherwise it is a restart
        if (saveAreaOut != null) {
            // Check the SaveArea object to determine
               // where to restart from
            if (saveAreaOut.isEmpty()) {
                sqlMethod(true);
```

```
            } else {
                String str =
                   (String)saveAreaOut.getObject(1);
                System.out.println("Retrieved string = "+str);
            }
        } else {
            sqlMethod(false);
        }
    }

    void sqlMethod(boolean isRestart)
       throws DLIException, SQLException {

        String sql
          = new String("SELECT * FROM Dealer.DealerSegment");
        Statement statement = connection.createStatement();
        ResultSet results = statement.executeQuery(sql);

        // this part of the code will be executed only during a normal
        // program start
        if (!isRestart && results.next()) {
            System.out.println("At first GetSegment call to the DealerDB: ");
            System.out.println("Dealer Number = "
               + results.getString("DealerNo"));
            System.out.println("Dealer Name   = "
               + results.getString("DealerName"));
            System.out.println("DealerCity    = "
               + results.getString("DealerCity"));
            System.out.println("DealerZip     = "
               + results.getString("DealerZip"));
            System.out.println("DealerPhone   = "
               + results.getString("DealerPhone"));
        }

        //String ckptid = null;
        for (int i=1; results.next(); i++) {
            System.out.println("GetSegment call to the DealerDB:");
            System.out.println("Dealer Number = "
               + results.getString("DealerNo"));
            System.out.println("Dealer Name   = "
               + results.getString("DealerName"));
            System.out.println("DealerCity    = "
               + results.getString("DealerCity"));
            System.out.println("DealerZip     = "
               + results.getString("DealerZip"));
            System.out.println("DealerPhone   = "
               + results.getString("DealerPhone"));

       // The checkpoint call, apart from storing program information,
       // causes the program to lose its position in the database
            this.transaction.checkpoint();
        }
    }

}
```

JBP アプリケーションにおける変更のロールバック

JMP アプリケーションと同様、JBP アプリケーションは、トランザクション中にデータベース処理および出力メッセージを何回でもロールバックできます。ロールバックを呼び出すと、すべてのデータベース処理と出力メッセージが、最新のコミットにバックアウトされます。

JBP アプリケーションでコミットおよびロールバック操作を実行するには、com.ibm.ims.dli.tm.Transaction クラスを使用します。

関連概念:

81 ページの『プログラムの再始動』

関連資料:

325 ページの『第 17 章 データベースのリカバリーとデータベース保全性の維持』

JBP アプリケーションから GSAM データへのアクセス

GSAM データは、z/OS データ・セットまたはフラット・ファイルと呼ばれることもよくあります。この種のデータは、非階層型の構造になっています。JBP アプリケーションから GSAM データベースのデータにアクセスできます。

JMP アプリケーションは、GSAM データベースに接続し、データベース処理を実行し、定期的にコミットし、プログラム終了時には接続解除を行います。 GSAM データにアクセスするには、JBP アプリケーションにそのデータベース用の Java データベース・メタデータ・クラスを提供する必要があります。

IMS システムにアクティブな IMS カタログ・データベースが組み込まれている場合は、データベース・メタデータ・クラス・ファイルを使用する代わりに、カタログに接続することができます。

カー・ディーラー・データベース用のサンプル・メタデータ・クラス

次の Java サンプル・コードは、Java データベース・メタデータ・クラスの例を示しています。

```
package samples.dealership.gsam;

import com.ibm.ims.db.*;
import com.ibm.ims.base.*;

public class AUTOGSAMDatabaseView extends DLIDatabaseView {

// This class describes the data view of PSB: AUTOGSAM
// PSB AUTOGSAM has database PCBs with 8-char PCBNAME or label:
// AUTOLPCB
// PCBGSAMG
// PCBGSAML

// The following describes Segment:
  // DEALER ("DEALER") in PCB: AUTOLPCB ("AUTOLPCB")
static DLITypeInfo[] AUTOLPCBDEALERArray= {
new DLITypeInfo("DLRNO", DLITypeInfo.CHAR, 1, 4,
  "DLRNO", DLITypeInfo.UNIQUE_KEY),
new DLITypeInfo("DLRNAME", DLITypeInfo.CHAR,
  5, 30, "DLRNAME"),
new DLITypeInfo("CITY", DLITypeInfo.CHAR,
  35, 10, "CITY"),
new DLITypeInfo("ZIP", DLITypeInfo.CHAR,
  45, 10, "ZIP"),
new DLITypeInfo("PHONE", DLITypeInfo.CHAR,
  55, 7, "PHONE")
};
static DLISegment AUTOLPCBDEALERSegment= new DLISegment
("DEALER","DEALER",AUTOLPCBDEALERArray,61);

// The following describes Segment: MODEL ("MODEL")
```

```
  // in PCB: AUTOLPCB ("AUTOLPCB")
static DLITypeInfo[] AUTOLPCBMODELArray= {
new DLITypeInfo("MODKEY", DLITypeInfo.CHAR, 3, 24,
  "MODKEY", DLITypeInfo.UNIQUE_KEY),
new DLITypeInfo("MODTYPE", DLITypeInfo.CHAR, 1, 2, "MODTYPE"),
new DLITypeInfo("MAKE", DLITypeInfo.CHAR, 3, 10, "MAKE"),
new DLITypeInfo("MODEL", DLITypeInfo.CHAR, 13, 10, "MODEL"),
new DLITypeInfo("YEAR", DLITypeInfo.CHAR, 23, 4, "YEAR"),
new DLITypeInfo("MSRP", DLITypeInfo.CHAR, 27, 5, "MSRP"),
new DLITypeInfo("COUNT1", DLITypeInfo.CHAR, 32, 2, "COUNT")
};
static DLISegment AUTOLPCBMODELSegment= new DLISegment
("MODEL","MODEL",AUTOLPCBMODELArray,37);

// An array of DLISegmentInfo objects follows
  // to describe the view for PCB: AUTOLPCB ("AUTOLPCB")
static DLISegmentInfo[] AUTOLPCBarray = {
new DLISegmentInfo(AUTOLPCBDEALERSegment,DLIDatabaseView.ROOT),
new DLISegmentInfo(AUTOLPCBMODELSegment,0),
};

// Warning: PCB: PCBGSAMG has no SENSEGS
// The following describes GSAM Record:
  // JAVGSAM1 ("JAVGSAM1") in PCB: PCBGSAMG ("GSAMRead")
static DLITypeInfo[] PCBGSAMGJAVGSAM1Array= {
new DLITypeInfo("DealerNo", DLITypeInfo.INTEGER, 1, 4),
new DLITypeInfo("DealerName", DLITypeInfo.CHAR, 5, 30),
new DLITypeInfo("ModelType", DLITypeInfo.CHAR, 35, 2),
new DLITypeInfo("ModelKey", DLITypeInfo.CHAR, 37, 24),
new DLITypeInfo("Make", DLITypeInfo.CHAR, 37, 10),
new DLITypeInfo("Model", DLITypeInfo.CHAR, 47, 10),
new DLITypeInfo("Year", "yyyy", DLITypeInfo.DATE, 57, 4),
new DLITypeInfo("MSRP", "S999999V99", DLITypeInfo.PACKEDDECIMAL, 61, 5),
new DLITypeInfo("Counter", DLITypeInfo.SMALLINT, 66, 2)
};
static GSAMRecord PCBGSAMGRecord= new GSAMRecord
("PCBGSAMGRecord",PCBGSAMGJAVGSAM1Array,80);

// An array of DLISegmentInfo objects follows
  // to describe the view for PCB: PCBGSAMG ("GSAMRead")
static DLISegmentInfo[] PCBGSAMGarray = {
new DLISegmentInfo(PCBGSAMGRecord, DLIDatabaseView.ROOT)
};

// Warning: PCB: PCBGSAML has no SENSEGS
// The following describes GSAM Record:
  // JAVGSAM1 ("JAVGSAM1") in PCB: PCBGSAML ("GSAMLoad")
static DLITypeInfo[] PCBGSAMLJAVGSAM1Array= {
new DLITypeInfo("DealerNo", DLITypeInfo.INTEGER, 1, 4),
new DLITypeInfo("DealerName", DLITypeInfo.CHAR, 5, 30),
new DLITypeInfo("ModelType", DLITypeInfo.CHAR, 35, 2),
new DLITypeInfo("ModelKey", DLITypeInfo.CHAR, 37, 24),
new DLITypeInfo("Make", DLITypeInfo.CHAR, 37, 10),
new DLITypeInfo("Model", DLITypeInfo.CHAR, 47, 10),
new DLITypeInfo("Year", "yyyy", DLITypeInfo.DATE, 57, 4),
new DLITypeInfo("MSRP", "S999999V99",
  DLITypeInfo.PACKEDDECIMAL, 61, 5),
new DLITypeInfo("Counter", DLITypeInfo.SMALLINT, 66, 2)
};
static GSAMRecord PCBGSAMLRecord= new GSAMRecord
("PCBGSAMLRecord",PCBGSAMLJAVGSAM1Array,80);

// An array of DLISegmentInfo objects follows
  // to describe the view for PCB: PCBGSAML ("GSAMLoad")
static DLISegmentInfo[] PCBGSAMLarray = {
new DLISegmentInfo(PCBGSAMLRecord, DLIDatabaseView.ROOT)
};
```

```
// Constructor
public AUTOGSAMDatabaseView() {
super("2.0","AUTOGSAM", "AUTOLPCB", "AUTOLPCB",
  AUTOLPCBarray);
addDatabase("GSAMRead", "PCBGSAMG", PCBGSAMGarray);
addDatabase("GSAMLoad", "PCBGSAML", PCBGSAMLarray);
} // end AUTOGSAMDatabaseView constructor

} // end AUTOGSAMDatabaseView class definition
```

GSAM データベースにアクセスする JBP アプリケーションのサンプル

次のサンプル・コードは、GSAM データにアクセスする前述のサンプル・コードに依存する JBP アプリケーションです。

```
package samples.dealership.gsam;

import java.io.*;
import java.util.Properties;
import java.math.BigDecimal;

import com.ibm.ims.dli.*;
import com.ibm.ims.dli.tm.*;

/**
* This is an auto dealership sample application
* demonstrating the use of the
* GSAM database functionality support in
* the IMS Java dependent region resource adapter.
*
*/
public class GSAMAuto {
private final String readOnlyGSAMPCB
  = new String("GSAMRead");
private final String writeOnlyGSAMPCB
  = new String("GSAMLoad");

private PSB psb;

/**
* The entry point of the application
*/
public static void main(String[] args) {
GSAMAuto gsamLoadSample = new GSAMAuto();
if (System.getProperty("com.ibm.ims.jdbcenvironment")
  == null) {
Properties properties = System.getProperties();
properties.put("com.ibm.ims.jdbcenvironment", "IMS");
}

try {
gsamLoadSample.setup();
} catch (Exception e) {
e.printStackTrace();
}

try {
gsamLoadSample.runSample();

gsamLoadSample.closeDown();
} catch (Throwable e) {
e.printStackTrace();
}
```

```
}

/**
* This method does the set up for the application:
* 1. Enable trace
* 2. Creates dbConnection
* 3. Creates GSAMConnection object
* @throws IOException
* @throws SecurityException
* @throws DLIException
*/
void setup() throws SecurityException,
  IOException, DLIException {
IMSConnectionSpec cSpec
  = IMSConnectionSpecFactory.createIMSConnectionSpec();
cSpec.setDatastoreName("IMS1");
cSpec.setDriverType(IMSConnectionSpec.DRIVER_TYPE_2);
cSpec.setMetadataURL("class://samples.dealership.gsam.AUTOGSAMDatabaseView");
psb = PSBFactory.createPSB(cSpec);
}

/**
* This method does the clean up before application exit.
* 1. Commits the database work done. IMS Java dependent
* regions require all applications to commit before exiting.
* @throws DLIException
*
* @exception Exception
*/
void closeDown() throws DLIException {
try {
Application app = ApplicationFactory.createApplication();
Transaction transaction = app.getTransaction();

// Always commit any work before exiting
transaction.commit();
} catch (DLIException e) {
System.out.println("IMS commit failed. Reason: "
  + e.toString());
throw e;
}
}

/**
* Demonstrates how to write to and read from a
* GSAM database. Also shows different data types
* being stored into the GSAM database using the
* internal data conversion methods.
*/
void runSample() {

final int dealerNo = 1171;
final String dealerName = "ABC Autos";
final String modelType = "LX";
final String make = "Santro";
final String model = "Zen";
final java.sql.Date year
  = java.sql.Date.valueOf("2011-05-18");
final BigDecimal msrp = new BigDecimal(17750.00);
final short count = (short) 8;

try {
GSAMPCB pcb1 = psb.getGSAMPCB(this.writeOnlyGSAMPCB);

Path myGSAMRecord = pcb1.getPathForInsert();
```

```
// Set values to individual fields in a GSAM record
myGSAMRecord.setInt("DealerNo", dealerNo);
myGSAMRecord.setString("DealerName", dealerName);
myGSAMRecord.setString("ModelType", modelType);
myGSAMRecord.setString("Make", make);
myGSAMRecord.setString("Model", model);
myGSAMRecord.setDate("Year", year);
myGSAMRecord.setBigDecimal("MSRP", msrp);
myGSAMRecord.setShort("Counter", count);

// Insert the GSAM record data
  // and save the RSA of the record
RSA rsa = pcb1.insert(myGSAMRecord);

// Close the GSAM database explicitly
  // for writing/loading data
pcb1.close();

// Open a GSAM Connection to write the GSAM dataset
GSAMPCB pcb2 = psb.getGSAMPCB(this.readOnlyGSAMPCB);

// Read the GSAM record data using
  // the RSA stored earlier
Path gsamRecord = pcb2.getUnique(rsa);

// Print the GSAM data
if (gsamRecord != null) {
System.out.println("Dealer Number: "
  + gsamRecord.getInt("DealerNo"));
System.out.println("Dealer Name: "
  + gsamRecord.getString("DealerName"));
System.out.println("Model Type: "
  + gsamRecord.getString("ModelType"));
System.out.println("Make: "
  + gsamRecord.getString("Make"));
System.out.println("Model: "
  + gsamRecord.getString("Model"));
System.out.println("Year: "
  + gsamRecord.getDate("Year"));
System.out.println("MSRP: "
  + gsamRecord.getBigDecimal("MSRP"));
System.out.println("Counter: "
  + gsamRecord.getShort("Counter"));

System.out.println
 ("¥nSuccessful completion of GSAM sample application");
} else {
System.out.println("GSAM DB is empty");
}

} catch (DLIException e) {
System.out.println
 ("GSAM sample failed. Reason: " + e.toString());
}
}
}
```

関連概念:

351 ページの『第 20 章 GSAM データベースの処理』

関連資料:

360 ページの『GSAM のコーディングの考慮事項』

Java 従属領域からの同期コールアウト要求の実行

IMS では、Java Message Service (JMS) の IMS 実装を介して、Java メッセージ処理 (JMP) アプリケーションまたは Java バッチ処理 (JBP) アプリケーションからの同期コールアウト機能をサポートします。

同期コールアウトの JMP および JBP サポートを使用するには、IMS Enterprise Suite JMS API jms.jar ファイルがクラスパス上になければなりません。IMS Enterprise Suite JMS API をダウンロードするには、Web URL http://www-01.ibm.com/software/data/ims/enterprise-suite/index.html にアクセスしてください。

JMS の IMS 実装は、Point-to-Point (PTP) メッセージ・ドメインのみのサポートに限定されます。また、Session.AUTO_ACKNOWLEDGE モードの非トランザクション化 QueueSession オブジェクトのみにサポートが提供されます。

JMP または JBP アプリケーションが、IMS でサポートされない JMS メソッドを呼び出そうとしたり、サポートされない引数を JMS メソッド呼び出しに渡そうとしたりすると、JMSException 例外が throw されます。

同期コールアウトの JMP および JBP サポートを使用してメッセージを送信し、同期的に応答を受信する手順は、次のとおりです。

1. com.ibm.ims.jms.IMSQueueConnectionFactory オブジェクトを作成する。
2. IMSQueueConnectionFactory オブジェクトの createQueueConnection メソッドを呼び出して、JMS QueueConnection インスタンスを作成する。
3. QueueConnection インスタンスの createQueueSession メソッドを呼び出して、JMS QueueSession インスタンスを作成する。 このメソッド呼び出しでは、入力パラメーター値を false と Session.AUTO_ACKNOWLEDGE に設定して、生成される QueueSession インスタンスが、非トランザクション化オブジェクトであり、AUTO_ACKNOWLEDGE モードで実行することを指定する必要があります。
4. QueueSession インスタンスの createQueue メソッドを呼び出して、キュー ID を作成する。 メソッド呼び出しで、入力パラメーター値を同期コールアウト操作の OTMA 記述子名に設定する必要があります。
5. JMS QueueRequestor インスタンスを作成し、ステップ 3 で作成した QueueSession インスタンスおよびステップ 4 で作成した Queue インスタンスを、入力パラメーターとして QueueRequestor コンストラクター・メソッドに渡す。
6. ステップ 3 で作成した QueueSession インスタンスの createTextMessage メソッドを呼び出し、TextMessage インスタンスを作成します。 メッセージ・データが含まれる文字列を設定します。
7. メッセージを送信して応答をリトリーブするために、ステップ 5 で作成した QueueRequestor オブジェクトの request メソッドを呼び出す。 このメソッドの呼び出しでは、ステップ 6 で作成した TextMessage インスタンスを渡します。要求メソッド呼び出しの戻り値を、TextMessage インスタンスにキャストする必要があります。呼び出しが正常に実行された場合、戻り値は、同期コールアウト要求に対する応答になります。

以下のコードは、メッセージを外部アプリケーションに送信して同期的に応答メッセージを受信する、単純な JMP (または JBP) アプリケーションの作成方法を示しています。この例では、タイムアウト値を 10 秒、応答メッセージを保持するために割り振られるスペースを 128 KB として IMSQueueConnectionFactory インスタンスを作成します。

```
import javax.jms.JMSException;
import javax.jms.Queue;
import javax.jms.QueueConnection;
import javax.jms.QueueRequestor;
import javax.jms.QueueSession;
import javax.jms.Session ;
import javax.jms.TextMessage;
import com.ibm.ims.jms.IMSQueueConnectionFactory;

public class IMS_Sample
{
    public static void main(String argv[])
    {
        IMSQueueConnectionFactory jmsConnectionFactory
           = new IMSQueueConnectionFactory();
        QueueConnection jmsConnection = null;
        QueueSession jmsQueueSession = null;
        Queue jmsQueue = null;
        QueueRequestor jmsQueueRequestor = null;

        try {
            jmsConnectionFactory.setTimeout(1000);
                        // set the timeout to 10 seconds
            jmsConnectionFactory.setResponseAreaLength(128000);
                        // allocate 128k to hold the response message
            jmsConnection = jmsConnectionFactory.createQueueConnection();
            jmsQueueSession
            = jmsConnection.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
      // set session to be non-transacted and in AUTO_ACKNOWLEDGE mode
            jmsQueue = jmsQueueSession.createQueue("OTMDEST1");
      // pass in the OTMA descriptor name

            jmsQueueRequestor
               = new QueueRequestor(jmsQueueSession, jmsQueue);
            TextMessage sendMsg = jmsQueueSession.createTextMessage();
            sendMsg.setText("MyMessage");
            System.out.println("Sending message: "+sendMsg.getText());
            TextMessage replyMsg
               = (TextMessage)jmsQueueRequestor.request(sendMsg);

            System.out.println("¥nReceived message: "+replyMsg.getText());
        } catch (JMSException e) {
            e.printStackTrace();
        }
    }
}
```

関連タスク:

568 ページの『同期コールアウト機能の実装』

関連資料:

- Java Message Service API Tutorial
- Java Platform Enterprise Edition, v5.0 API Specifications

制御データを含む ICAL コールアウトの IMS Java 従属領域リソース・アダプターによるサポート

Java で作成された、IMS Java 従属領域リソース・アダプターを利用する IMS トランザクションは、制御データを含む ICAL 呼び出しを発行できます。

制御データを使用して、ICAL コールアウト・メッセージにエンドポイント情報やその他の経路指定情報など、任意の情報を組み込むことができます。

制御データを使用した ICAL コールアウトのサポートでは、IMS Java 従属領域リソース・アダプター (imsutm.jar) 内の Java Message Service (JMS) API の IMS 実装が使用されます。IMS Java 従属領域リソース・アダプターは、JNI 呼び出しを介して Universal ドライバーの C ライブラリー (SDFSJLIB データ・セット内の DFSCLIBU) を呼び出し、ICAL 情報を指定して C から AIBTDLI インターフェースに対する呼び出しを発行します。

次の例では、制御データを含む同期コールアウト・メッセージを発行する IMS IMS Java 従属領域リソース・アダプター JMP アプリケーションを示しています。

```
public static void main(String args[]) {
  // Create an IMS JMP application
  app = ApplicationFactory.createApplication();

  // Get the IMS JMS queue connection factory
  IMSQueueConnectionFactory jmsConnectionFactory = app
    .getIMSQueueConnectionFactory();
  QueueConnection jmsConnection = null;
  QueueSession jmsQueueSession = null;
  javax.jms.Queue jmsQueue = null;
  QueueRequestor jmsQueueRequestor = null;

  try {
   // Get a reference to the IMS message queue
   msgQueue = app.getMessageQueue();

   // Create an input message object
   inputMessage = app.getIOMessage("class://MyInputMessage");

   // Create an output message object
   outputMessage = app.getIOMessage("class://MyOutputMessage");

   // Retrieve messages off the queue
   while (msgQueue.getUnique(inputMessage)) {

    // Setting the JMS settings to issue an ICAL call

    // Specify the amount of time to wait for a response from an
    // ICAL call. This value
    // corresponds to the RSFLD value in the AIB
    jmsConnectionFactory.setTimeout(999999);

    // Specify the expected size of the response message from the
    // ICAL call. This value
    // corresponds to the OAUSE value in the AIB
    jmsConnectionFactory.setResponseAreaLength(0x00000033);

    // Create the JMS queue connection
    jmsConnection = jmsConnectionFactory.createQueueConnection();

    // Create the JMS queue session
    jmsQueueSession = jmsConnection.createQueueSession(false, 1);
```

```
// Specify the OTMA Routing descriptor which describes the
// target that the ICAL call
// will be sent to. This value correponse with the RSNM1 value
// in the AIB
jmsQueue = jmsQueueSession.createQueue("DEST0001");

// Create the JMS queue requestor
jmsQueueRequestor = new QueueRequestor(jmsQueueSession,
  jmsQueue);

// Build the request area for the ICAL call
// For synchronous program switch a BytesMessage object must be
// used
BytesMessage sendMsg = jmsQueueSession.createBytesMessage();

// The content of the request area must follow the existing
// format for synchronous program switch:
// LL + ZZ + SWITCH-TO-TRAN + TRAN-INPUT
short ll = 50;
short zz = 0;
sendMsg.writeShort(ll); // Specify the LL value
sendMsg.writeShort(zz); // Specify the ZZ value

// The name of the SWITCH-TO-TRAN is 8 bytes long and encoded in
// CP1047
// This value must be converted to bytes to be written into the
// BytesMessage object
String trancode = new String("SWTCHTRN");
sendMsg.writeBytes(trancode.getBytes("Cp1047"));

// Specify the input data for the switch to transaction
sendMsg.writeUTF(inputMessage.getString("MYINPUT"));

// Build the control data object
IMSControlArea controlArea = new IMSControlArea();
byte controlData[] = "Richard".getBytes();
controlArea.addControlDataItem("name", controlData);

// Attach the control data object to the message object
((BytesMessageImpl) sendMsg).addControlArea(controlArea);

// The length of the request area can be retrieved by calling
// the BytesMessage.getBodyLength() method
// This value corresponds to the OALEN value in the AIB
System.out.println("Request Message Length (AIBOALEN): "
  + sendMsg.getBodyLength());

// Submit the ICAL call
// >>-ICAL--aib--request_area--responseArea---control_area---<<
// For synchronous program switch, the reply message will be a
// BytesMessage object
BytesMessage replyMsg = (BytesMessage) jmsQueueRequestor
  .request(sendMsg);

// The response message will have the following format
// LL + ZZ + TRAN-OUTPUT

// Retrieve the LL field
replyMsg.readShort();

// Retrieve the ZZ field
replyMsg.readShort();

// Retrieve the output data from the switch to transaction and
// place it in the output message
outputMessage.setString("MYOUTPUT", replyMsg.readUTF());
```

```
    // Send the output message back to IMS
    msgQueue
      .insert(outputMessage, MessageQueue.DEFAULT_DESTINATION);

   }

   // Terminate the application and free up any associated resources
   app.end();

  } catch (Exception e) {
   // Error scenario, free up resources
   app.end();
   e.printStackTrace();
  }
 }
```

JMP および JBP アプリケーションでのプログラム間通信

IMS では、JMP および JBP アプリケーションでのプログラム間通信が可能です。JMP および JBP アプリケーションで即時プログラム間通信を行うことができ、また、会話型 JMP アプリケーションにおいて、据え置きプログラム間通信を行うこともできます。

関連概念:

491 ページの『他の IMS アプリケーション・プログラムへのメッセージの送信』

JMP および JBP アプリケーションに対する即時プログラム間通信

IMS Java 従属領域リソース・アダプターは、JMP および JBP アプリケーションの即時プログラム間通信をサポートしています。即時プログラム間通信では、代替 PCB で指定された別の会話型プログラムに会話が直接渡されます。

アプリケーションで即時プログラム間通信を行うと、最初の `MessageQueue.insert` 呼び出しは、SPA を相手方の会話型プログラムに送信しますが、後続の `MessageQueue.insert` 呼び出しは、メッセージを新しいプログラムに送信します。そのプログラムは、元の端末に戻らず、応答もしません。

com.ibm.ims.dli.tm.MessageDestinationSpec クラスの setAlternatePCBName メソッドは、プログラム間通信用の代替 PCB の名前を設定します。setAlternatePCBName メソッドは、DL/I `CHNG` 呼び出しを発行します。

JMP または JBP アプリケーションで即時プログラム間通信を行う手順は、次のとおりです。

1. MessageDestinationSpec.setAlternatePCBName メソッドを呼び出して、代替 PCB の名前を設定する。
2. `MessageQueue.insert` メソッドを呼び出して、メッセージを代替 PCB に送る。

即時プログラム間通信のサンプル・コード

次のサンプル・コードは、JMP アプリケーションでの即時プログラム間通信の実行を示しています。

```
package sample.jmp;

import com.ibm.ims.dli.tm.Application;
import com.ibm.ims.dli.tm.ApplicationFactory;
```

```
import com.ibm.ims.dli.tm.IOMessage;
import com.ibm.ims.dli.tm.MessageDestinationSpec;
import com.ibm.ims.dli.tm.MessageQueue;
import com.ibm.ims.dli.tm.Transaction;

public class SampleJMPImmediatePgmSwitch {
  private static IOMessage outputMessage = null;
  private static MessageQueue msgQueue = null;
  private static Application app = null;
  private static IOMessage inputMessage  = null;

  public static void main(String[] args) {
     try {
        app = (Application) ApplicationFactory.createApplication();
 msgQueue = (MessageQueue) app.getMessageQueue();
 inputMessage
          = app.getIOMessage("class://sample.jmp.InMessage");
 outputMessage
          = app.getIOMessage("class://sample.jmp.OutMessage");

       //Define Message Destinations Specs
 MessageDestinationSpec mds2
          = new MessageDestinationSpec();
 mds2.setAlternatePCBName("TPPCB1");
 mds2.setDestination("JAVTRANJ");

       String in = new String("");
       while (msgQueue.getUnique(inputMessage)){
         in = inputMessage.getString("Message").trim();
   if(in.equalsIgnoreCase("ImmediatePGMSwitch1")){
            outputMessage.setString("Message",
               "Running ImmediatePGMSwitch1 Call");
     msgQueue.insert(outputMessage,
               MessageQueue.DEFAULT_DESTINATION);

       // Insert Message to JAVTRANJ TPPCB1: DLIWithCommit
            outputMessage.setString("Message",
               "Insert Message to JAVTRANJ TPPCB1");
     msgQueue.insert(outputMessage,
               MessageQueue.DEFAULT_DESTINATION);

     outputMessage.setString("Message", "DLIWithCommit");
     outputMessage.setTransactionName("JAVTRANJ");

       // Insert message to JAVTRANJ
      msgQueue.insert(outputMessage, mds2);

       // Commit transaction
            Transaction tran = app.getTransaction();
       tran.commit();
   } else {
            outputMessage.setString("Message",
               "Invalid input - valid input is 'ImmediatePGMSwitch1'");
     msgQueue.insert(outputMessage,
               MessageQueue.DEFAULT_DESTINATION);
     Transaction tran = app.getTransaction();
            tran.commit();
   }
       }
      } catch (Exception e) {
        e.printStackTrace();
      }
   }
}
```

即時プログラム間通信を JBP アプリケーションで実行する場合、その手順は JMP アプリケーションでの手順と似ています。 main モジュールで MessageDestinationSpec インスタンスをセットアップし、続いて別のトランザクションに対して insert 呼び出しを発行します。以下に例を示します。

```
MessageDestinationSpec mds2 = new MessageDestinationSpec();
mds2.setAlternatePCBName("TPPCB1");
mds2.setDestination("JAVTRANJ");
...
outputMessage.setString("Message", "Some Message");
outputMessage.setTransactionName("JAVTRANJ");
msgQueue.insert(outputMessage, mds2);
```

関連概念:

505 ページの『別の会話型プログラムへの会話の引き渡し』

会話型 JMP アプリケーションに対する据え置きプログラム間通信

会話型 JMP アプリケーションにおいて、据え置きプログラム間通信を行うことができます。据え置きプログラム間通信によって、スクラッチパッド域 (SPA) が IMS に返される前に、SPA 内でトランザクション・コードが変更されます。アプリケーションが据え置きプログラム間通信を行う際には、アプリケーションは端末に応答し、別の会話型アプリケーションへ会話を受け渡します。

com.ibm.ims.dli.tm.IOMessage クラスの setTransactionName(String) メソッドを使用し、SPA でトランザクション・コードを指定します。

会話型 JMP アプリケーションで据え置きプログラム間通信を行う手順は、次のとおりです。

1. insert(IOMessage) メソッドを呼び出して、出力メッセージを端末に送る。
2. setTransactionName(String) メソッドを呼び出して、SPA 内でトランザクション・コードの名前を設定する。
3. insert(IOMessage) メソッドを呼び出して、SPA を IMS に送る。

据え置きプログラム間通信のコード・サンプル

次のサンプル・コードは、JMP アプリケーションでの据え置きプログラム間通信の実行を示しています。

```
package sample.jmp;

import com.ibm.ims.dli.tm.Application;
import com.ibm.ims.dli.tm.ApplicationFactory;
import com.ibm.ims.dli.tm.IOMessage;
import com.ibm.ims.dli.tm.MessageDestinationSpec;
import com.ibm.ims.dli.tm.MessageQueue;
import com.ibm.ims.dli.tm.Transaction;


public class SampleJMPDeferredPGM {
   private static IOMessage spaMessage = null;
   private static MessageQueue msgQueue = null;
   private static Application app = null;
   private static IOMessage inputMessage  = null;
   private static Transaction tran = null;

   public static void main(String[] args) {
      try {
         app = ApplicationFactory.createApplication();
```

```
  spaMessage
          = app.getIOMessage("class://sample.jmp.SPAMessage");
  inputMessage
          = app.getIOMessage("class://sample.jmp.InMessage");
  msgQueue = app.getMessageQueue();
  tran = app.getTransaction();

  MessageDestinationSpec mds
          = new MessageDestinationSpec();
  mds.setAlternatePCBName("TPPCB1");
  mds.setDestination("IVTCM");

         String in = new String("");
  while (msgQueue.getUnique(spaMessage)) {
            if (msgQueue.getNext(inputMessage)) {
        in = inputMessage.getString("Message").trim();
        if (in.equalsIgnoreCase("DeferredPGMSwitch2")) {
           inputMessage.setString("Message", spaMessage.getString("Message"));
    msgQueue.insert(inputMessage, MessageQueue.DEFAULT_DESTINATION);

     // Setting Deferred Program Switch
    inputMessage.setString("Message", "Setting Deferred Program Switch");
    msgQueue.insert(inputMessage, MessageQueue.DEFAULT_DESTINATION);

    spaMessage.setString("Message", "SampleJMPDeferredPGM");
    spaMessage.setTransactionName("IVTCM");
    msgQueue.insert(spaMessage, mds);

    inputMessage.setString("Message", "SampleJMPDeferredPGM Completed");
    msgQueue.insert(inputMessage, MessageQueue.DEFAULT_DESTINATION);

    tran.commit();
  } else {
    inputMessage.setString("Message", spaMessage.getString("Message"));
    msgQueue.insert(inputMessage, MessageQueue.DEFAULT_DESTINATION);

    inputMessage.setString("Message",
                      "Input Message was not 'DeferredPGMSwitch2'");
    msgQueue.insert(inputMessage, MessageQueue.DEFAULT_DESTINATION);

  tran.commit();
  }
      }
   }
 } catch(Exception e){
    e.printStackTrace();

 }
   }
}
```

関連概念:

505 ページの『別の会話型プログラムへの会話の引き渡し』

Java 従属領域からの同期プログラム間通信要求の実行

IMS では、Java Message Service (JMS) の IMS 実装を介して、Java メッセージ処理 (JMP) アプリケーションまたは Java バッチ処理 (JBP) アプリケーションからの同期プログラム間通信機能をサポートします。

同期プログラム間通信の JMP サポートを使用するには、IMS Enterprise Suite JMS API jms.jar ファイルがクラスパス上になければなりません。IMS Enterprise Suite JMS API をダウンロードするには、URL http://www-01.ibm.com/software/data/ims/enterprise-suite/index.html にアクセスしてください。

この機能を使用するには、タイプ IMSTRAN を使用して OTMA 宛先記述子を作成する必要があります。

制約事項:

- JMS の IMS 実装は、Point-to-Point (PTP) メッセージ・ドメインのみのサポートに限定されます。また、Session.AUTO_ACKNOWLEDGE モードの非トランザクション化 QueueSession オブジェクトのみにサポートが提供されます。
- DFSYIOE0 および DFSMSCE0 出口ルーチンは、同期プログラム間通信要求では呼び出されません。
- アプリケーション・プログラムの開始では、ターゲット・トランザクションは RRS コミットの有効範囲の一部ではありません。
- JBP アプリケーションは、DBCTL 環境では同期プログラム間通信要求を作成することができません。
- ターゲット・トランザクションには、高速機能主記憶データベース (MSDB) への読み取り専用アクセス権があります。
- ターゲット・トランザクションは、会話型トランザクションであってはなりません。
- 同期プログラム間通信要求を共用キュー環境で使用できるのは、参加しているすべての IMS システムの DBRC MINVERS 値が 13.1 以上である場合のみです。

アプリケーションが、IMS によってサポートされていない JMS メソッドの呼び出しを試行した場合、あるいは JMS メソッド呼び出しでサポートされていない引数の受け渡しを試行した場合、API は JMSException をスローします。

以下の手順は、JMP アプリケーションからの同期プログラム間通信操作を実装するためのハイレベル・プログラミングの流れを示しています。

1. IMS JMP アプリケーション・オブジェクトを作成する。
2. IMSQueueConnectionFactory を作成する。 接続ファクトリーおよびアプリケーション・オブジェクトは再使用可能です。
3. IMS メッセージ・キュー参照を取得する。
4. 入出力メッセージ・オブジェクトを作成する。
5. 接続ファクトリーのタイムアウトおよび応答域サイズを構成し、接続を作成する。
6. 接続から JMS キュー・セッションを作成する。
7. IMSTRAN OTMA 宛先記述子の名前を使用してキュー・セッション・オブジェクトを構成し、JMS キュー・リクエスターを作成する。
8. ICAL 呼び出し入力メッセージ領域の BytesMessage オブジェクトを作成する。 同期プログラム間通信要求では、BytesMessage のみがサポートされます。
9. ターゲット・トランザクション・コード、メッセージ長情報、および入力データを指定して、メッセージ・オブジェクトを構成する。
10. ICAL 呼び出しをサブミットする。
11. 出力メッセージをリトリーブし、応答メッセージの長さ情報を解析し、出力メッセージ・データを取得する。
12. 出力メッセージをクリーンアップし、アプリケーションを終了する。

この例は、JMP アプリケーションからの同期プログラム間通信要求を行う方法について示しています。

```
import javax.jms.BytesMessage;
import javax.jms.QueueConnection;
import javax.jms.QueueRequestor;
import javax.jms.QueueSession;
import com.ibm.ims.jms.IMSQueueConnectionFactory;
import com.ibm.ims.dli.tm.Application;
import com.ibm.ims.dli.tm.ApplicationFactory;
import com.ibm.ims.dli.tm.IOMessage;
import com.ibm.ims.dli.tm.MessageQueue;

public class SyncCalloutSample
{
 private static IOMessage inputMessage  = null;
 private static IOMessage outputMessage = null;
 private static MessageQueue msgQueue = null;
 private static Application app = null;

    public static void main(String args[])
    {
     // Create an IMS JMP application
        app = ApplicationFactory.createApplication();

        // Get the IMS JMS queue connection factory
        IMSQueueConnectionFactory jmsConnectionFactory = app.getIMSQueueConnectionFactory();
        QueueConnection jmsConnection = null;
        QueueSession jmsQueueSession = null;
        javax.jms.Queue jmsQueue = null;
        QueueRequestor jmsQueueRequestor = null;

        try {
         // Get a reference to the IMS message queue
         msgQueue = app.getMessageQueue();

         // Create an input message object
         inputMessage = app.getIOMessage("class://MyInputMessage");

         // Create an output message object
         outputMessage = app.getIOMessage("class://MyOutputMessage");

         // Retrieve messages off the queue
         while(msgQueue.getUnique(inputMessage)) {

             // Setting the JMS settings to issue an ICAL call

             // Specify the amount of time to wait for a response
             // from an ICAL call. This value corresponds to the
             // RSFLD value in the AIB
             jmsConnectionFactory.setTimeout(999999);

             // Specify the expected size of the response message
             // from the ICAL call. This value corresponds to the
             // OAUSE value in the AIB
             jmsConnectionFactory.setResponseAreaLength(0x00000033);

             // Create the JMS queue connection
             jmsConnection = jmsConnectionFactory.createQueueConnection();

             // Create the JMS queue session
             jmsQueueSession = jmsConnection.createQueueSession(false, 1);

             // Specify the OTMA Routing descriptor which describes the
             // target that the ICAL call will be sent to. This value
             // correponse with the RSNM1 value in the AIB
             jmsQueue = jmsQueueSession.createQueue("DEST0001");

             // Create the JMS queue requestor
             jmsQueueRequestor = new QueueRequestor(jmsQueueSession, jmsQueue);
```

```
            // Build the request area for the ICAL call
            // For synchronous program switch a BytesMessage object must be used
            BytesMessage sendMsg = jmsQueueSession.createBytesMessage();

            // The content of the request area must follow the existing
            // format for synchronous program switch:
            // LL + ZZ + SWITCH-TO-TRAN + TRAN-INPUT
            short ll = 50;
            short zz = 0;
            sendMsg.writeShort(ll); // Specify the LL value
            sendMsg.writeShort(zz); // Specify the ZZ value

            // The name of the SWITCH-TO-TRAN is 8 bytes long and encoded in CP1047
            // This value must be converted to bytes to be written into the BytesMessage object
            String trancode = new String("SWTCHTRN");
            sendMsg.writeBytes(trancode.getBytes("Cp1047"));

            // Specify the input data for the switch to transaction
            sendMsg.writeUTF(inputMessage.getString("MYINPUT"));

            // The length of the request area can be retrieved by calling the
            // BytesMessage.getBodyLength() method
            // This value corresponds to the OALEN value in the AIB
            System.out.println("Request Message Length (AIBOALEN): " + sendMsg.getBodyLength());

            // Submit the ICAL call
            // For synchronous program switch, the reply message will be a BytesMessage object
            BytesMessage replyMsg = (BytesMessage)jmsQueueRequestor.request(sendMsg);

            // The response message will have the following format
            // LL + ZZ + TRAN-OUTPUT

            // Retrieve the LL field
            replyMsg.readShort();

            // Retrieve the ZZ field
            replyMsg.readShort();

            // Retrieve the output data from the switch to transaction and place it in
            // the output message
            outputMessage.setString("MYOUTPUT", replyMsg.readUTF());

            // Send the output message back to IMS
            msgQueue.insert(outputMessage, MessageQueue.DEFAULT_DESTINATION);

          }

          // Terminate the application and free up any associated resources
          app.end();

        }
        catch(Exception e) {
          // Error scenario, free up resources
          app.end();
              e.printStackTrace();
          }
      }

}
```

この例は、JBP アプリケーションからの同期プログラム間通信要求を行う方法について示しています。

```
package testcases.udb.opendb.t2;

import javax.jms.BytesMessage;
import javax.jms.QueueConnection;
import javax.jms.QueueRequestor;
```

```
import javax.jms.QueueSession;

import com.ibm.ims.jms.IMSQueueConnectionFactory;
import com.ibm.ims.dli.tm.Application;
import com.ibm.ims.dli.tm.ApplicationFactory;

public class SyncPgmSwitchFromJBPSample {
 private static Application app = null;

 public static void main(String args[]) {
  // Create an IMS JBP application
  app = ApplicationFactory.createApplication();

  // Get the IMS JMS queue connection factory
  IMSQueueConnectionFactory jmsConnectionFactory = app.getIMSQueueConnectionFactory();
  QueueConnection jmsConnection = null;
  QueueSession jmsQueueSession = null;
  javax.jms.Queue jmsQueue = null;
  QueueRequestor jmsQueueRequestor = null;

  try {

   // Setting the JMS settings to issue an ICAL call

   // Specify the amount of time to wait for a response from an
   // ICAL call. This value corresponds to the RSFLD value in the AIB
   jmsConnectionFactory.setTimeout(999999);

   // Specify the expected size of the response message from the
   // ICAL call. This value corresponds to the OAUSE value in the AIB
                        int expectedResponseLength = 50;
                        jmsConnectionFactory.setResponseAreaLength(expectedResponseLength);

   // Create the JMS queue connection
   jmsConnection = jmsConnectionFactory.createQueueConnection();

   // Create the JMS queue session
   jmsQueueSession = jmsConnection.createQueueSession(false, 1);

   // Specify the OTMA Routing descriptor which describes the
   // target that the ICAL call will be sent to. This value
   // corresponds with the RSNM1 value in the AIB
   jmsQueue = jmsQueueSession.createQueue("MYDEST");

   // Create the JMS queue requester
   jmsQueueRequestor = new QueueRequestor(jmsQueueSession,
     jmsQueue);

   // Build the request area for the ICAL call
   // For synchronous program switch a BytesMessage object must be used
   BytesMessage sendMsg = jmsQueueSession.createBytesMessage();

   // The content of the request area must follow the existing
   // format for synchronous program switch:
   // LL + ZZ + SWITCH-TO-TRAN + TRAN-INPUT
   short ll = 50;
   short zz = 0;
   sendMsg.writeShort(ll); // Specify the LL value
   sendMsg.writeShort(zz); // Specify the ZZ value

   // The name of the SWITCH-TO-TRAN is 8 bytes long and encoded in CP1047
   // This value must be converted to bytes to be written into the
   // BytesMessage object
   String trancode = new String("SWTCHTRN");
   sendMsg.writeBytes(trancode.getBytes("Cp1047"));

   // Specify the input data for the switch to transaction
```

```
   sendMsg.writeUTF("MYINPUT");

   // The length of the request area can be retrieved by calling
   // the BytesMessage.getBodyLength() method
   // This value corresponds to the OALEN value in the AIB
   System.out.println("Request Message Length (AIBOALEN): "
     + sendMsg.getBodyLength());

   // Submit the ICAL call
   // >>-ICAL--aib--request_area--responseArea------<<
   // For synchronous program switch, the reply message will be a
   // BytesMessage object
   BytesMessage replyMsg = (BytesMessage) jmsQueueRequestor
     .request(sendMsg);

   // The response message will have the following format
   // LL + ZZ + TRAN-OUTPUT

   // Retrieve the LL field
   replyMsg.readShort();

   // Retrieve the ZZ field
   replyMsg.readShort();

   // Retrieve the Data field
   byte[] messageBody = new byte[(int) replyMsg.getBodyLength()];
   replyMsg.readBytes(messageBody);

   // Terminate the application and free up any associated resources
   app.end();

  } catch (Exception e) {
   // Error scenario, free up resources
   app.end();
   e.printStackTrace();
  }
 }

}
```

関連概念:

同期プログラム間通信要求 (コミュニケーションおよびコネクション)

IBM Enterprise COBOL for z/OS と JMP および JBP アプリケーションとのインターオペラビリティー

IBM Enterprise COBOL for z/OS による、COBOL と Java 言語とのインターオペラビリティーのサポートにより、Java 従属領域で実行し、既存の COBOL プログラムを呼び出すことができる、Java アプリケーションおよびオブジェクト指向 (OO) COBOL アプリケーションを作成できます。

このサポートにより、以下を行うことができます。

- メッセージを処理するフロントエンド・アプリケーションを Java で、データベースを処理するバックエンド・アプリケーションを OO COBOL で作成することにより、Java アプリケーションからオブジェクト指向 (OO) COBOL アプリケーションを呼び出す。
- Java ルーチンを呼び出せるメインルーチンを含む OO COBOL アプリケーションを作成する。

JMP または JBP 領域内の COBOL コードにアクセスできます。これは、Enterprise COBOL が、以下を可能にするオブジェクト指向の言語構文を提供するためです。

- COBOL が実装するメソッドおよびデータをクラスに定義する
- Java および COBOL クラスのインスタンスを作成する
- Java および COBOL オブジェクトでメソッドを呼び出す
- Java クラスまたは他の COBOL クラスから継承するクラスを作成する
- 多重定義のメソッドを定義し、呼び出す

IBM Enterprise COBOL for z/OS プログラムでは、JNI が提供するサービスを呼び出して、COBOL 言語で直接取得できる基本的な OO 機能に加え、Java 指向の機能が取得できます。

IBM Enterprise COBOL for z/OS クラスでは、プロシージャー型 COBOL プログラムとのインターフェースをとる CALL ステートメントをコーディングできます。したがって、プロシージャー型 COBOL 論理用にラッパー・クラスを作成して、Java から既存の COBOL コードへのアクセスを可能にする場合、COBOL クラス定義の構文は特に有効なことがあります。

Java コードは、COBOL クラスのインスタンスを作成し、これらのクラスのメソッドを呼び出し、COBOL クラスを拡張できます。

関連資料**:** IBM Enterprise COBOL for z/OS を使用し、IMS 従属領域で稼働するアプリケーションの作成について詳しくは、「*Enterprise COBOL for z/OS* プログラミング・ガイド」を参照してください。

関連概念:

825 ページの『IMS Java 従属領域の概要』

JMP または JBP 領域での IBM Enterprise COBOL for z/OS バックエンド・アプリケーション

オブジェクト指向 (OO) COBOL クラスを定義し、それを IBM Enterprise COBOL for z/OS コンパイラーでコンパイルすると、コンパイラーは、ネイティブ・メソッドとそのネイティブ・メソッドを実装するオブジェクト・コードを持つ Java クラス定義を生成します。クラスのコンパイル後は、インスタンスを作成し、JMP または JBP 領域で実行する Java プログラムから、コンパイルしたクラスのメソッドを呼び出すことができます。

例えば、COBOL で適切な DL/I 呼び出しを用いて OO COBOL クラスを定義して、IMS データベースにアクセスできます。

このクラスの実装を、IMS で稼働する Java アプリケーションで使用可能にする手順は、次のとおりです。

1. COBOL クラスを IBM Enterprise COBOL for z/OS コンパイラーでコンパイルして、Java ソース・ファイルを生成する。これには、クラス定義、およびネイティブ・メソッドの実体が組み込まれたオブジェクト・モジュールが含まれています。

2. 生成された Java ソース・ファイルを Java コンパイラーでコンパイルして、アプリケーション・クラス・ファイルを作成する。
3. オブジェクト・モジュールを、HFS ファイル (.so) 内のダイナミック・リンク・ライブラリー (DLL) にリンクする。
4. JMP または JBP 領域のアプリケーション・クラスパス (ibm.jvm.application.class.path) を更新して、Java クラス・ファイルへのアクセスを可能にする。
5. JMP または JBP 領域のライブラリー・パスを更新して、DLL へのアクセスを可能にする。

JMP または JBP 領域での IBM Enterprise COBOL for z/OS フロントエンド・アプリケーション

IBM Enterprise COBOL for z/OS のオブジェクト指向構文を使用すると、COBOL アプリケーションを、JMP または JBP 領域で直接実行可能な `main` メソッドを用いて作成できます。

JMP または JBP 領域は、Java アプリケーションの `main` メソッドの場合と同じ方法で、この OO COBOL アプリケーションの `main` メソッドの配置、インスタンス化、および呼び出しを行います。

JMP または JBP 領域のアプリケーション全体を OO COBOL で作成できますが、フロントエンドの COBOL アプリケーションには、COBOL アプリケーションから Java ルーチンの呼び出しを使用するほうがより一般的です。

IBM Enterprise COBOL for z/OS ランタイム・サポートは、JMP または JBP 領域の JVM 内で実行される際、この JVM を自動的に探索し、Java クラスでのメソッドの呼び出しに使用します。

JMP または JBP 領域で稼働するメインルーチンを持つフロントエンドの OO COBOL アプリケーションの要件は、JMP または JBP 領域で稼働する Java プログラムと同じです。

JMP アプリケーションまたは JBP アプリケーションからの Db2 for z/OS データベースへのアクセス

JMP または JBP アプリケーションは、Db2 for z/OS 用の JDBC ドライバー (JCC ドライバーのバージョン 3.57.91) を使用して、Db2 for z/OS バージョン 8 および Db2 for z/OS バージョン 9 のデータベースにアクセスできます。

重要: Java と COBOL の両方を同じアプリケーションで使用して Db2 for z/OS データベースにアクセスする場合、予期しない動作が発生することがありますが、これはアクティブ・カーソルが Java の一部にある際にコミットまたはロールバック処理が COBOL で実行された場合のみです。

アプリケーションが稼働している JMP または JBP 領域も、DB2 リカバリー可能リソース・マネージャー・サービス接続機能 (RRSAF) によって接続された Db2 for z/OS で定義することができます。

JMP または JBP アプリケーションから Db2 for z/OS データへのアクセスは、IMS データへのアクセスに似ています。 Db2 for z/OS データにアクセスする JMP または JBP アプリケーションを作成するときは、IMS データベース・アクセスとの相違点と、他の環境での Db2 for z/OS データへのアクセスとの相違点の両方を考慮してください。

- Db2 for z/OS へのアクセスに使用する PSB ごとに (通常は Java アプリケーションごとに) DB2 プランの作成が必要です。
- 任意の時点でオープンにできるアクティブな Db2 for z/OS 接続は 1 つのみです。
- Db2 for z/OS 用のタイプ 2 JDBC ドライバーを使用している場合、アプリケーション・プログラムでデフォルトの接続 URL を使用する必要があります。例えば、`jdbc:db2os390:` または `db2:default:connection` です。
- タイプ 4 DB2 JDBC ドライバーを使用している場合、アプリケーション・プログラムで特定の接続 URL を使用できます。
- 処理をコミットまたはロールバックするには、Transaction.commit メソッドまたは Transaction.rollback メソッドを使用します。
  - JMP アプリケーションの場合、Transaction.commit メソッドは SQL 呼び出しを含むすべての処理をコミットします。Transaction.commit および Transaction.rollback メソッドを呼び出しても、Db2 for z/OS への接続は自動的にリセットされません。Db2 for z/OS への接続は、MessageQueue.getUnique 呼び出しを発行した場合にリセットされます。
  - JBP アプリケーションの場合、Transaction.commit メソッドは SQL 呼び出しをコミットします。
- RRSAF はコーディネーターであるため、Db2 for z/OS 用の JDBC ドライバーの Connection.setAutoCommit または Connection.commit メソッドは使用できません。

関連概念:

825 ページの『IMS Java 従属領域の概要』

関連タスク:

DB2 接続機能を使用するためのシステムの準備 (コミュニケーションおよびコネクション)

DB2: IBM Data Server Driver for JDBC and SQLJ のインストール

第 7 部 IMS Enterprise Suite SOAP Gateway Web サービス用の PL/I トップダウン開発

Web サービス記述言語 (WSDL) ドキュメントから PL/I アプリケーション・テンプレートを生成することができます。このドキュメントは、IBM Developer for System z を使用して、Web サービスの操作とメッセージを記述します。次に、PL/I セグメンテーション API を使用して、生成されたアプリケーションにビジネス・ロジックを追加してから、このアプリケーションを IMS Enterprise Suite SOAP Gateway 上で実行される Web サービスとして使用可能にします。

IBM Developer for System z には、エンタープライズ PL/I で WSDL ファイルから IMS Web サービス・アプリケーション・テンプレートを生成することができるバッチ・プロセッサーがあります。このアプローチは、トップダウン・アプローチと呼ばれており、Web サービスとしての役割を果たすアプリケーションが Web サービス記述ファイルから生成されます。バッチ・プロセッサーは、Web サービスを記述した WSDL ファイルからデータ構造体、XML コンバーター、および PL/I アプリケーション・テンプレート・ファイルを生成します。

第 42 章 生成された PL/I テンプレートにビジネス・ロジックを追加するための WSDL - PL/I 間セグメンテーション API

IBM Developer for System z は、ユーザーが準備した WSDL ファイルと PL/I セグメンテーション API に基づいて生成された成果物の間のハイレベル関係を記録するためのメタデータを生成します。この API のセットを使用して、ビジネス・ロジックを Web サービスとして IMS Enterprise Suite SOAP Gateway で使用可能にする前に追加します。

重要: IBM Developer for System z V9.0.1 以前のバージョンの IRZPWSIO セグメンテーション API は、IMS では DFSPWSIO に名前変更されています。IMS Enterprise Suite V3.1、SOAP Gateway、および IBM Developer for System z V9.0.1.1 以降では、IMS の DFSPWSIO セグメンテーション API を使用する必要があります。

言語構造体は、単一の組み込みファイルに書き込まれています。このファイルの先頭には、操作と言語構造体の関係を示したディクショナリー・コメントが記載されています。メタデータ・ファイルは XML 形式で、バッチ・プロセッサーが XML コンバーター、デプロイメント・メタデータ、およびテンプレート・プログラムを生成するために使用します。生成されたソース・コードにアノテーションが追加され、生成された言語構造体とそれらの派生元の XML スキーマの間の関係を記述します。

アノテーションは、言語構造体またはそれらが適用される言語構造体メンバーの定義の直前に言語コメントとして表示されます。 IBM Developer for System z の WSDL2PLI コンポーネントは、PL/I プログラムのコンパイル時に必要となる DFSPWSH インクルード・ファイルで一連のセグメンテーション API を使用します。これらの API は、IMS メッセージをコンシュームおよび生成する方法を定義します。SDFSSMPL データ・セット内の DFSPWSH インクルード・ファイルは、PL/I バインディングを提供し、データ構造体へのポインターを提供します。

指定されたサービスおよびポートでの各操作ごとに、以下のものが生成されます。

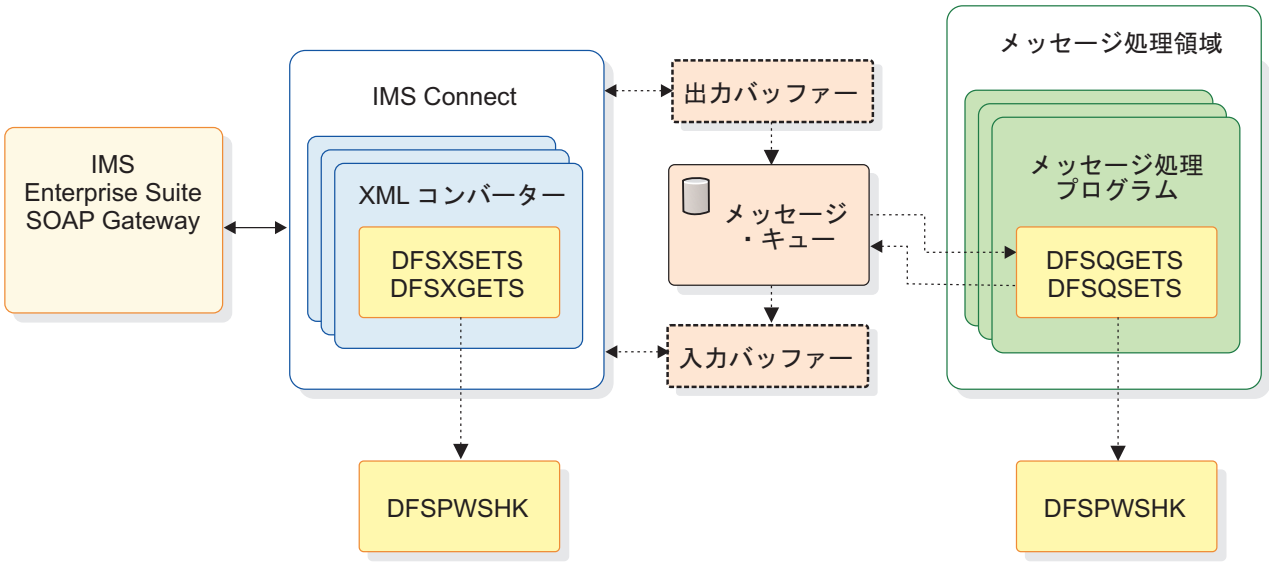
- 操作入力メッセージ用の PL/I 構造体
- 操作入力メッセージ用の XSD から PL/I へのマッピング・セッション
- 操作出力メッセージ用の PL/I から XSD へのマッピング・セッション
- 操作出力メッセージ用の PL/I 構造体

WSDL での各操作ごとに、生成済みのテンプレートで operationNameHandler プロシージャーおよび operationNameImpl プロシージャーが作成されます。operationNameHandler プロシージャーには、プロトコル・ロジックが含まれます。一方、operationNameImpl プロシージャーは、ユーザーのビジネス・ロジックを記入してカスタマイズできるようになっています。

DFSPWSIO セグメンテーション API によって処理されている現行データ構造が入っているバッファーを検査、変更、または置換する方法を例示するために、DFSPWSHK ユーザー出口が用意されています。データ構造が DFSPWSIO API に

よって処理されるときに、データ構造を検査、変更、または置換する必要がある場合は、DFSPWSHK ユーザー出口をカスタマイズしてください。DFSPWSHK ユーザー出口により、パラメーター dfs_in_struct_state および dfs_out_struct_state を使用してデータ構造が変更または置換されたかどうかを追跡することができます。これらのパラメーターと他のパラメーターについては、サンプル・プロシージャーおよび「*IMS V15* 出口ルーチン」で詳しく説明しています。

DFSPWSHK ユーザー出口は、DFSPWSIO セグメンテーション API が IMS Connect の XML コンバーターによって、あるいはメッセージ処理プログラムによって呼び出されると、DFSPWSIO セグメンテーション API によって呼び出されます。したがって、DFSPWSHK 出口をコンパイルし、IMS Connect やメッセージ処理領域が標準の MVS 検索順序 (例えば、STEPLIB および LINKLIST) を使用して検出できるようなデータ・セットにリンクする必要があります。以下の図は、DFSPWSHK 出口が IMS Connect の XML コンバーターとメッセージ処理プログラムの両方によってどのように呼び出されるかを示しています。



- **DFSXSETS**: この API は、IMS Connect の PL/I XML コンバーターが、SOAP ヘッダー、SOAP 本体、または SOAP 障害のいずれかが含まれている言語構造を設定するために使用します。この API は、言語構造を IMS Connect 出力バッファーにコピーするようパラメーター @dfs_commit_structs によって指示されるまで、このコピーを行いません。すべての構造を IMS Connect 出力バッファーにコミット (コピー) するように API に指示する前に、パラメーター @dfs_struct_ptr によって API に渡された構造ポインターを割り振り解除したり、あるいは無効にしたりすると、エラーになります。
- **DFSQGETS**: この API は、メッセージ処理 PL/I プログラムが、CEETDLI インターフェースを使用して IMS メッセージ・キューから言語構造を取得するために使用します。言語構造には、SOAP ヘッダー、SOAP 本体、または SOAP 障害のいずれかが含まれています。DFSQSETS API を使用してその言語構造を設定する前に、すべての言語構造を IMS メッセージ・キューからリトリーブする必要があります。
- **DFSQSETS**: この API は、SOAP ヘッダー、SOAP 本体、または SOAP 障害のいずれかが含まれている言語構造を設定します。この API は、パラメーター

@dfs_commit_structs によって IMS メッセージ・キューに言語構造を挿入するよう指示されるまで、この挿入を行いません。したがって、すべての構造を IMS メッセージ・キューにコミット (挿入) するように API に指示する前に、パラメーター @dfs_struct_ptr により API に渡された構造ポインターを割り振り解除したり、あるいは無効にしたりすると、エラーになります。

- **DFSXGETS**: この API は、IMS Connect の PL/I XML コンバーターが、SOAP ヘッダー、SOAP 本体、または SOAP 障害のいずれかが含まれている言語構造を取得するために使用します。IMS メッセージ・キューは IMS Connect の XML 変換では使用できないため、言語構造は IMS Connect 入力バッファーからリトリーブされます。予想される IMS Connect 入力バッファーの形式は、LLZZDATA バイト・ストリームです。

生成される DFSPWSH 組み込みファイルについて詳しくは、「*IMS V15* アプリケーション・プログラミング *API*」を参照してください。

関連資料:

インクルード・ファイル DFSPWSH (アプリケーション・プログラミング API)

WSDL - PL/I 間セグメンテーション API 出口ルーチン (DFSPWSHK) (出口ルーチン)

第 43 章 生成される PL/I アプリケーション・テンプレートのサンプル

IMS PL/I アプリケーションには、サービス・プロバイダーのメッセージ処理プログラム (MPP) および XML コンバーターによるメッセージ・プロトコルとセグメンテーション API (DFSPWSIO) が必要です。

このサンプルに含まれている FAST247.wsdl で定義された操作の例を示しています。ここで示す操作は、要求を受け取って応答を返すチェック・バランス操作です。 SOAP 障害エレメントも定義されています。

```
<wsdl:operation name="CheckBalanceOperation">
   <soap:operation soapAction="CheckBalanceOperation" style="document" />
  <wsdl:input name="CheckBalanceRequest">
      <soap:body parts="CheckBalancePart" use="literal" />
   </wsdl:input>
   <wsdl:output name="CheckBalanceResponse">
      <soap:body parts="CheckBalancePart" use="literal" />
   </wsdl:output>
   <wsdl:fault name="ServiceExceptionFault">
      <soap:fault use="literal" name="ServiceExceptionFault" />
   </wsdl:fault>
</wsdl:operation>
```

トップダウン・アプリケーション開発ツールは、PL/I アプリケーション・テンプレートで対応する操作を作成します。

```
CheckBalanceOperationImpl: procedure(iopcb_mask_ptr, checkBalanceReques
t_ptr, checkBalanceResponse_ptr, ServiceException_ptr) internal;
   dcl iopcb_mask_ptr pointer byvalue;
   dcl checkBalanceRequest_ptr pointer byvalue;
   dcl checkBalanceResponse_ptr pointer byaddr;
   dcl ServiceException_ptr pointer byaddr;

   return;

end CheckBalanceOperationImpl;
```

第 44 章 WSDL - PL/I 間セグメンテーション API のトレース出力

通常、セグメンテーション API のトレース情報は標準出力に書き込まれるため、メッセージ処理領域のジョブ・ログで見つけることができます。

各 API にはトレース・モードがあり、これが有効になっている場合、メッセージ・ヘッダーからの情報および言語構造体の 2 列 16 進ダンプを書き出します。 IMS Connect Recorder Trace では、IMS メッセージの最初の 670 バイトしか表示されないため、これらのダンプが役に立つ場合があります。

以下は、ソース・コードの例です。

```
01: /* Invoke API DFSQSETS to set the SOAP body language
02:  * structure and commit it to the IMS Message Queue.
03:  */
04: @dfs_struct_name      = 'gettteam_1_0Response';
05: @dfs_struct_ptr       = gettteam_1_0Response_ptr;
06: @dfs_struct_size      = storage(getteam)1_0Response);
07: @dfs_commit_structs   = '1'b;
08: @dfs_cee_feedback_ptr = addr(@dfs_cee_feedback);
09: @dfs_debug            = '1'b;
10:
11: @return_code =
12:     DFSQSETS(@dfs_async_msg_header_ptr,
13:     @dfs_iopcb_mask_ptr, @dfs_soap_body_struct,
14:     @dfs_struct_name, @dfs_struct_ptr,
15:     @dfs_struct_size, @dfs_commit_structs,
16:     @dfs_cee_feedback_ptr, @dfs_debug);
17:
18: if (@return_code != @dfs_success) then do;
19:   display('MYMPP#handle_getteam():
20:   || 'ERROR, DFSQSETS @dfs_soap_body_struct, '
21:   || '@return_code: '|| trim(@return_code) || '.');
22:    return;
23: end;
```

このソース・コードに対応するトレース出力は、以下のとおりです。

```
..:: DFSPWSIO#DFSQSETS() @20140415152643909 ::..
o @dfs_asyn_msg_header_ptr: 877656904.
o @dfs_iopcb_ptr: 110672.
o @dfs_struct_type: 2.
o @dfs_struct_name: getteam_1_0Response.
o @dfs_struct_ptr: 878837800.
o @dfs_struct_size: 150274.
o @dfs_commit_struct: 1.
o @dfs_cee_feedback_ptr: 875679616.
o DFSQSETS#setBodyStruct()
o body_struct_ptr: 878837800.
o body_struct_size: 150274.
o body_struct_ptr(1:body_struct_size):
00000000: 000001F4 00000001 00000002 000DE296 |...4.........So|
00000010: 86A3A681 998540E3 85A2A300 00000000 |ftware Test.....|
00000020: 00000000 00000000 00000000 00000000 |................|
...
```

第 45 章 セグメンテーション API の制限および制約事項

API は、SOAP ヘッダー、本体、および障害構造をサポートするように設計されていますが、現在の API は、SOAP 本体と障害構造のみを実装しています。

第 8 部 IMS Transaction Manager Resource Adapter

IMS Transaction Manager リソース・アダプター(IMS TM リソース・アダプターとも呼ばれます) を使用して、インターネットを介して IMS トランザクションにアクセスするための Java Platform, Enterprise Edition (Java EE (従来の J2EE)) アプリケーションを作成するほか、IMS 従属領域で実行される IMS アプリケーションから外部 Java EE アプリケーションに対するコールアウト要求を作成することができます。

WebSphere 開発環境または Rational 開発環境内でこのリソース・アダプターを使用することにより、以下のことが行えます。

- サービス指向アーキテクチャーをサポートした、ビジネス・プロセスのコンポーネントの開発
- Java Bean からの Java EE アプリケーションの作成
- サービス・ベース・アプリケーションの開発

これで、アプリケーションを、WebSphere Application Server、WebSphere Process Server、WebSphere Transformation Extender、または IBM Integration Bus などのアプリケーション・サーバーにデプロイすることができます。

第 46 章 IMS Transaction Manager Resource Adapter の概要

IBM IMS Transaction Manager リソース・アダプター (IMS TM リソース・アダプターとも呼ばれます) は、Java アプリケーション、Java Platform, Enterprise Edition (Java EE。旧称 J2EE) アプリケーション、または Web サービスにより、ホスト IMS システムで実行されている IMS トランザクションにアクセスするために使用されます。

IMS TM リソース・アダプターは Java EE コネクター・アーキテクチャー (JCA) を実装し、JCA は IMS などのエンタープライズ情報システム (EIS) を Java EE プラットフォームに接続します。JCA により、アプリケーションに、Java EE アプリケーション・サーバーから得られる高品質のサービス (例えば、接続管理、トランザクション管理、セキュリティー管理) が提供されます。さらに、IMS TM リソース・アダプターは、JCA Common Client Interface (CCI) (アプリケーションで IMS Transaction Manager との通信に使用するプログラミング・インターフェース) も実装します。

IMS TM リソース・アダプターは、汎用の Java EE 1.4 以降の任意のアプリケーション・サーバー (例えば、IBM WebSphere Application Server や IBM WebSphere Application Server Liberty Profile) で使用でき、また、Java アプリケーションがホスト IMS システムで実行されている IMS トランザクションにアクセスするときにも使用できます。IMS TM リソース・アダプターを使用すると、IMS アプリケーションが Java EE サーバーでアプリケーションを呼び出すクライアントとしても機能できます。

IMS TM リソース・アダプターは基本的に、トランザクションを IMS にサブミットする Java アプリケーションや Web サービスによって使用されることを目的としていますが、IMS TM リソース・アダプターは、IMS コマンドを IMS にサブミットするサービスによっても使用でき、IMS アプリケーションが外部の Java EE アプリケーションを呼び出す場合にも使用できます。

関連資料:

- IMS TM Resource Adapter V15 リリース・ノート

関連情報:

- J2EE コネクター・アーキテクチャーの仕様
- WebSphere Application Server バージョン 8 の Knowledge Center にある WebSphere Application Server 要求メトリック・ツールの情報

IMS TM リソース・アダプターのコンポーネント

IMS TM リソース・アダプターには、開発時コンポーネントとランタイム・コンポーネントが組み込まれています。

IMS TM リソース・アダプター・ランタイム・コンポーネントは、IBM WebSphere Application Server (z/OS または分散)、IBM WebSphere Integration Developer, IBM IBM Integration Bus、または IBM WebSphere Process Server にデプロイする必要があります。

IMS TM リソース・アダプターには、統合開発環境 (IDE) (例えば、IBM Rational Application Developer for WebSphere Software、IBM Rational Software Architect、WebSphere Integration Developer、IBM WebSphere Transformation Extender、IBM Integration Bus) を使用してアプリケーションを作成できるようにする開発コンポーネントも含まれています。開発コンポーネントは、これらの IDE のオプションの Java EE コネクター・アーキテクチャー (J2C) 機能に組み込まれています。

これらの IDE の J2C ウィザードを使用して、J2C アプリケーション、EJB コンポーネント、および Web サービスを、スタンドアロン・プログラムとして、または既存のアプリケーションへの追加機能として作成できます。また、このウィザードを使用して、特定のバージョンのリソース・アダプター (ご使用のワークスペースにまだデプロイされていない場合) を動的にインポートすることもできます。 Java EE コネクター・アーキテクチャーの Common Client Interface (CCI) を使用して独自のアプリケーションをコーディングすることはできますが、IDE を使用すると開発プロセスが大幅に簡素化されます。

IMS TM リソース・アダプターのランタイム・プロセス

IMS TM リソース・アダプターと IMS の間の通信は、IMS Open Transaction Manager Access (OTMA) と通信する IMS Connect を介して行われます。

開始側クライアント (Java アプリケーションまたは Web サービス) が IMS トランザクションへのアクセス要求を発行すると、IMS TM リソース・アダプターは TCP/IP 接続またはローカル・オプション接続を介して IMS Connect と通信します。IMS Connect はその後、システム間カップリング・ファシリティー (XCF) を使用してトランザクション要求を IMS OTMA に送信し、トランザクションが IMS で実行されます。応答は同じ経路を使用して Java アプリケーションに返されます。

IMS アプリケーションが外部の Enterprise JavaBeans (EJB) コンポーネント、メッセージ駆動型 Bean (MDB)、または Web サービス (IMS コールアウト機能とも呼ばれます) を呼び出すと、IMS アプリケーションからのコールアウト要求が IMS OTMA 保留キューに入れられます。WebSphere Application Server 内の Java アプリケーションは、開始後、IMS TM リソース・アダプターを介して IMS Connect への接続を取得するようにセットアップされています。 IMS TM リソース・アダプター は IMS Connect に対してポーリングを行い、保留キューからコールアウト要求をリトリーブさせます。Java アプリケーションは要求を処理し、通常の IMS トランザクション要求を発行して IMS に何らかの応答データを返します。応答は、IMS アプリケーションが同期、非同期のどちらのコールアウト要求を発行したかに応じて、同じトランザクションまたは異なるトランザクションに返されます。

次の図は、最初のシナリオ (開始側クライアントが IMS トランザクションへのアクセス要求を発行する場合) での、IMS TM リソース・アダプターのランタイム・プ

ロセスを示しています。

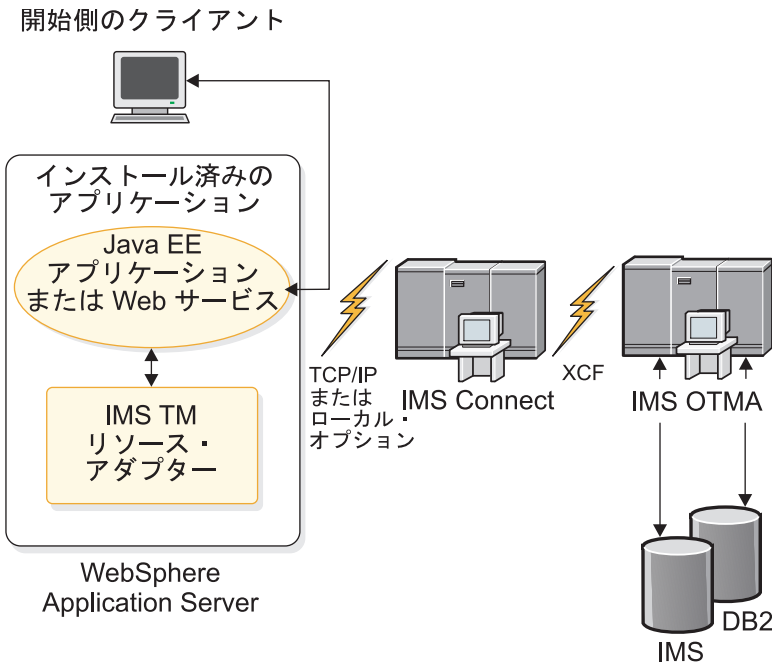


図 *110. IMS TM* リソース・アダプターのランタイム・プロセス

IMS TM リソース・アダプターの機能

IMS TM リソース・アダプターの バージョン 15 には、Java アプリケーションがセキュア・ソケット接続を使用してさまざまなタイプの IMS トランザクションにアクセスするための主要機能が用意されています。

IMS TM リソース・アダプター で使用できる基本機能には、以下のものがあります。

- コンポーネント管理セキュリティーとコンテナー管理セキュリティー
- 接続のプーリングおよび再使用
- コミット・モード 1 とコミット・モード 0 の両方の IMS トランザクション
- IMS TM リソース・アダプターと IMS Connect との間の Secure Sockets Layer (SSL) 通信
- SSL Null 暗号化のサポート
- SSL 鍵ストアおよびトラストストアとしての RACF 鍵リングの使用。
- コミット・モード 0 の対話が失敗したため、または代替プログラム制御ブロック (PCB) への挿入を行うことによる、キューに入れられた出力メッセージのリトリーブ。
- 共用可能永続ソケット接続の使用時における、代替クライアント ID を指定した非同期出力のリトリーブ

- 共用可能永続ソケット接続においてコミット・モード 0 の対話の配信されなかった出力を、キューに入れるか破棄するかの制御。この機能は purgeAsyncOutput プロパティーによって制御されます。
- 共用可能永続ソケット接続におけるコミット・モード 0 の対話の配信されなかった出力の宛先名の指定。この機能は、reRoute フラグおよび reRouteName プロパティーによって制御されます。
- 会話型処理
- コミット・モード 1、同期レベル CONFIRM のアプリケーションのサポート
- PL/I IMS アプリケーションのサポート

  Rational Application Developer (または、Rational Application Developer の必須バージョンを含む、その他の WebSphere および Rational 開発環境) では、IMS TM リソース・アダプターを使用して PL/I IMS アプリケーションを呼び出す Java EE アプリケーションおよび Web サービス・アプリケーションを生成できます。
- グローバル・トランザクションおよび 2 フェーズ・コミットのサポート
- Run-as-thread 識別サポート

注:

- IMS TM リソース・アダプターと IMS Connect の間のローカル・オプション接続のサポートは打ち切られています。代わりに TCP/IP 接続を使用してください。
- MFS SOA のサポートは打ち切られています。現行ユーザーは IBM Rational Host On Demand にマイグレーションする必要があります。

関連資料:

IMS TM Resource Adapter V15 リリース・ノート

IMS TM Resource Adapter バージョン 15 の新機能

V15.1 では、IMS 15 の分散ネットワーク・セキュリティー資格情報に対するサポートが追加されています。

IMS 15 分散ネットワーク・セキュリティー資格情報のサポート

IMS TM Resource Adapter は、インバウンド要求とアウトバウンド (コールアウト) 要求の両方で、IMS ログ・レコード内で監査されるオリジナルの分散ユーザー資格情報を渡すことができます。この機能を使用するには、IMS 15 と IMS Connect 15、およびアプリケーション・サーバー (WebSphere Application Server または WebSphere Liberty) が、ネットワーク・セキュリティー資格情報を使用できるように構成されている必要があります。

拡張可能な Java 認証・承認サービス (JAAS) ログイン・モジュールが IMS TM リソース・アダプターによって提供され、ネットワーク・セキュリティー資格情報を Java EE アプリケーションから IMS に渡せるようになっています。ネットワーク・セキュリティー資格情報を IMS に渡し、IMS ログ・レコード内で監査できるようにするには、事前に JAAS モジュールをインストールし、ご使用のアプリケーションにリンクしておく必要があります。

アクティベーション・スペックが resumeTpipeNsc プロパティーによって拡張され、IMS TM リソース・アダプターが IMS 同期コールアウト・メッセージのネットワーク・セキュリティー資格情報をサポートできるようになりました。 IMS TM リソース・アダプターが非同期コールアウト・メッセージのネットワーク・セキュリティー資格情報をサポートできるように、IMS 相互作用仕様が setResumeTpipeNSC プロパティーによって拡張されています。

分散ネットワーク・セキュリティー資格情報

973 ページの『IMS TM リソース・アダプターのセキュリティー』

990 ページの『分散ネットワーク・セキュリティー資格情報に対するサポートの使用可能化』

991 ページの『分散ネットワーク・セキュリティー資格情報に対応した WebSphere Application Server の構成』

993 ページの『分散ネットワーク・セキュリティー資格情報に対応した WebSphere Liberty の構成』

サポートされるプラットフォーム

IMS TM リソース・アダプターのランタイム・コンポーネントは、各種の分散プラットフォームおよび z/OS 上の WebSphere Application Server をサポートします。

IMS TM リソース・アダプターのランタイム・コンポーネントは、以下のプラットフォーム上の WebSphere Application Server で実行できます。

- z/OS
- Windows
- AIX®
- HP-UX
- Linux
- Linux on System z
- Solaris

その他の汎用 Java EE アプリケーション・サーバーについては、それぞれのサポートされるプラットフォーム用の固有のアプリケーション・サーバーの資料を参照してください。

サポートされるソフトウェア構成

IMS TM リソース・アダプターは、Java EE Connector Architecture (JCA)バージョン 1.5 をベースにしています。

次の表に、サポートされるソフトウェア構成をリストします。

表 *110. IMS TM* リソース・アダプターバージョン *15* によるサポートされるソフトウェア構成

ソフトウェア	サポートされるバージョン
統合 IMS Connect[1]が備わった IMS	• IMS バージョン 15 • IMS バージョン 14 • IMS バージョン 13
IBM WebSphere Application Server IBM WebSphere Liberty サーバー	バージョン 8.5 以降[2]
IBM WebSphere Transformation Extender	バージョン 8.4
IBM Integration Bus	バージョン 8
IBM 以外の Java EE アプリケーション・サーバー	任意の汎用 Java EE 1.7 以降の認定済みアプリケーション・サーバー[2]

1. 統合 IMS Connect の機能は、このバージョンより前および後のバージョンの IMS と共存できます。IMS Connect との共存に関する考慮事項および共存 APAR については、ご使用の IMS バージョンの「リリース計画ガイド」を参照してください。
2. 汎用 Java EE 1.7 以降の認証済みアプリケーション・サーバー (WebSphere Application Server Liberty Profile を含む) は、IMS TM リソース・アダプター インストール検査プログラムが正常に実行できる場合にサポートされます。サポートされるのは、IMS TM リソース・アダプター機能のサブセットのみです。使用の制約事項については、制約事項に関するトピックを参照してください。汎用 Java EE バージョン 1.7 以降の認証済みアプリケーション・サーバーでは、以下の場合、特定の機能に対して IBM サポートが提供されます。
   - WebSphere Application Server または WebSphere Application Server Liberty Profile のサポート対象バージョンで問題を再現できる場合。
   - 診断トレースにより、問題の原因が IMS TM リソース・アダプターであることが確認されている場合。

これらのサポートされるソフトウェア構成における特定の機能のサポートについて詳しくは、要件および制約事項の各トピックを参照してください。

サポートされるアーキテクチャー、開発環境、ランタイム環境 (WebSphere Process Server および IBM Integration Bus の使用を含む) の互換性に関する詳しい説明については、IMS TM リソース・アダプターのサポートされる開発環境およびランタイム環境の技術情報を参照してください。

関連タスク:

898 ページの『インストール検査プログラムを使用したインストールの検査』

関連資料:

883 ページの『IMS TM リソース・アダプターの要件』

『IMS TM リソース・アダプターの制約事項』

関連情報:

サポートされる開発環境とランタイム環境に関する技術情報

IMS TM リソース・アダプターの要件

IMS TM Resource Adapterの一部の機能では、TCP/IP 接続が必要です (ローカル・オプションなし)。

次のリストは、特定のフィーチャーのその他の要件について説明したものです。

- WebSphere Transformation Extender を使用した複雑なデータ形式のサポートには、WebSphere Transformation Extender バージョン 8.2.0.2 以降が必要です。
- IMS Java 従属領域内の IMS アプリケーションから同期コールアウト要求を発行する場合は、IMS Enterprise Suite で Java Message Service (JMS) API をダウンロードしてインストールします。
- ネットワーク・セキュリティー資格情報の IMS TM Resource Adapter サポートには、以下の要件があります。
  - IBM z/OS Connect Enterprise Edition (z/OS Connect EE) の IMS サービス・プロバイダーの場合を除き、以下のアプリケーション・サーバーのいずれか:
    - WebSphere Application Server バージョン 8.0 以降
    - WebSphere Liberty バージョン 8.5.5.9 以降
  - IMS バージョン 15
  - IMS Connect バージョン 15
  - Java 7 以降

各リリース・レベルで使用可能な新機能および機能拡張について詳しくは、IMS TM Resource Adapter Web サイトを参照してください。

関連概念:

879 ページの『IMS TM リソース・アダプターの機能』

関連情報:

IMS Connect 会話型サポート (IMS バージョン 15)

IMS Enterprise Suite JMS API

IMS TM リソース・アダプターの制約事項

IMS TM Resource Adapterには、JCA 1.5 実装環境のサポートに関していくつかの制約事項があります。

- IMS TM Resource Adapter では、JCA 1.5 Kerberos サポートは提供されません。

- IMS 会話型トランザクションを呼び出す複合ビジネス・アプリケーションの作成の詳しい説明および制約については、資料「*IMS* コミュニケーションおよびコネクション」に記載されている IMS Connect 会話型サポートのトピックを参照してください。
- IMS TM Resource Adapter バージョン 10 より後に追加された新機能では、TCP/IP 接続が必要です (ローカル・オプションなし)。
- WebSphere Liberty サーバーを含む汎用 Java EE アプリケーション・サーバーの場合、2 フェーズ・コミット (2PC)、MFS 機能、会話型トランザクション、およびグローバル・トランザクション (XA) はサポートされません。これらの機能は、IMS TM リソース・アダプターが IBM WebSphere Application Serverで実行される場合にのみサポートされます。

  重要**:** この IMS TM リソース・アダプター資料セットに記載されている情報は、主として、WebSphere Application Server を使用した場合に基づいています。WebSphere Application Server Liberty Profile でサポートされない IMS TM リソース・アダプターの機能は、他の汎用 Java EE アプリケーション・サーバーではサポートされません。

関連情報:

IMS Connect 会話型サポート (IMS バージョン 15)

WebSphere Application Server プラットフォーム構成および通信プロトコルに関する考慮事項

IBM WebSphere Application Server と IMS Connect の間の通信プロトコルとしては、TCP/IP を使用してください。

IMS TM リソース・アダプターは、分散プラットフォーム (AIX、HP-UX、Linux、Linux on System z、Solaris、または Windows) 用の WebSphere Application Server、および WebSphere Application Server for z/OS にデプロイできます。IMS TM リソース・アダプターは、TCP/IP 通信プロトコルを使用して IMS Connect と通信します。 .

次の図は、IMS TM リソース・アダプターで可能ないくつかのデプロイメント・シナリオと、サポートされる通信プロトコルを示しています。

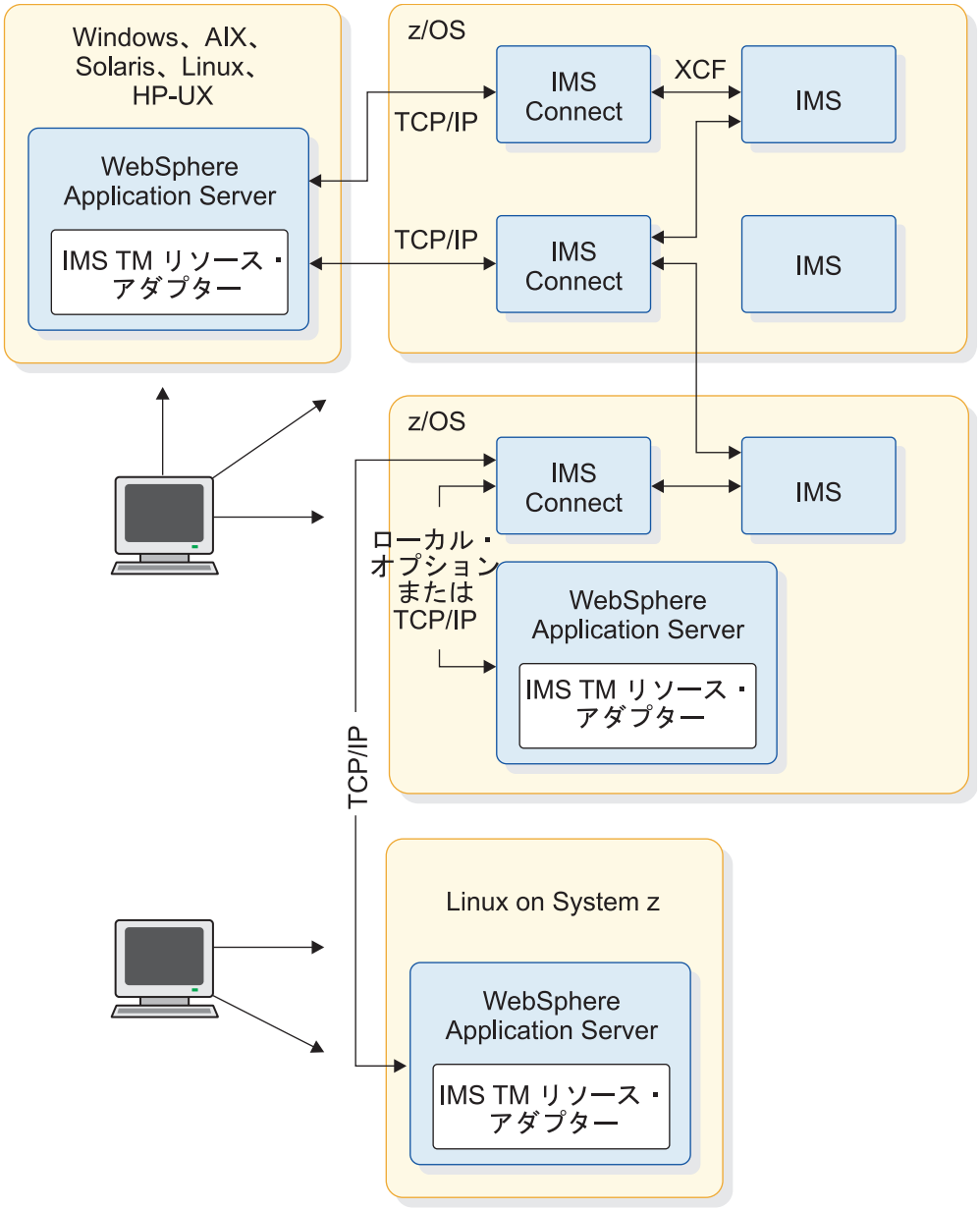


図 *111. IMS TM* リソース・アダプターの接続

次の表は、さまざまなプラットフォーム構成でサポートされる通信プロトコルについて説明しています。すべてのプラットフォーム構成は、グローバル・トランザクション (2 フェーズ・コミット) をサポートします。

IMS TM リソース・アダプターで使用される **WebSphere Application Server** プラットフォーム	サポートされる通信プロトコル
AIX	TCP/IP
HP-UX	TCP/IP
Linux	TCP/IP
Linux on System z	TCP/IP
Solaris	TCP/IP

IMS TM リソース・アダプターで使用される WebSphere Application Server プラットフォーム	サポートされる通信プロトコル
Windows	TCP/IP
z/OS	TCP/IP

IMS と WebSphere Application Server でグローバル・トランザクション・サポートを使用するには、詳細について、グローバル・トランザクションと 2 フェーズ・コミットのサポート・プロセスに関するトピックを参照してください。

関連概念:

1009 ページの『グローバル・トランザクションと 2 フェーズ・コミット・サポート処理』

関連情報:

965 ページの『IMS 接続ファクトリーの構成』

第 47 章 IMS TM リソース・アダプターのランタイム・コンポーネントのインストール

IMS TM リソース・アダプター・ランタイム・コンポーネントのインストールでは、インストール・アーカイブ・ファイルのコンテンツをターゲット・インストール・ディレクトリーに解凍します。これは、後で WebSphere Application Server などのアプリケーション・サーバーにデプロイするためです。

前提条件:

888 ページの『IMS TM リソース・アダプターを使用する準備』の準備ステップに従って、必要な IMS TM リソース・アダプターのバージョンを判別し、ランタイム・コンポーネントをダウンロードします。

IMS TM リソース・アダプターは、以下のように、.tar ファイルと .zip ファイルの両方として使用可能です。

表 *111.* さまざまなプラットフォームのさまざまな形式での *IMS TM* リソース・アダプターのインストール・ファイル

ファイル名	説明
ico*xxxx*.zip	圧縮フォーマットでの分散対象の IMS TM リソース・アダプターのランタイム・コンポーネント。ここで、*xxxx* は、インストールする IMS TM リソース・アダプターのバージョンです。
ico*xxxx*.tar	圧縮フォーマットでの分散対象の IMS TM リソース・アダプターのランタイム・コンポーネント。ここで、*xxxx* は、インストールする IMS TM リソース・アダプターのバージョンです。
ico*xxxx*zos.tar	圧縮フォーマットでの z/OS用の IMS TM リソース・アダプターのランタイム・コンポーネント。ここで、*xxxx* は、インストールする IMS TM リソース・アダプターのバージョンです。

IMS TM リソース・アダプターをインストールするには、以下のようにします。

1. ご使用のプラットフォーム用のアーカイブ・ファイルを解凍してからインストールします。
   - 890 ページの『分散プラットフォームでインストールするための圧縮ファイルの解凍』
   - 891 ページの『z/OS でインストールするための圧縮ファイルの解凍』
2. 解凍された内容を確認します。
3. 次のように、リソース・アダプター RAR ファイルをインストールします。
   - 893 ページの『WebSphere Application Server へのリソース・アダプターのインストール』
   - 896 ページの『WebSphere Liberty サーバーへのリソース・アダプターのインストール』
4. 次のように、接続ファクトリーを作成します。

- 895 ページの『WebSphere Application Server での接続ファクトリーの作成』
- 897 ページの『WebSphere Liberty サーバー用の接続ファクトリーの構成』

5. 898 ページの『インストール検査プログラムを使用したインストールの検査』

IMS TM リソース・アダプターを使用する準備

IMS TM リソース・アダプターを使用する前に、使用する IMS TM リソース・アダプターのバージョンと開発環境を先に決定する必要があります。

1. 使用する IMS TM リソース・アダプターのバージョンを決定します。 使用する IMS TM リソース・アダプターのバージョンは、IMS TM Resource Adapter に追加された新機能が必要かどうか、およびご使用の IMS のバージョンによって異なります。
   - 882 ページの『サポートされるソフトウェア構成』
2. 使用する統合開発環境 (IDE) を決定します。
3. IMS TM リソース・アダプターを IBM WebSphere Application Server (WebSphere Liberty サーバーを含む) 以外のアプリケーション・サーバーにデプロイしようとしている場合は、International Components for Unicode (ICU) バージョン 4.4.2 以降が必要です。このコンポーネントは、IMS TM リソース・アダプターのダウンロード・サイトにある WebSphere Liberty サーバー用の特別なランタイム・パッケージに含まれています。
4. 発生する可能性のあるマイグレーションの問題を検討し、その問題がユーザー自身に該当するかどうかを判断します。
5. 選択したバージョン用のランタイム・コンポーネントをダウンロードします。ランタイム・コンポーネントは、IMS TM リソース・アダプターのダウンロード・サイトからダウンロードできます。これは、SMP/E を使用して z/OS にインストールする場合にも取得できます。
6. 古いバージョンの IMS TM リソース・アダプターからアップグレードする場合は、 910 ページの『IMS TM リソース・アダプターのサービスおよび更新のインストール』のステップに従ってください。

IMS TM リソース・アダプターのマイグレーションに関する潜在的な問題

古いコネクター・フレームワーク、アーキテクチャー、または API を使用する既存のアプリケーションは、IMS TM リソース・アダプターの新しいバージョンにアップグレードするときに、変更が必要なことがあります。

非推奨の機能とサポート

- SYNC_RECEIVE_ASYNCOUTPUT 対話 Verb は IMS TM Resource Adapter バージョン 10 では非推奨になったため、SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT に置き換えられました。マイグレーション・ステップは不要です。

  新しい対話 Verb である SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT と SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT の違いについては、『非

同期出力プログラミング・モデル』のトピックを参照してください。この 2 つの Verb では、IMS OTMA 非同期保留キューに出力がないか確認する際に、対話をより厳密に制御できます。

削除された機能とサポート

- Common Connector Framework (CCF) はサポートされません。CCF を使用するアプリケーションは JCA 仕様にマイグレーションする必要があります。
- 非管理対象接続を使用するアプリケーション、または旧バージョンの JCA 1.0 ベースの IMS TM Resource Adapter を使用して生成されたアプリケーションを最新バージョンの IMS TM Resource Adapter で実行すると、警告が表示される可能性があります。IMS TM Resource Adapter バージョン 9 で使用可能な TransactionResourceRegistration プロパティーは、IMS TM リソース・アダプターの接続ファクトリーのカスタム・プロパティーから削除されました。 このプロパティーは、JCA 1.0 ベースのリソース・アダプターにおいて動的なトランザクションの参加をサポートしていました。この機能は JCA 1.5 ベースのリソース・アダプターに組み込まれたため、TransactionResourceRegistration プロパティーは不要になりました。
- WebSphere Studio Application Developer Integration Edition によって生成された MFS Web サービス・アプリケーションは、IMS TM Resource Adapter ではサポートされません。

関連概念:

921 ページの『非同期出力プログラミング・モデル』

IMS TM リソース・アダプターの更新

WebSphere Application Server バージョン 7 以降の管理コンソールにある「**RAR** の更新」機能を使用して、新しいバージョンの IMS TM リソース・アダプターをインストールします。

既存のすべての接続ファクトリーの削除と再構成を行わずに IMS TM リソース・アダプターのバージョンを更新するには、WebSphere Application Server の管理コンソールにある **RAR** の更新機能を利用し、それに合わせてクラス・パスを更新します。

1. WebSphere Application Server 管理コンソールで、「**Resources**」「**Resource Adapters**」をクリックして展開します。
2. 「**Resource adapters**」をクリックします。
3. チェック・ボックスをクリックして、更新する IMS TM リソース・アダプターを選択し、「**Update RAR**」をクリックします。

図 *112.* 新規バージョンのリソース・アダプター用の *RAR* の更新

4. ステップ 1 で、「**Local file system**」または「**Remote file system**」のいずれかを選択し、「**Browse**」をクリックして新しい IMS TM リソース・アダプター の RAR ファイルがある場所までナビゲートします。 例えば、ローカル・ファイル・システムでは、RAR ファイルは *install_path*/IBM/IMS/ICO*xx*/V*xxxx*/JCA15/ims*xxxx*.rar にあります。
5. 「**Next**」をクリックします。
6. ステップ 2 で、既存のリソース・アダプターと新しいリソース・アダプターのバージョン番号を確認し、「**Next**」をクリックします。
7. ステップ 3 で、プロンプトに従って、新しいバージョンのリソース・アダプターで導入された新しいプロパティーを構成します。「**Next**」をクリックします。
8. 要約ステップで変更内容を確認し、「**Finish**」をクリックします。 「Resource adapter」ページに戻ります。
9. 更新したリソース・アダプターの名前をクリックします。
10. 「General Properties」の下にあるクラスパス・フィールドで、RAR ファイル名を更新します。 例えば、IMS TM リソース・アダプターのバージョン 14.1 にアップグレードする場合は、クラスパスを ${CONNECTOR_INSTALL_ROOT}/imsico1410.RAR に変更します。
11. 「**OK**」をクリックします。 「Resource adapter」ページに戻ります。
12. ページの上部にある「**Save**」をクリックします。

IMS TM リソース・アダプターの RAR ファイルが更新されます。これで、既存のすべての接続ファクトリーが新しいバージョンの RAR を使用するようになります。

分散プラットフォームでインストールするための圧縮ファイルの解凍

分散プラットフォームに IMS TM リソース・アダプターをインストールする場合は、インストール・ファイルを圧縮ファイルから適切な場所に解凍して、後でアプリケーション・サーバーにデプロイできるようにします。

解凍ツールを使用して圧縮ファイルを開き、その内容をターゲットのインストール・ディレクトリーに解凍します。
インストール・アーカイブの内容を回答するための多くのオプション (プラットフォーム依存のものもあります) の 1 つは、次のようになります。

```
tar -xvf icoxxxx.tar install_path
```

tar プログラムは、UNIX バリアントおよび Linux バリアントでオペレーティング・システムの一部として使用可能です。tar プログラムのバージョンは、無料のソフトウェア・プログラム・ダウンロードとして、さまざまなソースからも入手できます。一部の解凍ツールは、.zip ファイルだけでなく、.tar ファイルも処理できることに注意してください。

解凍が正常に完了すると、圧縮ファイルの内容は、ディレクトリー *install_path*/IBM/IMS/ICO*xxx*/V*xxxx*に入れられます。ここで、*install_path* は、選択したターゲットのインストール・パスであり、V*xxxx* は、ダウンロードした IMS TM リソース・アダプターのインストール圧縮ファイルのバージョン番号です。例えば、Windows 上では c:¥Program Files¥IBM¥IMS¥ICO*xxx*¥V*xxxx*、AIX の場合は /opt/IBM/IMS/ICO*xxx*/V*xxxx*です。

解凍されたディレクトリー構造およびファイル内容の検証に進みます。

z/OS でインストールするための圧縮ファイルの解凍

z/OS プラットフォームでは、IMS に組み込まれている IMS Java On Demand 機能をインストールするか、IMS TM Resource Adapter Web サイトからダウンロードした z/OS TARファイルを解凍します。

IMS に組み込まれている IMS Java On Demand 機能をインストールする場合、SMP/E のインストール手順は「*Program Directory for Information Management System (IMS) Transaction and Database Servers*」に記載されています。

IMS TM リソース・アダプターをインストールするために IMS ダウンロードのページから z/OS TAR ファイルをダウンロードするには、以下のようにします。

1. ダウンロードした TAR ファイルをバイナリー・モードで z/OS HFS に転送します。
2. tar プログラムを使用して、TAR ファイルの内容をターゲット・インストール・ディレクトリーに解凍します。 以下に例を示します。

   ```
   tar -xvf icoxxxx.tar
   ```

SMP/E プロセスを使用するか、またはダウンロードした TAR ファイルの内容を手動で展開することによって、正常に解凍およびインストールされると、ファイルは、*install_path*/V*xxxx*に配置されます。ここで、*install_path* は/usr/lpp/ims/ico*xxx* であり、V*xxxx* は、ファイルの解凍元であった TAR ファイルの IMS TM リソース・アダプター のバージョン番号、あるいはインストールに使用した APAR または PTF です。

解凍されたディレクトリー構造およびファイル内容の検証に進みます。

IMS TM リソース・アダプターのランタイム・コンポーネントのファイル・コンテンツの検査

IMS TM リソース・アダプター・ランタイム・コンポーネントの .tar または .zip ファイルを解凍した後、正しいディレクトリーに必要なファイルがあることを確認します。

ファイル名	説明
/JCA15/ims*xxxx*.rar	IMS TM リソース・アダプターのリソース・アダプター・アーカイブ (RAR) ファイル。
/JCA15/imsicoivp.ear	IMS TM リソース・アダプターのインストール検査プログラム (IVP) のエンタープライズ・アーカイブ (EAR) ファイル。この IVP により、IMS TM リソース・アダプターが WebSphere Application Server に正しくデプロイメントされたかが検証されます。
/JCA15/imsicocalloutivp.ear	IMS TM リソース・アダプターのコールアウト IVP 用のエンタープライズ・アーカイブ (EAR) ファイル。この IVP にはメッセージ駆動型 Bean (MDB) が含まれています。この Bean を使用して、アプリケーションが IMS TM リソース・アダプターを使用してホストIMS システムからコールアウト要求を受信し、同期コールアウト要求に対する応答を返すことができるかを検証できます。
/licenses/license_*xx*.txt	すべてのサポート対象言語でのご使用条件。ユーザーは、IMS TM リソース・アダプターをインストールまたは使用する前に、これらのご使用条件のすべての条件に同意する必要があります。z/OS プラットフォームでは、これらのご使用条件は z/OS 圧縮ファイル (ims*xxxx*zos.tar および ims*xxxx*zos.tar) で EBCDIC エンコード・テキスト・ファイルとしてフォーマットされ、z/OS HFS で直接表示できます。
/README.html	新機能とバグ修正に関する情報を含む README ファイル。

解凍したファイルのディレクトリー構造とファイル・コンテンツが同じにならない場合は、.tar ファイルまたは .zip ファイルの内容をもう一度正しいディレクトリーに解凍する必要があります。

IMS TM リソース・アダプターの解凍済み RAR ファイルのインストールを続けます。

- 893 ページの『WebSphere Application Server へのリソース・アダプターのインストール』
- 896 ページの『WebSphere Liberty サーバーへのリソース・アダプターのインストール』

WebSphere Application Server へのリソース・アダプターのインストール

WebSphere Application Server 管理コンソールを使用して、IMS TM リソース・アダプターの RAR ファイルをデプロイします。

重要:

1. IMS TM リソース・アダプター・アーカイブ (RAR) を汎用アプリケーション・サーバーにインストールする場合、IMS TM リソース・アダプターで使用するために追加のクラス・ライブラリーが必要になることがあります。 IMS TM リソース・アダプター・インストール・アーカイブに含まれていないクラス・ライブラリーの場合は、必要な JAR ファイルのライセンス交付およびロケーションに関する情報について、選択した開発環境のプロダクト担当者にお問い合わせください。
2. 「**Install RAR**」ダイアログを使用して RAR ファイルをインストールする場合、「Resource Adapters」ページで定義した有効範囲は、RAR ファイルのインストール場所に影響しません。RAR ファイルは、ノード・レベルでのみインストールすることができます。このレベルは、「Install RAR」ページで指定します。RAR ファイルの有効範囲を特定のクラスターまたはサーバーに設定するには、各ノード・レベルで RAR ファイルをインストールした後で、適切なクラスター有効範囲またはサーバー有効範囲を持つ RAR ファイルのコピーを作成します。

前提条件:

- IMS TM リソース・アダプター RARを、ご使用の WebSphere Application Server からアクセス可能なファイル・システムにインストールしてある必要があります。
- WebSphere Application Server を始動して、管理コンソールにログインします。

WebSphere Application Server に RAR をデプロイするには、以下のようにします。

1. 管理コンソール (Integrated Solutions Console とも呼ばれます) のナビゲーション・ペインで、「**Resources**」 > 「**Resource Adapters**」 > 「**Resource adapters**」をクリックします。

図 *113. WebSphere Application Server* でのリソース・アダプター構成

2. 使用しない RAR が既にインストールされている場合は、最初にそれを削除します。 WebSphere Application Server を停止して、再始動します。前のステップを繰り返して、リソース・アダプター構成ページに戻ります。
3. コンテンツ・ペインで、「**Install RAR**」をクリックします。 ボタンを見つけるのに、ペインでスクロールダウンが必要な場合があります。 「Install RAR File」ページが表示されます。
4. RAR ファイルの場所に応じて、「**Local file system**」または「**Remote file system**」をクリックします。
   - 「**Local file system**」を選択した場合は、「**Browser**」をクリックして、インストール済みの IMS TM リソース・アダプター RARファイルの位置を確認して指定します (例えば、*install_path*/IBM/IMS/ICO*xx*/V*xxxx*/JCA15/ims*xxxx*.rar)。
5. 「**Next**」をクリックします。
6. 「**Configuration**」ページで、RAR の名前を入力します (例えば、imstmrav*xxxx*)。「**OK**」をクリックする。
7. オプションで、異なる有効範囲レベルの RAR ファイルのコピーを作成します。各ノード・レベルで RAR ファイルをインストールした後、そのファイルの有効範囲として特定のサーバーまたはクラスターを持つファイルのコピーをもう 1 つ別に作成できます。WebSphere Application Server の Knowledge Center にあるトピック 『リソース・アダプター・アーカイブのインストール (Installing a resource adapter archive)』を参照してください。
8. 結果をマスター構成に保管するには、パネルの上部にある「**Save**」をクリックします。

IMS TM リソース・アダプターのランタイム RAR ファイルがデプロイされます。

IMS TM リソース・アダプターのための接続ファクトリーの作成に進むことができます。

関連タスク:

1019 ページの『第 49 章 スタンドアロン WebSphere Application Server でのアプリケーションの実行』

WebSphere Application Server での接続ファクトリーの作成

IMS TM リソース・アダプターを WebSphere Application Server にインストールおよびデプロイしたら、IMS TM リソース・アダプターに接続ファクトリーを作成します (接続ファクトリーがない場合)。

接続ファクトリーは、IMS と IMS Connect との間に接続を作成するために 1 つ以上のアプリケーションが使用します。

前提条件: WebSphere Application Server を始動して、WebSphere Application Server 管理コンソールにログインします。

IMS TM リソース・アダプターの接続ファクトリーを作成するには、以下のようにします。

1. WebSphere Application Server 管理コンソールで、「**Resources**」 > 「**Resource Adapters**」をクリックして展開します。
2. 「**Resource adapters**」をクリックします。
3. 接続ファクトリーを作成する、デプロイ済みの IMS TM リソース・アダプターの名前をクリックします。
4. 「Additional Properties」セクションで、「**J2C connection factories**」を選択します。
5. 「**New**」をクリックし、「Name」フィールドに、接続ファクトリーの名前を入力します。 例えば、`myIMSTMRA` と入力します。
6. 必要に応じて、JNDI 名を指定します。
7. 「**Apply**」をクリックします。
8. ページの上部にある「**Save**」をクリックします。
9. 上記で作成した接続ファクトリーの名前 (myIMSTMRA) をクリックします。「Configuration」ページが開きます。
10. 「Configuration」ページの「Additional Properties」の下の列で、「**Custom properties**」をクリックします。
11. 各フィールドをクリックして、接続ファクトリーを構成するための値を入力します。「Host name」、「port number」、および「data store name」を構成する必要があります。
12. プロパティーの構成が済んだら、ページの上部にある「**Save**」をクリックします。
13. サーバーを停止してから再始動すると、新しい接続ファクトリーが表示されます。

これで、IMS TM リソース・アダプターの接続ファクトリーが作成されました。

インストール検査プログラム (IVP) を使用したインストールの検査に進みます。

WebSphere Liberty サーバーへのリソース・アダプターのインストール

リソース・アダプターをインストールする前に、IMS TM リソース・アダプター RAR ファイルをダウンロードして、WebSphere Application Server Liberty Profile 環境をセットアップします。

以下のソフトウェアが必要です。

- IMS TM Resource Adapter バージョン 13 以降のランタイム・パッケージ (WebSphere Liberty Profile 用)
- WebSphere Application Server バージョン 8.5.5.4 Liberty Profile 以降 (JCA フィーチャーがインストール済みのもの)

前提条件:

1. ご使用の WebSphere Liberty サーバーのランタイム環境が適切にインストールされ、セットアップされている必要があります。

   WebSphere Application Server Liberty Profile Repository (https://developer.ibm.com/wasdev/downloads/) から JCA Connection Architecture 1.6 フィーチャーをダウンロードします。検索ボックスで「JCA Connection Architecture 1.6」を検索します。インストールおよび構成の手順は、そのページに示されます。
2. WebSphere Application Server Liberty Profile 用の IMS TM リソース・アダプター ランタイム・コンポーネント・アーカイブ・ファイルを IMS TM リソース・アダプターWeb サイトからダウンロードします。使用するランタイム・オファリングのバージョンを選択し、WebSphere Liberty Profile の IMS TM Resource Adapter v*xxxx* ランタイム・コンポーネントを選択します。

IMS TM リソース・アダプターをインストールするには、`server.xml` ファイルを編集する必要があります。WebSphere Application Server Liberty Profile は、グラフィカル管理コンソールを提供しません。一般に、WebSphere Application Server Developer Tools for Eclipse またはテキスト・エディターを使用して、`server.xml` ファイルを編集することができます。

リソース・アダプターをインストールするには、以下のステップを実行します。

1. エディターで、WebSphere Liberty サーバー・インスタンスの `server.xml` ファイルを開きます。
2. JCA 1.6 機能および JNDI 1.0 機能を `<featureManager>` セクションに追加して、これらの機能を有効にします。

   ```
   <featureManager>
      ...
      <feature>jca-1.6</feature>
      <feature>jndi-1.0</feature>
      ...
   </featureManager>
   ```
3. 次のように `<resourceAdapter>` エントリーを追加して、IMS TM リソース・アダプターをインストールします。

   ```
   <resourceAdapter id="IMS_adapter_id" location="IMS_adapter_RAR_file_location">
   ```

IMS_adapter_id は、このリソース・アダプターを、インストール済みの他のリソース・アダプターと区別するために使用されます。この値は、後で、IMS に接続するよう接続ファクトリーを構成する際に使用されます。

IMS_adapter_RAR_file_locationn は、IMS TM リソース・アダプター RAR ファイルへの、リソース・アダプター・ファイル名を含む絶対パスです。

server.xml は、次のようになる場合があります。

```
<server description="myLibertyServer">
   <!-- Enable features -->
   <featureManager>
      <feature>jsp-2.2</feature>
      <!-- Added jca-1.6 and jndi-1.0 features required by the IMS
           TM resource adapter -->
      <feature>jca-1.6</feature>
      <feature>jndi-1.0</feature>
   </featureManager>

   <!-- To access this server from a remote client add a host attribute
    to the following element, e.g. host="*" -->
   <httpEndpoint id="defaultHttpEndpoint" httpPort="9080"
    httpsPort="9443" />

    <resourceAdapter id="IMSTMRA"
     location="C:\IBM\IMS\ico1410\IBM\IMS\ICO14\V1410\JCA15\imsico1410.rar" />

</server>
```

この例では、IMSTMRA はリソース・アダプター ID で、imsico1410.rar ファイルを指します。

4. 変更内容を保管します。
5. WebSphere Liberty サーバーを始動します。

コンソールで、インストールが正常に行われたことを示す、以下の J2CA7001I メッセージが発行されます。

```
J2CA7001I: Resource adapter imsicoxxxx installed in 0.735 seconds.
```

次に、ご使用の IMS ホスト・システムに接続するように接続ファクトリーを構成する必要があります。

関連情報:

基本 JCA リソース・アダプターの構成およびデプロイ (WebSphere Application Server Liberty Core V8.5.5)

WebSphere Liberty サーバー用の接続ファクトリーの構成

V13.2 以降 IMS TM ホスト・システムに接続するために server.xml ファイルに接続ファクトリーを構成します。

WebSphere Liberty サーバーでは、接続ファクトリーは server.xml ファイルに構成されます。

1. server.xml ファイルを開きます。
2. <connectionFactory> エントリーを追加することによって、ご使用の IMS ホスト・システムに接続するために接続ファクトリーを構成します。

```
<connectionFactory jndiName="your_JNDI_name" type="javax.resource.cci.ConnectionFactory">
   <properties.IMS_adapter_id hostName="your_host_name"
      portNumber="xxxx" dataStoreName="data_store_name" />
</connectionFactory>
```

IMS_adapter_id は、リソース・アダプターをインストールしてデプロイしたときに `<resourceAdapter>` エントリーに指定された ID です。

`server.xml` は、次のようになる場合があります。

```
<server description="myLibertyServer">
   <!-- Enable features -->
   <featureManager>
      <feature>jsp-2.2</feature>
      <!-- Added jca-1.6 and jndi-1.0 features required by the IMS
           TM resource adapter -->
      <feature>jca-1.6</feature>
      <feature>jndi-1.0</feature>
   </featureManager>
   . . .
   <!-- Added connection factory for the IMS TM resource adapter -->
   <connectionFactory jndiName="myJNDIName"
       type="javax.resource.cci.ConnectionFactory">
       <properties.IMSTMRA hostName="MY.IMS.IBM.COM"
           portNumber="9999" dataStoreName="IMS1" />
   </connectionFactory>

   <resourceAdapter id="IMSTMRA"
     location="C:\IBM\IMS\ico1410\IBM\IMS\ICO14\V1410\JCA15\imsico1410.rar" />
</server>
```

3. 変更内容を保管します。

提供されているインストール検査プログラム (IVP) のデプロイおよびテストによる接続のテストに進む準備が整いました。

関連情報:

基本 JCA リソース・アダプターの構成およびデプロイ (WebSphere Application Server Liberty Core V8.5.5)

インストール検査プログラムを使用したインストールの検査

IMS TM リソース・アダプターのインストール検査プログラム (IVP) を使用して、エンタープライズ・アプリケーションが IMS TM リソース・アダプターを使用してアプリケーション・サーバーからターゲット IMS システムに正常にアクセスできることを確認します。

IMS TM リソース・アダプター IVP は、IMS TM リソース・アダプター ランタイム・コンポーネントのインストール時にインストールされる単純な Java EE アプリケーションです。IVP は、/STA OTMA コマンドに応答して、IMS からメッセージを受信することを目的としています。IVP では IMS アプリケーション・プログラムを実行しないため、IMS アプリケーション・プログラムの実行は必要ありません。

ヒント: IVP は、IMS トランザクションが正常に実行できるかどうかを検査しません。実行するのは、以下の検査のみです。

- クライアント、IMS Connect、および IMS 間の通信が機能していること。

- IMS が実行されており、IMS コマンドを実行し、そのコマンドの出力を返せること。
- クライアントが IMS コマンド要求をサブミットし、そのコマンドからの出力を受信できること。

したがって、IVP にはホスト IMS アプリケーションは不要です。

IVP を実行するための前提条件

IVP を実行する前に、いくつかのコンポーネントがターゲット・ホスト・システムで実行中であることを確認する必要があります。

- IMS Connect の場合:
  - ターゲット・システムのシステム・コンソールに、IMS Connect の未解決の応答が表示されていることを確認します。以下に例を示します。

    ```
    HWSC0000I *IMS CONNECT READY* IMS_Connect_Name
    ```

    IMS Connect がアクティブなときに HWSC0000I メッセージがシステム・コンソールに表示されるようにするために、IMS Connect HWSCFG*xx* 構成メンバーの HWS ステートメントに **WTORCMD=Y** が指定されていること、またはその HWS ステートメントから **WTORCMD** パラメーターが省略されていることを確認します。
  - IMS Connect コマンド VIEWHWS を IMS Connect の未解決応答に入力して、ターゲット・データ・ストアとポートがアクティブであることを確認します。必要な場合は、IMS Connect コマンド OPENDS *datastore_name* および OPENPORT *port_number* をそれぞれ使用して、データ・ストアとポートを活動化します。
- IMS の未解決応答がターゲット・システムのシステム・コンソールに表示されていることを確認します。以下に例を示します。

  ```
  DFS996I *IMS READY*
  ```
- IMS の未解決応答に IMS コマンド /DISPLAY OTMA を入力して、IMS と IMS Connect の両方のメンバーの XCF 状況がアクティブであることを確認します。

  ```
  DFS000I   GROUP/MEMBER   XCF-STATUS    USER-STATUS SECURITY
         DFS000I   GROUPNM
         DFS000I   -IMSMEM        ACTIVE        SERVER      NONE
         DFS000I   -ICONNMEM      ACTIVE        ACCEPT TRAFFIC
  ```
- IMS TM リソース・アダプターの最新の更新を取得して、フィックスと機能拡張を入手します。
- 接続ファクトリーが作成済みであることを確認します。

  WebSphere Application Server の場合は、895 ページの『WebSphere Application Server での接続ファクトリーの作成』を参照してください。

これらの前提条件が満たされたら、WebSphere Application Server への IVP EAR ファイルのデプロイに進んでください。

Java EE アプリケーション・サーバーでの IVP EAR ファイルのデプロイ

IMS TM リソース・アダプターのインストール検査プログラム (IVP) は、エンタープライズ・アーカイブ (EAR) ファイルとしてパッケージされており、IMS TM リソース・アダプター (RAR ファイル) のインストールが正常に行われたかどうかをテストするには、この EAR ファイルをアプリケーション・サーバーにデプロイする必要があります。

前提条件:

- IMS ホスト・システムに接続するために、接続ファクトリーを作成する必要があります。まだ実行していない場合は、IVP で使用するための接続ファクトリーを作成します。
- アプリケーション・サーバーが始動し、適切に実行されている必要があります。
- WebSphere Application Server では、サーバーの管理コンソールにログインしている必要があります。

WebSphere Application Server への IVP EAR ファイルのデプロイ

WebSphere Application Server に IVP EAR ファイルをデプロイするには、まず、IVP EAR ファイルをアップロードして新規アプリケーションを作成し、次にそのアプリケーションを開始します。

WebSphere Application Server 管理コンソールで、以下のようにします。

1. 以下のように、エンタープライズ・アプリケーションをインストールします。
   a. 管理コンソールのナビゲーション・ツリーで、「**Applications**」を展開し、「**New Application**」をクリックします。
   b. 「New Application」ページで、「**New Enterprise Application**」をクリックします。
   c. IVP 用の EAR ファイル (`imsicoivp.ear`) の場所に応じて、「**Local file system**」または「**Remote file system**」をクリックします。
   d. 「**Browse**」をクリックし、EAR ファイルがある場所にナビゲートします。ファイル `imsicoivp.ear` は *target_dir*`/IBM/IMSICO/ICO`*xx*`/`*xx*`/JCA`*xx*`/` にあります。*target_dir* はユーザーの選択したディレクトリー、*xx* は IMS TM リソース・アダプターのインストール済み環境に対応するバージョン番号です。
   e. 「**Next**」をクリックします。
   f. 「Preparing for application installation」ページで、「**Next**」をクリックします。
   g. 「**Fast Path**」をクリックします。 「Fast Path」インストール・オプションでは、ご使用のアプリケーションまたはモジュールの内容に基づいて、必要なオプションのみが表示されます。
   h. 「**Next**」をクリックします。 インストール・オプションを指定してモジュールをサーバーにマップするように求めるプロンプトが出されます。
   i. ステップ 1 で、必要に応じてインストール・オプションを指定または変更し、「**Next**」をクリックします。

j. ステップ 2 で、必要に応じてこのモジュールをマップするサーバーを変更し、「**Next**」をクリックします。

k. 「Summary」ページで、表示された情報を確認し、「**Finish**」をクリックします。

l. 「**Save directly to the master configuration**」リンクをクリックします。IVP EAR ファイルがインストールされます。

2. エンタープライズ・アプリケーションを開始します。

a. 管理コンソールで、「**Applications**」 > 「**Application Types**」 > 「**WebSphere enterprise applications**」を展開します。 「Enterprise Applications」ページに IMSICOIVPServiceEAR が表示されます。

b. アプリケーション IMSICOIVPServiceEAR を選択します (その横に表示されるチェック・ボックスを選択します)。「**Start**」をクリックしてアプリケーションを開始します。 状況が緑の矢印に変わります。これは、アプリケーションが開始されたことを示しています。

ヒント: 状況が「始動済み」に変わらない場合は、サーバーを再始動してみてください。

IMS TM リソース・アダプター IVP の EAR ファイルのデプロイが完了しました。

これで、IMS TM リソース・アダプター IVP を実行する準備ができました。

WebSphere Liberty サーバーへの IVP EAR ファイルのデプロイ

V13.2 以降 WebSphere Liberty サーバーに IVP EAR ファイルをデプロイするには、server.xml ファイルを編集します。

1. server.xml ファイルを開きます。
2. <enterpriseApplication> エントリーを以下のように追加します。

```
<enterpriseApplication location="file_location" type="ear">
      <classloader classProviderRef="IMS_adapter_id" />
 </enterpriseApplication>
```

IMS_adapter_id は、<resourceAdapter> タグに設定したリソース・アダプター ID の値に設定する必要があります。例えば、server.xml は次のようになります。

```
<server description="myLibertyServer">
   <!-- Enable features -->
   <featureManager>
      <feature>jsp-2.2</feature>
      <!-- Added jca-1.6 and jndi-1.0 features required by the IMS
           TM resource adapter -->
      <feature>jca-1.6</feature>
      <feature>jndi-1.0</feature>
   </featureManager>

    <resourceAdapter id="IMSTMRA"
     location="C:\IBM\IMS\ico1410\IBM\IMS\ICO14\V1410\JCA15\imsico1410.rar" />

   <enterpriseApplication location="imsicoivp.ear" type="ear">
      <classloader classProviderRef="IMSTMRA" />
   </enterpriseApplication>
```

IMS TM リソース・アダプター IVP の EAR ファイルのデプロイが完了しました。

これで、IMS TM リソース・アダプター IVP を実行する準備ができました。

IMS TM リソース・アダプターの IVP の実行

IMS TM リソース・アダプター IVP は任意の Web ブラウザーから実行できます。

IMS TM リソース・アダプター の IVP を実行するには、以下のようにします。

1. Web ブラウザーで、以下の URL を入力して IVP を呼び出します。
   - アプリケーション・サーバーがローカル・システムで実行されている場合:
     `http://localhost:9080/IMSICOIVPServiceWeb/IMSICOIVPInputForm.html`
   - アプリケーション・サーバーがリモート・システムで実行されている場合:
     `http://remote_server:port/IMSICOIVPServiceWeb/IMSICOIVPInputForm.html`

   remote_server:port は、アプリケーション・サーバーが実行されているホスト名とポート番号です。
2. 「**Submit**」をクリックします。

   ご使用のセキュリティー構成に応じて、IMS から以下のメッセージのいずれかを受信します。
   - DFS1292E SECURITY VIOLATION
   - DFS058I *hh:mm:ss* START COMMAND COMPLETED

   これらのメッセージがいずれも表示されない場合、IVP は正常に実行されませんでした。サーバーを再始動します。サーバーを再始動しても問題が解決しない場合は、 1021 ページの『IVP 障害の診断』のトピックを参照してください。

IMS コールアウト要求の処理に IMS TM リソース・アダプターを使用する計画がある場合は、IMS TM リソース・アダプターのコールアウト IVP サンプルを使用して、環境が正しくセットアップされているかテストします。

IMS TM リソース・アダプターのコールアウト IVP サンプルの実行

コールアウト IVP サンプルには、サンプル IMS アプリケーションからの同期および非同期の両方のコールアウト要求を処理するように設計されたメッセージ駆動型 Bean が含まれています。また、コールアウト IVP サンプルには、サンプル IMS アプリケーションと、それに関連する IMS ホスト・システム上でのセットアップ・ファイルが含まれていて、サンプル IMS アプリケーションからメッセージ駆動型 Bean へのコールアウト要求を発行して処理させることができます。

前提条件: インストール検査プログラム (IVP) には、IMS OTMA 宛先記述子とトランザクション・パイプ (TPIPE) をセットアップするために変更可能なジョブとタスク一式が用意されています。 OTMA 宛先記述子は、1 つは同期コールアウト要求、もう 1 つは非同期コールアウト要求のためのものであり、コールアウト要求をどこでキューに入れるか (TPIPE 名) を表します。

コールアウト要求の発行用に提供されている IMSアプリケーションを実行するために、2 つのジョブが用意されています。

表 *112.* 非同期コールアウトと同期コールアウトのサンプル用の *IVP* ジョブ、*TPIPE* 名、*OTMA* 宛先記述子、および *COBOL* アプリケーション・パーツ名

コールアウト要求のタイプ	ジョブ	**TPIPE**	**OTMA** 宛先記述子	**COBOL** アプリケーションのパーツ名
非同期	IV_S227J	IVPPIPE3	IVPDTOR3	DFSASCBL
同期	IV_S228J	IVPPIPE4	IVPDTOR4	DFSSSCBL

シンプルな IMS BMP アプリケーションが 2 つ用意されています。

- 1 つのアプリケーションは、IMS DL/I テスト・プログラム DFSDDLT0 を使用して、同期コールアウト要求を送信するための DL/I ICAL 呼び出しを発行します。この ICAL 呼び出しは、コールアウト要求を保持するための TPIPE、コールアウト・メッセージの経路指定のための OTMA 宛先記述子、およびコールアウト・メッセージ内の要求域と応答域を指定します。
- もう 1 つのアプリケーションは、IMS DL/I テスト・プログラム DFSDDLT0 を使用して、非同期コールアウト要求を IVPPIPE3 TPIPE に入れるために代替 PCB (プログラム連絡ブロック) 呼び出しに挿入 (ISRT) を発行します。

IMS TM リソース・アダプターのコールアウト IVP サンプルはシンプルなメッセージ駆動型 Bean であり、IMS TM Resource Adapter ランタイム・インストールによってインストールされます。この Bean を WebSphere Application Server 環境にデプロイして、この Bean が IMS アプリケーションからの非同期コールアウト要求と同期コールアウト要求のどちらも受信できること、および同期コールアウト要求の場合は応答を送信できることを検証できます。このサンプルは、事前定義された TPIPE と OTMA 宛先記述子を使用して、IMS コールアウト IVP サンプルで動作するように設計されています。

IMS アプリケーション用のコールアウト・ソリューションを実装するために行う一般の作業は以下のとおりです。

表 *113. IMS* アプリケーションのコールアウト・ソリューションを実装するためのステップ：

ステップ	説明	作業を実行する環境	コールアウト・サンプルによる作業の処理方法
1	IMS アプリケーションを作成または変更して、DL/I ICAL 呼び出し (同期) または ISRT altpcb 呼び出し (非同期) を使用してコールアウト要求を発行します。	IMS ホスト・システム	IMS コールアウト・サンプル IVP によって IMS アプリケーションが提供されます。 IMS のインストール情報で関連する IVP ジョブを参照してください。 IVP ジョブを実行してサンプルをコンパイルおよびバインドします。 • IMS V14 のインストール情報 • IMS V13 のインストール情報 • IMS V12 のインストール情報
2	OTMA 宛先記述子を定義します。		
3	新しく定義された OTMA 宛先記述子を使用するために IMS を再始動します。		
4	メッセージ駆動型 Bean または J2C アプリケーションを作成して IMS インバウンド・トランザクションを処理します。	IBM Rational Application Developer for WebSphere Software. またはその他のアプリケーション開発環境	IMS TM リソース・アダプターのコールアウト IVP サンプルがこのメッセージ駆動型 Bean を提供します。

表 113. IMS アプリケーションのコールアウト・ソリューションを実装するためのステップ (続き):

ステップ	説明	作業を実行する環境	コールアウト・サンプルによる作業の処理方法
5	IMS TM リソース・アダプターのランタイムを Java EE アプリケーション・サーバーにデプロイします。	Java EE アプリケーション・サーバー。	以下を参照してください。 • 893 ページの『WebSphere Application Server へのリソース・アダプターのインストール』 • 896 ページの『WebSphere Liberty サーバーへのリソース・アダプターのインストール』
6	Java EE アプリケーション・サーバーで J2C アクティベーション・スペックを構成し、IMS TM リソース・アダプターの接続情報 (例えば、IMS ホスト名、ポート番号、データ・ストア名、キュー名 (TPIPE 名)) を指定してコールアウト・メッセージをプルします。		WebSphere Application Server の場合は、 907 ページの『コールアウト IVP のための J2C アクティベーション・スペックの構成』を参照してください。 WebSphere Liberty サーバーの場合、J2C アクティベーション・スペックの情報は、ステップ 7 のデプロイメントに関するトピックの中で示されます。
7	Java EE アプリケーション・サーバーにメッセージ駆動型 Bean または J2C アプリケーションをデプロイし、そのアプリケーションを開始します。		以下を参照してください。 • 906 ページの『コールアウト要求を処理するためのサンプル・アプリケーションの WebSphere Application Server へのデプロイ』 • 908 ページの『コールアウト要求を処理するためのサンプル・アプリケーションの WebSphere Liberty サーバーへのデプロイ』
8	IMS アプリケーションを実行して、同期コールアウト要求または非同期コールアウト要求を発行します。	IMS ホスト・システム	909 ページの『IMS ホスト・コールアウト IVP アプリケーションの実行』を参照してください。

コールアウト要求を処理するためのサンプル・アプリケーションの WebSphere Application Server へのデプロイ

IMS 同期コールアウト要求を listen するコールアウト IVP アプリケーションを WebSphere Application Server にデプロイします。

前提条件: 907 ページの『コールアウト IVP のための J2C アクティベーション・スペックの構成』を完了します。

1. 以下のようにしてコールアウト・アプリケーションをインストールします。
   a. 管理コンソールのナビゲーション・ツリーで、「**Applications**」を展開し、「**New Application**」をクリックします。
   b. 「New Application」ページで、「**New Enterprise Application**」をクリックします。
   c. IVP 用の EAR ファイル (`imsicocalloutivp.ear`) の場所に応じて、「**Local file system**」または「**Remote file system**」をクリックします。
   d. 「**Browse**」をクリックし、EAR ファイルがある場所にナビゲートします。ファイル `imsicocalloutivp.ear` は *target_dir*`/IBM/IMSICO/ICO`*xx*`/`*xx*`/JCA`*xx*`/` にあります。*target_dir* はユーザーの選択したディレクトリー、*xx* は IMS TM リソース・アダプターのインストール済み環境に対応するバージョン番号です。
   e. 「**Next**」をクリックします。
   f. 「Preparing for application installation」ページで、「**Next**」をクリックします。
   g. 「**Fast Path**」をクリックします。 「Fast Path」インストール・オプションでは、ご使用のアプリケーションまたはモジュールの内容に基づいて、必要なオプションのみが表示されます。
   h. 「**Next**」をクリックします。 インストール・オプションを指定してモジュールをサーバーにマップするように求めるプロンプトが出されます。
   i. ステップ 1 で、必要に応じてインストール・オプションを指定または変更し、「**Next**」をクリックします。
   j. ステップ 2 で、必要に応じて、このモジュールをマップするサーバーを変更し、「**Next**」をクリックします。
   k. 「Summary」ページで、表示された情報を確認し、「**Finish**」をクリックします。
   l. 「**Save directly to the master configuration**」リンクをクリックします。コールアウト IVP EAR ファイルがインストールされます。
2. アプリケーションを開始します。
   a. 管理コンソールで、「**Applications**」 > 「**Application Types**」 > 「**WebSphere enterprise applications**」を展開します。 「Enterprise Applications」ページに IMSICOIVPServiceEAR が表示されます。
   b. アプリケーション IMSICOCalloutIVPMDBEAR を選択します (その横に表示されるチェック・ボックスを選択します)。「**Start**」をクリックしてアプリケーションを開始します。 状況が緑の矢印に変わります。これは、アプリケーションが開始されたことを示しています。

IMS からのコールアウト要求を listen するアプリケーションの準備ができました。

909 ページの『IMS ホスト・コールアウト IVP アプリケーションの実行』に進みます。

コールアウト IVP のための J2C アクティベーション・スペックの構成

WebSphere Application Server 管理コンソールで、リソース・アダプターの IVP サンプルのジョブおよびタスクで既に定義されている接続情報をセットアップして、コールアウト・メッセージを取り出して応答を返信できるようにします。

IMS コールアウト・サンプル IVP を使用するには、同期コールアウト要求を入れるキュー名 (TPIPE) を IVPPIPE4 に設定する必要があります。この TPIPE 値は、IMS OTMA 宛先記述子の中で既に定義されています。また、IMS Connect が実行されている IMS ホスト名とポート番号、およびデータ・ストア名も取得する必要があります。

WebSphere Application Server で J2C アクティベーション・スペックを作成および構成するには、以下のようにします。

1. IMS TM リソース・アダプターの J2C スペックを作成します。
   a. 管理コンソールのナビゲーション・ペインで、「**Resources**」 > 「**Resource Adapters**」 > 「**J2C activation specifications**」を選択します。
   b. コンテンツ・ペインで「**New**」をクリックします。 「Configuration」タブが表示されます。
   c. リスから IMS TM リソース・アダプターを選択します。 このページ下部の「Message listener type」フィールドは、その選択に基づいて自動的に設定されます。
   d. この J2C アクティベーション・スペックの名前を指定します。 例えば、`IMSICOCalloutIVP` とします。
   e. JNDI 名を入力します。 例えば、`eis/IMSICOCalloutIVP` とします。
   f. 「**Apply**」をクリックします。 J2C アクティベーション・スペックのページに戻ります。
   g. 上部のメッセージ・ボックスで「**Save**」をクリックし、マスター構成への変更を保管します。 J2C アクティベーション・スペックが作成されます。
2. IMS TM リソース・アダプターで使用するように J2C アクティベーション・スペックを構成するには、以下のようにします。
   a. IMS TM リソース・アダプターの J2C アクティベーション・スペックの名前をクリックします。 「Configuration」タブが表示されます。
   b. サイドにある「Additional Properties」セクションで、「**J2C activation specification custom properties**」リンクをクリックします。 J2C アクティベーション・スペックのこのインスタンスのカスタム・プロパティーが表示されます。
   c. テーブルで以下の必須プロパティーの名前をクリックし、該当する値を入力して、これらのプロパティーの値を指定します。
      - queueName:

- 非同期と同期の両方のコールアウト・メッセージを検査しようとしている場合は、IVPPIPE3,IVPPIPE4 を指定します。
- 非同期コールアウト・メッセージのみを検査しようとしている場合は、IVPPIPE3 を指定します。
- 同期コールアウト・メッセージのみを検査しようとしている場合は、IVPPIPE4 を指定します。

• portNumber: IMS Connect のポート番号
• hostName: IMS Connect のホスト名
• dataStoreName: IMS データ・ストア名

d. 環境に基づいて、その他のオプション・プロパティーの値を指定します。
e. 「**Apply**」をクリックします。
f. 上部のメッセージ・ボックスで「**Save**」をクリックし、マスター構成への変更を保管します。

3. サーバーを再始動して、新しいアクティベーション・スペックを確認します。

サンプル・アプリケーションのアクティベーション・スペックが構成されます。

906 ページの『コールアウト要求を処理するためのサンプル・アプリケーションの WebSphere Application Server へのデプロイ』に進みます。

コールアウト要求を処理するためのサンプル・アプリケーションの WebSphere Liberty サーバーへのデプロイ

コールアウト IVP EAR ファイルを WebSphere Liberty サーバーにデプロイするには、必要な拡張コンテンツおよびメッセージ駆動型 Bean (MDB) フィーチャーをダウンロードします。その後、server.xml ファイルを変更して、MDB フィーチャーおよび JCA アクティベーション・スペックを構成します。

前提条件**:** コールアウト要求の処理には、WebSphere Application Server Liberty Profile 拡張コンテンツおよび MDB フィーチャーが必要です。それらをまだダウンロードおよびインストールしていない場合は、次のようにします。

1. WebSphere Application Server for z/OS Liberty リポジトリー (https://developer.ibm.com/wasdev/downloads/)にアクセスします。
2. 「Extended Content」を検索します。 画面上の指示に従って、WebSphere Application Server Liberty Profile 拡張コンテンツを提供する Liberty Profile 拡張プログラミング・モデルをインストールします。
3. 「MDB」を検索します。 画面の指示に従って、「Message-Driven Beans 3.1」フィーチャーをインストールします。

server.xml ファイルで以下の変更を行います。

1. MDB フィーチャーを構成します。

```
<!-- Enable features -->
<featureManager>
   ...
  <!-- Added the mdb-3.1 feature required for callout support -->
   <feature>mdb-3.1</feature>
</featureManager>
```

2. MDB がリスナーとして動作するように、リソース・アダプターを使用する JCA アクティベーション・スペックを構成します。 JCA アクティベーション・スペックは、次のように <activationSpec> タグを追加することによって構成されます。

```
<activationSpec id="imsicocalloutivp/IMSICOCalloutIVPMDB/IMSICOCalloutIVPMDB">
     <properties.IMS_adapter_id hostName="host_name" portNumber="port_number"
      dataStoreName="data_store_name" queueNames="tpipe_name"/>
 </activationSpec>
```

IMS_adapter_id は、リソース・アダプターをインストールしてデプロイしたときに <resourceAdapter> エントリーに指定された IMS TM リソース・アダプターの ID です。前の例では、ID を IMSTMRA に設定しているため、対応するアクティベーション・スペックは次のようになります。

```
<activationSpec id="imsicocalloutivp/IMSICOCalloutIVPMDB/IMSICOCalloutIVPMDB">
     <properties.IMSTMRA hostName="my.host.server.com" portNumber="9999"
      dataStoreName="myDataStoreName" queueNames="myTpipeName"/>
 </activationSpec>
```

3. コールアウト IVP の <enterpriseApplication> エントリーを追加します。

```
<enterpriseApplication location="imsicocalloutivp.ear" type="ear">
      <classloader classProviderRef="IMS_adapter_id" />
   </enterpriseApplication>
```

IMS_adapter_id は、再度、<resourceAdapter> エントリーに指定された IMS TM リソース・アダプターの ID であるため、この例では、次のように、IMSTMRA となります。

```
<enterpriseApplication location="imsicocalloutivp.ear" type="ear">
      <classloader classProviderRef="IMSTMRA" />
   </enterpriseApplication>
```

4. 変更内容を保管します。

これで、必要な MDB フィーチャーおよび JCA アクティベーション・スペックを構成し、IMS TM リソース・アダプター・コールアウト IVP の EAR ファイルをデプロイしました。

ホスト・システム上で IMS コールアウト・アプリケーションを実行する準備ができました。

IMS ホスト・コールアウト IVP アプリケーションの実行

IMS ホスト・コールアウト IVP アプリケーションを実行すると、同期または非同期のコールアウト・メッセージを発行できます。

前提条件**:** サンプル・アプリケーションをターゲット Java EE アプリケーション・サーバーにデプロイする必要があります。

非同期コールアウト要求の場合:

1. IV_S227J ジョブを IV_S001T の説明に従って実行します。IV_S001T には、ホスト上の非同期 IMS コールアウト IVP アプリケーションに関する入門情報が記載されています。 次のサンプル出力は、「HELLO FROM IMS」という要求メッセージを示しています。

```
CALL=ISRT
        0100
SEGMENT =(0E00HELLO FROM IMS        )

COMP                          RET CODE=OK
ALTPCB SOURCE/DEST=IVPDTOR3 RET CODE=
```

サーバー・ログまたはコンソールに、次のメッセージが表示されます。このメッセージが存在することで、IMS TM リソース・アダプターのコールアウト IVP MDB がホスト IMS コールアウト IVP アプリケーションから非同期コールアウト要求を受け取ったことを確認できます。

```
SystemOut O Asynchronous callout request from IMS: HELLO FROM IMS
```

同期コールアウト要求の場合:

2. IV_S228J ジョブを、ホスト上の IMS コールアウト IVP アプリケーションにある説明に従って実行します。 次の出力は、要求メッセージ「HELLO FROM IMS」、および応答メッセージ「HELLO FROM WEBSPHERE MDB」を示しています。

```
CALL=ICAL SENDRECV IVPDTOR4 001000 00050 00050

          0003             0003             0000
AIBOALEN = 0002, AIBOAUSE = 0002, AIBRSFLD = 003Y

--------------------------------------------------------------------------

SEGMENT =(HELLO FROM IMS )

--------------------------------------------------------------------------

CALL=ICAL SEGMENT = (HELLO FROM WEBSPHERE MDB )
COMP RET CODE=OK
```

WebSphere Application Server のログまたはコンソールに以下のメッセージが表示されます。これらのメッセージが存在することで、IMS TM リソース・アダプターのコールアウト IVP メッセージ駆動型 Bean がホスト IMS コールアウト IVP アプリケーションから同期コールアウト要求を受け取り、ホスト IMS アプリケーションに応答メッセージを送信したことを確認できます。

```
SystemOut O Synchronous callout request from IMS: HELLO FROM IMS
SystemOut O Synchronous callout response from WAS MDB: HELLO FROM WEBSPHERE MDB
```

IMS TM リソース・アダプターのサービスおよび更新のインストール

サービスおよび更新は、IMS TM リソース・アダプターの新規バージョンとして入手できます。WebSphere Application Server 管理コンソールにある**RAR** の更新機能を使用して、新しいバージョンの IMS TM リソース・アダプターをインストールします。

前提条件**:**

1. IMS TM リソース・アダプター V15 リリース・ノートを確認します。入手可能な更新とフィックス、およびリソース・アダプターの他のサポート対象バージョンについてのリリース・ノートへのリンクのリストを確認できます。

2. (すべてのサポート対象プラットフォームで) 新規バージョンをダウンロードします。z/OS の場合は、更新を APAR または PTF として入手することもできます。

既存のすべての接続ファクトリーの削除と再構成を行わずに IMS TM リソース・アダプターのバージョンを更新するには、WebSphere Application Server の管理コンソールにある **RAR** の更新機能を利用し、それに合わせてクラス・パスを更新します。

新規バージョンをインストールするには、以下のようにします。

1. WebSphere Application Server 管理コンソールで、「**Resources**」「**Resource Adapters**」をクリックして展開します。
2. 「**Resource adapters**」をクリックします。
3. チェック・ボックスをクリックして、更新する IMS TM リソース・アダプターを選択し、「**Update RAR**」をクリックします。

図 *114.* 新規バージョンのリソース・アダプター用の *RAR* の更新

4. ステップ 1 で、「**Local file system**」または「**Remote file system**」のいずれかを選択し、「**Browse**」をクリックして新しい IMS TM リソース・アダプター の RAR ファイルがある場所までナビゲートします。 例えば、ローカル・ファイル・システムでは、RAR ファイルは *install_path*/IBM/IMS/ICO*xx*/V*xxxx*/JCA15/ims*xxxx*.rar にあります。
5. 「**Next**」をクリックします。
6. ステップ 2 で、既存のリソース・アダプターと新しいリソース・アダプターのバージョン番号を確認し、「**Next**」をクリックします。
7. ステップ 3 で、プロンプトに従って、新しいバージョンのリソース・アダプターで導入された新しいプロパティーを構成します。「**Next**」をクリックします。
8. 要約ステップで変更内容を確認し、「**Finish**」をクリックします。 「Resource adapter」ページに戻ります。
9. 更新したリソース・アダプターの名前をクリックします。
10. 「General Properties」の下にあるクラスパス・フィールドで、RAR ファイル名を更新します。 例えば、IMS TM リソース・アダプターのバージョン 14.1 にアップグレードする場合は、クラスパスを ${CONNECTOR_INSTALL_ROOT}/imsico1410.RAR に変更します。
11. 「**OK**」をクリックします。 「Resource adapter」ページに戻ります。
12. ページの上部にある「**Save**」をクリックします。

13. EAR ファイルがすでにデプロイされている場合には、同じ方法により、IMS TM リソース・アダプター IVPEAR ファイルを更新します。
14. オプションで、古い IMS TM リソース・アダプター RARファイルを削除します。
    - 分散プラットフォームの場合、*install_path*/IBM/IMS/ICO*xxx*/V*xxxx*にあるインストール・ディレクトリーを削除します。
    - z/OS の場合は、*install_path*/V*xxxx* にあるディレクトリーを削除します。

    install_path はターゲット・インストール・パスであり、V*xxxx* は、削除する IMS TM リソース・アダプターのバージョンです。

IMS TM リソース・アダプターの RAR ファイルが更新されます。これで、既存のすべての接続ファクトリーが新しいバージョンの RAR を使用するようになります。

関連情報:

WebSphere Application Server バージョン 8 の資料

リソース・ワークロード・ルーティングの構成

WebSphere Application Server のリソース・ワークロード・ルーティング機能を使用すると、データ・ソースおよび接続ファクトリーのフェイルオーバーと、事前定義された代替リソースからのフェイルバックが提供されます。

前提条件: IMS TM リソース・アダプターには、少なくとも 2 つの接続ファクトリーを作成しておく必要があります。接続ファクトリーの作成方法に関する手順については、 895 ページの『WebSphere Application Server での接続ファクトリーの作成』を参照してください。

WebSphere Application Server バージョン 8 以降ではリソース・ワークロード・ルーティング機能が提供されます。アプリケーションはこの機能を使用することで、代替リソースや構成情報を組み込まなくてもリソースの停止からリカバリーできます。

指定された障害しきい値またはデフォルトの障害しきい値に達したときに、自動的にデータ・ソースと接続ファクトリーをフェイルオーバーおよびフェイルバックできます。フェイルオーバーが行われると、アプリケーションは 1 次リソースの使用から代替リソースの使用に切り替えます。フェイルバックは、アプリケーションが代替リソースから 1 次リソースに切り替え直すときに行われます。

IMS TM リソース・アダプター用に代替リソースが構成されると、WebSphere Application Server は ping メッセージを IMS Connect に送信してそれが稼働中であることを確認したうえで、現在使用可能なリソースへの接続を再確立します。

リソース・ワークロード・ルーティング機能を有効にするには、1 次接続ファクトリーのカスタム接続プール・プロパティーとして alternateResourceJNDIName プロパティーを構成します。 alternateResourceJNDIName プロパティーの値は 2 次接続ファクトリーの JNDI 名に設定します。

1. WebSphere Application Server 管理コンソールで、「**Resources**」 > 「**Resource Adapters**」を展開します。「**Resource adapters**」をクリックします。
2. 「Additional Properties」セクションの下の「**Connection pool properties**」リンクをクリックします。
3. 「Connection pools」ページで、「**Connection pool custom properties**」をクリックします。
4. 「Custom properties」ページで、「**New**」をクリックして alternateResourceJNDIName プロパティーを追加します。
5. プロパティー名は alternateResourceJNDIName に設定する必要があります。値は、代替接続ファクトリーの JNDI 名です。

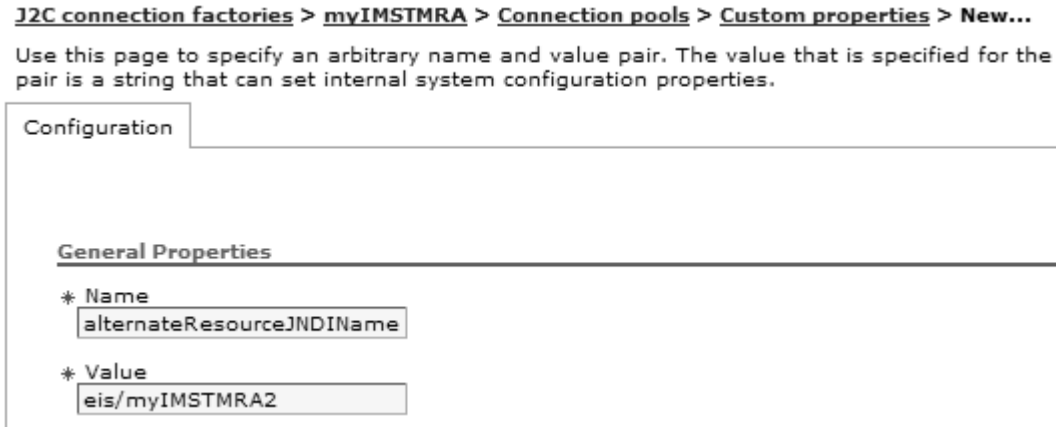


図 *115.* プロパティー作成時のカスタム・プロパティー・ページ。

6. 「**OK**」をクリックします。 代替リソースが構成されます。

IMS TM リソース・アダプターに対して代替接続ファクトリーを作成しました。

同じ手順を繰り返すと、failureThreshold など、他のカスタム・プロパティーを追加できます。構成可能なその他のプロパティーについては、WebSphere Application Server の情報を参照してください。

関連情報:

WebSphere Application Server (IBM Knowledge Center)

第 48 章 IMS TM リソース・アダプターとともに使用するアプリケーションの開発

IMS TM リソース・アダプターを使用して IMS トランザクション・マネージャー (IMS TM) と対話するには、Rational または WebSphere 統合開発環境 (IDE) を使用するか、または Java EE コネクター・アーキテクチャーの Common Client Interface (CCI) を使用して IDE の外部でアプリケーションをコーディングすることで、アプリケーションを開発します。

アプリケーションは、IMS TM リソース・アダプターを使用して IMS Transaction Manager で IMS トランザクションを実行します。IMS TM リソース・アダプターは、IMS OTMA によってサポートされる IMS コマンドの実行など、IMS TM との他のタイプの対話もサポートします。アプリケーションを開発する際には、IMS TM との対話のタイプを考慮してください。また、IMS TM リソース・アダプターは、Java EE コネクター・アーキテクチャーの Common Client Interface (CCI) API を実装します。このアプリケーション・プログラミング・インターフェースをご使用のアプリケーションで使用すると、IMS TM リソース・アダプターを介してエンタープライズ情報システム (EIS) (ここでは IMS TM) と通信することができます。

推奨事項**:** IDE を使用してアプリケーションを開発する場合、提供されたウィザードおよびファイル・インポーターを使用して、WebSphere Application Server で実行される Java EE アプリケーションを生成できます。IDE を使用してコードを生成する場合、Java アプリケーションが必要とするサービスの品質 (QOS) コードまたは CCI コードを書き込む必要はありません。代わりに、コードの作成に集中して、ビジネス・ロジックおよび関数を実装することができます。

関連情報:

965 ページの『IMS へのコマンド送信』

IMS Transaction Manager との対話

Java アプリケーションは、IMS Transaction Manager (IMS TM) と対話して、IMS トランザクションの実行、配信されなかった出力メッセージまたは非同期出力メッセージのリトリーブ、IMS コールアウト要求のリトリーブとその要求への応答、または IMS Open Transaction Manager Access (OTMA) でサポートされるいずれかの IMS コマンドの呼び出しを行います。

非会話型 IMS トランザクションを実行する Java アプリケーションは、IMS TM リソース・アダプターを使用する最も一般的なタイプの Java アプリケーションです。非会話型トランザクションを実行する場合、Java アプリケーションは IMS Connect を介して IMS TM と対話し、トランザクションの入力メッセージを IMS TM に渡し、IMS TM から返される出力メッセージを受け入れます。

Java EE コネクター・アーキテクチャー (JCA) では、この対話は Interaction オブジェクトのインスタンスを使用して行われます。Java アプリケーションは、

Interaction クラスの execute メソッドを呼び出して IMS と対話し、IMS トランザクションの入力メッセージと IMSInteractionSpec オブジェクトを渡します。com.ibm.connector2.ims.ico パッケージ内の IMSInteractionSpec クラスは、その対話のプロパティーを記述します。

IMSInteractionSpec クラスのプロパティーは、Java アプリケーションが IMS TM との間で行う対話のタイプを決定します。対話はトランザクション・パイプ (TPIPE) で行われます。TPIPE は OTMA クライアント (IMS Connect など) とサーバー (IMS OTMA) の間の論理接続です。

Java アプリケーションの IMSInteractionSpec は、Interaction オブジェクトの execute メソッドを呼び出すように構成できます。これには、以下のように複数の方法があります。

- 統合開発環境 (IDE) のウィザードを使用して、Java アプリケーションの Common Client Interface (CCI) コードを生成します。この場合、IMSInteractionSpec クラスのプロパティーの値がウィザードに渡され、ユーザーのために CCI コードが生成されます。

  推奨事項: この方法ではユーザーのためにスケルトン・コードが生成されるため、アプリケーションを開発する場合はこの方法を使用してください。
- IMS TM リソース・アダプターの Common Client Interface を直接呼び出す Java アプリケーションをコーディングします。set メソッドを使用して IMSInteractionSpec オブジェクトを構成し、Interaction オブジェクトの execute メソッドを呼び出します。
- IMSInteractionSpec のプロパティーが実行時に動的に渡される、コード生成 IDE モデルの 1 つのバリエーションを使用します。通常この方法は、IMSInteractionSpec (または IMSConnectionSpec) オブジェクトのプロパティーをデータとして公開する、と呼ばれます。

関連資料:

1063 ページの『IMS 相互作用仕様プロパティー』

関連情報:

1018 ページの『サンプルおよびチュートリアル』

プログラミング・モデル

使用するプログラミング・モデルは、Java アプリケーションが対話する IMS トランザクションのタイプによって異なります。

IMS TM リソース・アダプターは、以下のプログラミング・モデルをサポートします。

送信/受信プログラミング・モデル

送信/受信プログラミング・モデルを使用して、IMS 応答モードのトランザクションを実行します。

IMS でトランザクションを実行するために、Java アプリケーションは SYNC_SEND_RECEIVE 対話を実行します。ご使用のアプリケーションで、Interaction オブジェクトの execute メソッドによって使用されている IMSInteractionSpec オブジェクトに以下の値を指定してください。

- interactionVerb プロパティーに値 SYNC_SEND_RECEIVE
- commitMode プロパティーに値 0 または 1

ただし、SYNC_SEND_RECEIVE 対話の処理は、共用可能永続ソケット接続と専用永続ソケット接続では異なります。ソケット接続のタイプにより、トランザクションの通常の処理時、あるいはエラーまたは実行タイムアウトの発生時の処理モデルは異なります。

応答を予期している送信後コミット (CM1) アプリケーションを、応答を予期していないコミット後送信 (CM0) アプリケーションに変換する場合は、IMSInteractionSpec の CM0Response プロパティーを true に設定します。CM0 トランザクションにこのプロパティーが設定されているときに、IMS アプリケーションが IOPCB に応答しないか、別のトランザクションへのメッセージ通信を完了しない場合、トランザクションの応答モードに関係なく、IMS OTMA はクライアントに対して DFS2082 メッセージを発行します。

共用可能永続ソケットの処理モデル:

共用可能永続ソケット接続は、コミット・モード 1 対話およびコミット・モード 0 の対話で使用できる接続です。以下のシナリオは、通常処理、エラー処理、および実行タイムアウトにおける、共用可能永続ソケットでの SYNC_SEND_RECEIVE 対話を表しています。

通常処理のシナリオ

IMS TM リソース・アダプターは、アプリケーション・サーバーによって、接続プールから使用可能な接続を取得するか、または新規接続を作成します。 IMS TM リソース・アダプターは、新規接続の初期設定の一環として、その接続のクライアント ID を生成します。生成されたクライアント ID は、ソケット接続を識別し、コミット・モード 0 の対話の場合はさらに、TPIPE と関連する OTMA 非同期保留キューを識別します。

IMS TM リソース・アダプターは、ソケットが接続に関連付けられていることを確認し、そのソケットを使用して、IMS Connect に要求を入力データとともに送信します。IMS Connect はその後、メッセージを IMS に送信します。この場合、IMS がトランザクションを実行し、出力メッセージを返します。

コミット・モード 0 の対話の場合、IMS TM リソース・アダプターは、出力メッセージを受け取ると、ACK メッセージを IMS に送信します。ACK メッセージは、出力を IMS キューから破棄するように IMS に通知します。クライアント・アプリケーションが接続を閉じるか終了すると、接続は接続プールに返され、他のコミット・モード 0 またはコミット・モード 1 の対話によって再使用できるようになります。

エラー処理のシナリオ

エラーが発生した場合はすべて、リソース例外がクライアント・アプリケーションにスローされます。さらに、一部のエラーでは、ソケットが IMS Connect から切断されます。コミット・モード 0 の対話では、例外は出力メッセージをクライアント・アプリケーションに配信できないことを意味します。ただし、SYNC_SEND_RECEIVE 対話で、配信されなかった出力が特定の宛先に転送される

ことが指定されている場合、例外の後に、共用可能永続ソケット接続上のコミット・モード 0 の対話の配信されなかった出力メッセージをリトリーブできます。配信されなかった出力メッセージが特定の宛先に転送されるようにするには、SYNC_SEND_RECEIVE 対話で渡される IMSInteractionSpec オブジェクトに以下の追加プロパティーが指定されている必要があります。

- purgeAsyncOutput プロパティーを false に設定し、配信されなかった出力がパージされないようにします。
- reRoute プロパティーを true に設定し、RouteName プロパティーに転送宛先を指定します。

専用永続ソケット接続では、配信されなかった出力を転送宛先からリトリーブするために、別のクライアント・アプリケーションが SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT または SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT 対話を発行します。そのクライアント・アプリケーションは、対話のクライアント ID として転送宛先を指定します。

あるいは、共用可能永続ソケット接続では、配信されなかった出力を転送宛先からリトリーブするために、別のクライアント・アプリケーションが代替クライアント ID を指定して、SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT または SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT 対話を発行します。代替クライアント ID を使用して、クライアント・アプリケーションは任意の TPIPE から、配信されなかった非同期出力メッセージをリトリーブできます。

purgeAsyncOutput プロパティーのデフォルト値は true です。purgeAsyncOutput プロパティーの値が true の場合、以下の出力メッセージがパージされます。

- 1 次 IMS アプリケーション・プログラムによって入出力プログラム連絡ブロック (入出力 PCB) に挿入された、配信されなかった出力メッセージ
- プログラム間通信によって呼び出された 2 次 IMS アプリケーション・プログラムによって入出力 PCB に挿入された出力メッセージ

purgeAsyncOutput プロパティーが false に設定されている場合は、転送宛先を指定する必要があります。

実行タイムアウトのシナリオ

実行タイムアウトが発生すると、ソケット接続は開いたままですが、出力メッセージがクライアント・アプリケーションに配信されません。ただし、実行タイムアウト例外の後に、共用可能永続ソケット接続上のコミット・モード 0 の対話の配信されなかった出力メッセージは、以下の 2 つの方法のいずれかでリトリーブできます。

- SYNC_SEND_RECEIVE 対話を発行したものと同じクライアント・アプリケーションが、SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT または SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT 対話を発行できます。
- 配信されなかった出力メッセージを、エラー処理のシナリオの説明に従って、特定の宛先に転送できます。

クライアント・アプリケーションが接続を閉じるか終了すると、接続は接続プールに返され、他のコミット・モード 0 またはコミット・モード 1 の対話によって再使用できるようになります。

関連概念:

959 ページの『共用可能永続ソケット』

専用永続ソケットの処理モデル:

専用永続ソケット接続は、コミット・モード 0 の対話でのみ使用できる接続です。

以下のシナリオは、通常処理、エラー処理、および実行タイムアウトにおける、専用永続ソケットでのコミット・モード 0 の SYNC_SEND_RECEIVE 対話を表しています。

通常処理のシナリオ

クライアント・アプリケーションによってコミット・モード 0 の SYNC_SEND_RECEIVE 対話が実行される場合、アプリケーション・サーバーは、ユーザー指定のクライアント ID を含む既存接続を返すか、またはユーザー指定のクライアント ID を使用して新規接続を作成します。ユーザー指定のクライアント ID は、ソケット接続と、OTMA 非同期保留キューに関連付けられた TPIPE を識別します。

IMS TM リソース・アダプターは、ソケットが接続に関連付けられていることを確認し、そのソケットを使用して、IMS Connect に要求を入力データとともに送信します。IMS Connect はその後、メッセージを IMS に送信します。この場合、IMS がトランザクションを実行し、出力メッセージを返します。 IMS TM リソース・アダプターは、出力メッセージを受け取ると、ACK を IMS に送信します。ACK は、出力を IMS キューから破棄するように IMS に通知します。接続が閉じられるかアプリケーションが終了すると、接続が接続プールに返され、同じユーザー指定のクライアント ID を使用してコミット・モード 0 の対話を実行している別のアプリケーションがそれを再使用できるようになります。

エラー処理のシナリオ

エラーが発生した場合はすべて、リソース例外がクライアント・アプリケーションにスローされます。さらに、一部のエラーでは、ソケットが IMS Connect から切断されます。コミット・モード 0 の対話では、例外は出力メッセージをクライアント・アプリケーションに配信できないことを意味します。配信されなかった出力は、ユーザー指定のクライアント ID に関連付けられた TPIPE のキューに入れられます。

プロパティー purgeAsyncOutput および reRoute は、専用永続ソケットには適用されません。専用永続ソケットでは、配信されなかった出力メッセージはパージも転送もできません。

実行タイムアウトのシナリオ

実行タイムアウトが発生すると、ソケットは開いたままになり、コミット・モード 0 の対話の出力は、後でリトリーブできるようにユーザー指定のクライアント ID に関連付けられた TPIPE のキューに入れられます。接続が閉じられるかアプリケー

ションが終了すると、IMSManagedConnection オブジェクトが接続プールに返され、同じユーザー指定のクライアント ID を使用してコミット・モード 0 の対話を実行している別のアプリケーションがそれを再使用できるようになります。

関連概念:

956 ページの『専用永続ソケット』

送信専用プログラミング・モデル

送信専用プログラミング・モデルを使用して、IMS Transaction Manager (IMS TM) で IMS 非応答モードのトランザクションを実行します。

IMS TM で非応答モードのトランザクションを実行するために、Java アプリケーションは SYNC_SEND 対話を実行します。 SYNC_SEND 対話では、IMS TM リソース・アダプターは IMS Connect を介して IMS に要求を送信しますが、IMS からの応答は予期していません。IMS TM リソース・アダプターが IMS との送信専用の対話を実行するので、SYNC_SEND 対話は通常、非応答モードのトランザクションで使用されます。

SYNC_SEND 対話を使用してトランザクションを実行するには、アプリケーションにより、interactionVerb プロパティーに値 SYNC_SEND が指定され、executeメソッドによって使用される IMSInteractionSpec オブジェクトの commitMode プロパティーに値 0 が指定される必要があります。 SYNC_SEND 対話の処理は、使用される永続ソケットのタイプ (共用可能または専用) および実行される IMS トランザクションのタイプによって異なります。

重要**:** IMSInteractionSpec プロパティー purgeAsycOutput、reRoute、および reRouteName は SYNC_SEND 対話には適用されないため、IMS TM リソース・アダプターによって無視されます。

共用可能永続ソケットの処理モデル

以下のシナリオは、さまざまなタイプのトランザクションでの、共用可能永続ソケット接続上の SYNC_SEND 対話を表しています。

- 非応答モード・トランザクション

  IMS に対して非応答モードとして定義されているトランザクションに関連付けられた IMS アプリケーション・プログラムは、通常、出力メッセージを入出力 PCB に挿入しません。したがって、出力メッセージは作成されず、TPIPE では何もキューに入れられません。
- 応答モード・トランザクション

  IMS に対して応答モードとして定義されているトランザクションに関連付けられた IMS アプリケーション・プログラムは、通常、出力メッセージを入出力 PCB に挿入します。 IMS TM リソース・アダプターが SYNC_SEND 対話からの応答を予期していないため、出力メッセージ (挿入された場合) は生成されたクライアント ID の名前を使用して TPIPE でキューに入れられます。ただし、対話 SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT または SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT が同じアプリケーション内、および同じ接続上で SYNC_SEND 対話の後に実行されている場合は、これらを使用して応答をリトリーブできます。

- 代替 PCB にメッセージを挿入する IMS アプリケーション・プログラムを起動する非応答モードまたは応答モードのトランザクション

  代替 PCB に挿入されるメッセージは、専用永続ソケット接続上で対話を実行してリトリーブできます。このメッセージを挿入するには、トピック『非同期出力プログラミング・モデル』に記載されている専用永続ソケット接続での非同期出力のリトリーブに関する説明を参照してください。

専用永続ソケットの処理モデル

以下のシナリオは、さまざまなタイプのトランザクションでの、専用永続ソケット接続上の SYNC_SEND 対話を表しています。 SYNC_SEND 対話ではコミット・モード 0 が使用され、専用永続ソケット接続はコミット・モード 0 の対話用にのみ使用できます。

- 非応答モード・トランザクション

  IMS に対して非応答モードとして定義されているトランザクションに関連付けられた IMS アプリケーション・プログラムは、通常、出力メッセージを入出力 PCB に挿入しません。したがって、出力メッセージは作成されず、TPIPE では何もキューに入れられません。
- 応答モード・トランザクション

  IMS に対して応答モードとして定義されているトランザクションに関連付けられた IMS アプリケーション・プログラムは、通常、出力メッセージを入出力 PCB に挿入します。 IMS TM リソース・アダプターが SYNC_SEND 対話からの応答を予期していないため、出力メッセージ (挿入された場合) はその対話のクライアント ID に指定されているの名前を使用して TPIPE でキューに入れられます。このタイプの TPIPE でキューに入れられるメッセージは、SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT または SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT 対話を発行してリトリーブできます。TPIPE 名は、SYNC_SEND 対話に指定されているクライアント ID です。クライアント ID は、専用永続ソケット接続を使用する対話に必要です。
- 代替 PCB への挿入を行う IMS アプリケーションを起動する非応答モードまたは応答モードのトランザクション

  代替 PCB に挿入されるメッセージは、専用永続ソケット接続上で対話を実行してリトリーブできます。詳しくは、トピック『非同期出力プログラミング・モデル』を参照してください。

関連概念:

『非同期出力プログラミング・モデル』

非同期出力プログラミング・モデル

このプログラミング・モデルを使用して、ユーザーは IMS によってキューに入れられた出力をリトリーブできます。

アプリケーションは、過去のいずれかの時点でキューに入れられた出力をリトリーブしなければならない場合があります。このような出力を、非同期出力 と呼びます。非同期出力はいくつかの状況が原因で発生します。具体的には、コミット・モ

ード 0 の対話からの配信されなかった出力、プログラム間通信からの出力、代替 PCB への ISRT 呼び出しからの出力、複数の出力メッセージを返す IMS アプリケーションからの出力といったものです。

非同期出力をリトリーブするために Java アプリケーションは SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT 対話または SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT 対話を実行します。アプリケーションでは、interactionVerb プロパティーにこの値を指定し、さらに、対話の execute メソッドで使用される IMSInteractionSpec オブジェクトの commitMode プロパティーには値 0 を指定する必要があります。

非同期出力のリトリーブに使用できるソケット接続には、共用可能永続ソケットと専用永続ソケットの 2 つのタイプがあります。非同期出力メッセージのリトリーブ方法は、使用されているソケット接続のタイプによって異なります。非同期出力のリトリーブに使用できる interactionVerb プロパティーの値は、以下のとおりです。

- SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT
- SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT

ヒント: SYNC_RECEIVE_ASYNCOUTPUT は IMS TM Resource Adapter バージョン 10 では非推奨になったため、SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT に置き換えられました。

SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT が SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT と違う点は、IMS Connect が IMS OTMA 非同期保留キューの出力の有無を確認する方法です。

- SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT 対話では、リトリーブ要求が行われた時点で IMS OTMA 非同期保留キューに非同期出力が入っていない場合、IMS Connect は、クライアント・アプリケーションによって指定された実行タイムアウト値が過ぎると直ちに実行タイムアウト通知を返します。

  推奨事項: SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT 対話では、最小実行タイムアウト値である 10 を指定してください。
- SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT 対話の場合、実行タイムアウト値が過ぎたときに保留キューに非同期出力が入っていない場合、IMS Connect は実行タイムアウト・エラーを返します。該当しない場合、IMS Connect は、最初の出力メッセージがキューに入れられたときにそのメッセージを返します。

  推奨事項: SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT 対話では、最小値ではなく適切な実行タイムアウト値を選択してください。

非同期出力プログラミング・モデルではコミット・モード 0 を使用する必要があり、このモデルは共用可能永続ソケット接続と専用永続ソケット接続の両方で使用できます。さらに、IMSInteractionSpec プロパティー purgeAsycOutput、reRoute、および reRouteName は適用されず、interactionVerb プロパティーがこのいずれかの値に設定されても無視されます。interactionVerb プロパティーは、専用永続ソケット接続と共用可能永続ソケット接続ではそれぞれ異なる方法で呼び出されます。

関連概念:

1025 ページの『出力メッセージを含む Java 例外』

989 ページの『IMS 保留キューからのメッセージのリトリーブの保護』

専用永続ソケット接続での非同期出力のリトリーブ:

専用永続ソケット接続で非同期出力をリトリーブするには、コミット・モード 0 の対話を使用して、クライアント ID を指定します。

1. IMSInteractionSpec の interactionVerb プロパティーを SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT または SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT に設定して、コミット・モード 0 の対話を実行します。
2. clientID プロパティーに値を指定します。clientID プロパティーは、非同期出力がリトリーブされる TPIPE を判別するため、必須です。
   - 専用永続ソケット上でコミット・モード 0 対話から出力メッセージをリトリーブするには、SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT 対話または SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT 対話に指定されたクライアント ID が、元のコミット・モード 0 対話用に指定された値と一致する必要があります。
   - 代替 PCB に送信された出力メッセージをリトリーブするには、SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT 対話または SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT 対話に指定されたクライアント ID が、代替 PCB の名前と一致する必要があります。
   - reRouteName 宛先に転送された出力メッセージをリトリーブするには、SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT 対話または SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT 対話に指定されたクライアント ID が、reRouteName プロパティーによって指定された宛先に設定されている必要があります。

関連概念:

956 ページの『専用永続ソケット』

代替クライアント **ID** を使用した共用可能永続ソケット接続での非同期出力のリトリーブ:

クライアント・アプリケーションは代替クライアント ID を使用して、任意の OTMA TPIPE から転送されるか、または転送されない非同期出力メッセージをリトリーブすることができます。

この代替クライアント ID は、IMSInteractionSpec オブジェクトの altClientID プロパティーによって指定されます。代替クライアント ID は、非同期出力メッセージを持つ OTMA TPIPE の名前と一致します。クライアント・アプリケーションは、同じ対話で転送名 (reRouteName プロパティー) と代替クライアント ID (altClientID プロパティー) を指定することはできません。

代替クライアント ID は、クライアント・アプリケーションが以下のすべての要件を満たす場合にサポートされます。

1. TCP/IP 接続を共用可能永続ソケットで使用する。

2. 非同期出力メッセージ (resume tpipe) の SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT および SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT をリトリーブするために interactionVerb プロパティーに有効な値を使用する。
3. commitMode プロパティー値を 0 に設定する。

関連概念:

959 ページの『共用可能永続ソケット』

代替クライアント **ID** を使用しない共用可能永続ソケット接続での非同期出力のリトリーブ**:**

非同期出力のリトリーブは、代替クライアント ID が指定されていない場合は直前の対話と同じ接続で行う必要があります。

代替クライアント ID を使用しない場合、クライアント・アプリケーションは、SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT 対話または SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT 対話を実行することにより、共用可能永続ソケット接続での対話から転送されない非同期出力メッセージのみをリトリーブできます。この対話は、同じアプリケーションからの対話が、キューに入れられている非同期出力につながったのと同じ共用可能永続ソケット接続で行う必要があります。

以下の表は、異なるプロパティー値が指定された場合の非同期出力メッセージ (resume tpipe) 対話のリトリーブの動作を示しています。

表 *114.* 異なる *IMSConnectionSpec* プロパティー値を持つ *resume tpipe* 対話の動作

対話	ソケット	クライアント **ID**	転送名	代替クライアント **ID**	コメント
resume tpipe (代替クライアント ID を指定しない)	共用可能	NULL	NULL	NULL	非同期出力は、前の対話と同じ接続でリトリーブする必要があります。そうでない場合、非同期出力はリトリーブできません。リトリーブできなかった場合、メッセージは失われます。
resume tpipe (代替クライアント ID を指定しない)	共用可能	NULL	*myRR*	NULL	非同期メッセージは、前の対話と同じ接続でリトリーブする必要があります。リトリーブできなかった場合、メッセージは *myRR* に転送されます。
resume tpipe (代替クライアント ID を指定)	共用可能	NULL	N/A	*myTpipe*	非同期メッセージは tpipe *myTpipe* からリトリーブされます。reRouteName プロパティーと altClientID プロパティーは、相互に排他的であり、同時に使用できません。
RESUME TPIPE	専用	*myCID*	N/A	N/A	非同期メッセージは、クライアント ID *myCID* を使用してリトリーブする必要があります。

共用可能永続ソケット接続では、
SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT 対話または
SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT 対話のコミット・モードは、その対話の IMSInteractionSpec オブジェクトに設定された値に関係なく、IMS TM リソース・アダプターによって自動的に 0 に設定されます。この動作は、専用永続ソケット接続での SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT 対話または SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT 対話の動作とは異なります。専用ソケット接続の場合、コミット・モードは明示的に 0 に設定する必要があります。

関連概念:

959 ページの『共用可能永続ソケット』

出力メッセージ・カウントの表示と解釈**:**

IMS Connect コマンドを使用すると、非同期出力メッセージの適切なリトリーブをモニターしたり、そのトラブルシューティングを行ったりする場合に、出力メッセージ・カウントを表示することができます。

コミット・モード 0 の対話では、TPIPE 名は、その対話に使用されるクライアント ID です。コミット・モード 0 の対話では、TPIPE と関連付けられている IMS OTMA 非同期保留キューは、クライアント ID と同じ名前を持ちます。

コミット・モード 1 の対話では、TPIPE 名は、その対話に使用される IMS Connect ポート番号です。各ポートには、そのポートでコミット・モード 1 の対話を実行するすべてのクライアントに使用される TPIPE があります。

1. IMS TM リソース・アダプターに送信される出力メッセージに加え、代替プログラム連絡ブロック (代替 PCB) に挿入されるメッセージのカウントを表示するために、IMS Connect コマンド /DISPLAY TMEMBER *IMSConnect_Name* TPIPE ALL を使用します。

   以下の出力例は、/DISPLAY TMEMBER HWS1 TPIPE ALL コマンドからのものです。コマンド出力の TPIPE およびカウントのタイプについても記述されています。

```
DFS000I     MEMBER/TPIPE       ENQCT   DEQCT     QCT  STATUS     IMS1
DFS000I     HWS1                                                 IMS1
DFS000I    -9999                   0       0       0             IMS1
DFS000I    -HWSMIJRC               2       2       0             IMS1
DFS000I    -CLIENT01               3       2       1             IMS1
DFS000I    -ALTPCB1                2       1       1             IMS1
DFS000I    -HWS$DEF                1       0       1             IMS1
DFS000I    -RRNAME                 1       0       1             IMS1
```

2. コマンド出力を解釈するために、どの TPIPE が対象のキューであるかを判別します。対応する QCT 列にメッセージ・カウントが入っています。 TPIPE 名は、対話と接続のタイプによって決まります。
   - 共用可能永続ソケットでのコミット・モード **1** の対話の場合:
     - TPIPE 名は、対話に使用されるポート番号です。このサンプルでは、TPIPE 9999 は、このキューが共用可能永続ソケットでのコミット・モード 1 の対話のためのものであることを示しています。

- エンキュー・カウント (ENQCT) とデキュー・カウント (DEQCT) は同じであり、キュー・カウント (QCT) は 0 です。これは、配信されなかった出力メッセージはコミット・モード 1 のトランザクションに対してリカバリーできないためです。

- 共用可能永続ソケットでのコミット・モード **0** の対話の場合:
  - TPIPE 名は、IMS TM リソース・アダプターによって生成され、HWS という接頭部を持ちます。この例では、IMS TM リソース・アダプターにより TPIPE 名 HWSMIJRC が生成されます。
  - すべてのメッセージが IMS TM リソース・アダプターに配信される場合、エンキュー・カウント (ENQCT) とデキュー・カウント (DEQCT) は同じであり、キュー・カウント (QCT) は 0 です。
  - 以下の条件が両方とも満たされた場合、エンキュー・カウント (ENQCT) とデキュー・カウント (DEQCT) は同じであり、キュー・カウント (QCT) は 0 です。配信されなかった出力メッセージはすべて廃棄されます。
    - 出力メッセージは、SYNC_SEND_RECEIVE 対話では IMS TM リソース・アダプターに配信されない
    - reRoute プロパティーのデフォルト (false) および purgeAsyncOutput プロパティーのデフォルト (true) が使用される
  - 以下の条件が両方とも満たされた場合、エンキュー・カウント (ENQCT) はデキュー・カウント (DEQCT) よりも大きく、キュー・カウント (QCT) は、IMS TM リソース・アダプターに配信されなかったメッセージの数です。
    - 出力メッセージは、SYNC_SEND_RECEIVE 対話では IMS TM リソース・アダプターに配信されない
    - reRoute プロパティーは true に設定され、purgeAsyncOutput プロパティーは false に設定されている

    TPIPE 名は、reRouteName プロパティーに対して指定された値です。例えば、RRNAME、またはデフォルト値 (例えば HWS$DEF) です。
  - SYNC_SEND 対話の場合、出力は予期されないため、配信されなかった出力は適用されません。
    SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT 対話および SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT 対話が正しく実行されない場合、キュー・カウントは変わりません。

- 専用永続ソケットでのコミット・モード **0** の対話の場合:
  - 通常、TPIPE 名は、Java アプリケーションによって提供され、HWS という接頭部は付けられません。例えば、CLIENT01 です。ただし、HWS$DEF というTPIPE 名が付いていることがあります。これは、reRouteName プロパティーのデフォルト値です。
  - エンキュー・カウント (ENQCT) とデキュー・カウント (DEQCT) は同じであり、すべてのメッセージが IMS TM リソース・アダプターに配信され、配信されなかった出力メッセージが共用可能永続ソケット接続での対話から転送されなかった場合、キュー・カウント (QCT) は 0 です。
  - 出力メッセージが IMS TM リソース・アダプターに配信されないか、または共用可能永続ソケット接続での対話から転送される場合、エンキュ

ー・カウント (ENQCT) はデキュー・カウント (DEQCT) より大きく、キュー・カウント (QCT) は、配信されなかったメッセージの数です。TPIPE 名は、ユーザーが指定したクライアント ID 名です。例えば、CLIENT01 です。

- 代替 **PCB** に挿入される出力メッセージの場合:
  - TPIPE 名は、代替 PCB の名前 (ALTPCB1 など) になります。

コールアウト・プログラミング・モデル

このプログラミング・モデルを使用して、IMS アプリケーションからアウトバウンド・メッセージを送信し、メッセージ駆動型 Bean (MDB)、Enterprise JavaBeans (EJB) コンポーネント、または Web サービスに対してサービスまたはデータを要求し、オプションで応答データを受信します。

IMS コールアウト・プログラミング・モデルでは、IMS はクライアントとして機能して、MDB、EJB コンポーネント、または Web サービスにあるビジネス・データまたはビジネス・ロジックと対話します。 IMS アプリケーションは、コールアウト要求を発行して外部 Java アプリケーションを呼び出します。 WebSphere Application Server の観点では逆に、IMS アプリケーションからのこれらのメッセージはインバウンド・メッセージです。

外部 Java アプリケーションからの応答が予期されているかどうか、および同じトランザクション内で IMS アプリケーションを開始することによって応答が返されることが予期されているかどうかに応じて、異なるプログラミング・モデルを採用できます。

同期コールアウト

同期コールアウト要求は、外部 Java アプリケーションまたは Web サービスからの応答が同じトランザクション内で同じ IMS アプリケーションに返されることが予期されている要求です。

IMS アプリケーションが IMS DL/I ICAL 呼び出しを使用して同期コールアウト要求を発行すると、その要求は OTMA トランザクション・パイプ (TPIPE) を使用して保持されます。従属領域内の IMS アプリケーションはスケジュールされた状態のまま、応答を待機します。外部アプリケーションが TPIPE から要求メッセージをプルすると、その外部アプリケーションから応答が返されるまで TPIPE は待機状況になります。

メッセージ駆動型 Bean (MDB) の場合は、Java EE コネクター・アーキテクチャー 1.5 インバウンド仕様、およびさまざまな Rational または WebSphere 統合開発環境 (IDE) のツール・サポートを利用できます。IMS TM リソース・アダプターは IMS Connect からのコールアウト・メッセージを listen し、要求とその応答の相関を処理します。

非 MDB アプリケーションの場合、IMS Connect にポーリングを行って保留キューにコールアウト要求がないか確認するのはアプリケーションの役目です。また、そのアプリケーションは、IMSInteractionSpec クラスのプロパティーとして渡される相関関係子トークンの処理も行う必要があります。

コールアウト要求と応答が同じトランザクション・インスタンス内にあることの主な利点は、要求を発行するためのプログラミング・ロジックと応答を処理するためのプログラミング・ロジックを同じ IMS アプリケーションに含めることができる点です。ただし、このアプリケーションでは IMS アプリケーションが応答を待機する必要があるため、従属領域がブロックされます。

非同期コールアウト

非同期コールアウト要求は、応答を予期していない要求、または応答が別のトランザクションに返されることが予期されている要求です。

IMS アプリケーションが ISRT ALPCB 呼び出しを使用して非同期コールアウト要求を発行すると、その要求は、OTMA トランザクション・パイプ (TPIPE) または代替宛先 (出口ルーチンでコーディングされた宛先) を使用して保持されます。外部アプリケーションが保留キューから要求メッセージをプルすると、IMS アプリケーションは IMS 従属領域内でスケジュールされた状態でなくなるため、その従属領域は解放されます。

応答が予期されている場合、応答メッセージは新しい IMS トランザクション要求として、同じ IMS アプリケーションまたは別の IMS アプリケーション (アプリケーション設計によります) に出力データとともに返されます。

応答が非同期的に返されるようにする主な利点は、従属領域がブロックされ、そのために長期間 IMS リソースがロックされる事態を回避できる点です。ただし、このアプローチでは、応答の処理が必要な場合、IMS アプリケーションが別のトランザクション・インスタンスで出力を処理できる設計になっていることが求められます。また、応答を要求と相関させることもアプリケーションの役目です。

関連概念:

996 ページの『実行タイムアウト』

989 ページの『IMS 保留キューからのメッセージのリトリーブの保護』

関連資料:

1024 ページの『コールアウト要求での問題の診断』

OTMA 宛先記述子:

ユーザーは OTMA 宛先記述子を使用することにより、アセンブラーの経路指定出口をコーディングすることなく、IMS アプリケーションに対して、IMS TM リソース・アダプターにアクセス可能なサービスにコールアウト要求を経路指定するよう指示することができます。

OTMA 宛先記述子を使用することにより、IMS コールアウト要求メッセージの経路指定先の IMS Connect 宛先 (トランザクション・パイプ、つまり TPIPE) を定義できます。特殊な経路指定が必要ない場合、コールアウト要求の TPIPE 名を直接、IMS アプリケーションの ICAL 呼び出し (同期コールアウトの場合) または ISRT ALTPCB 呼び出し (非同期コールアウトの場合) で指定できます。

OTMA 宛先記述子を使用するには、IMS システム・プログラマーが DFSYDTx PROCLIB メンバーでその記述子を構成する必要があります。次の例は、HWS1 というターゲット・メンバーおよび HWS1TP01 という TPIPE にメッセージを経路指定する、IMSTMRA という宛先用の記述子を表しています。

```
D IMSTMRA TYPE=IMSCON TMEMBER=HWS1 TPIPE=HWS1TP01
```

OTMA 宛先記述子は、同期コールアウト要求の経路指定には必要ですが、非同期コールアウト要求の経路指定には不要です。ただし、OTMA 出口ルーチンのアプローチより記述子のアプローチの方が使用しやすいため、非同期コールアウト要求には OTMA 宛先記述子の使用が推奨されます。

詳しくは、「IMS コミュニケーションおよびコネクション」のトピック『OTMA 記述子』、および「IMS システム定義」の DFSYDTx PROCLIB メンバーを参照してください。

コールアウト要求のリトリーブのセキュリティー:

IMS OTMA RESUME TPIPE によって提供されるセキュリティー機能では、RACF または OTMA RESUME TPIPE セキュリティー出口ルーチン (DFSYRTUX) のいずれか、あるいはこの両方を使用して、コールアウト・メッセージが許可なく使用されないように保護できます。

IMS コールアウト要求は、RESUME TPIPE 呼び出しを使用して、IMS Connect からリトリーブされます。セキュリティーが有効な場合、OTMA クライアントにメッセージが送信される前に、RESUME TPIPE 呼び出しを発行するユーザー ID に、RESUME TPIPE 呼び出しメッセージに含まれる TPIPE 名にアクセスするための権限が付与されている必要があります。セキュリティーが有効であり、RESUME TPIPE 呼び出しの発行時点で TPIPE が存在しない場合、その呼び出しは拒否されます。

メッセージ駆動型 Bean (MDB) の場合、IMS との通信用に Secure Sockets Layer (SSL) 認証がサポートされます。セキュリティー情報は、WebSphere Application Serverで構成されている J2C アクティベーション・スペック (IMSActivationSpec) で指定されます。

非 MDB アプリケーションの場合、権限があるユーザーのみが保留キューからコールアウト要求メッセージをリトリーブできるように IMS セキュリティーを構成している場合、オプションで Java アプリケーション内にユーザー ID を指定できます。このユーザー ID は、ご使用のアプリケーション内、またはそのアプリケーションによって使用されている接続ファクトリー内の接続仕様で指定する必要があります。

同期コールアウト・メッセージ・フロー:

同期コールアウト・メッセージは、TPIPE 保留キューに入れられ、外部アプリケーションまたはサービスによって取り出されるのを待機します。外部アプリケーションから応答メッセージが返されるまで、TPIPE は待ち状態になります。その後、次のコールアウト・メッセージを配信できます。

標準的なメッセージ・フローは、次のとおりです。

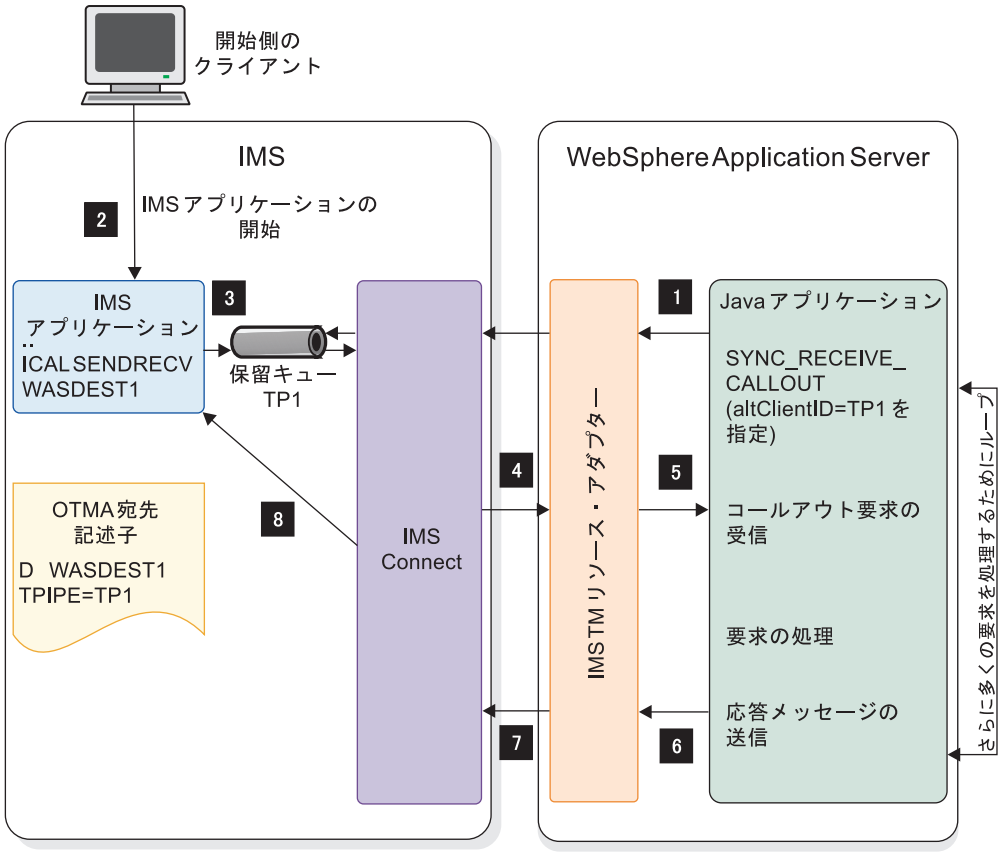


図 *116.* 同期コールアウト・メッセージ・フロー

1. WebSphere Application Server 内の Java EE アプリケーションが開始され、WebSphere Application Server が IMS TM リソース・アダプターを介して IMS Connect に接続されます。IMS TM リソース・アダプターは、それに対し、TPIPE に対して RESUME TPIPE 要求を発行し、IMS Connect からのコールアウト要求を待機します。
2. 開始側クライアント (例えば、端末や、IMS Connect または OTMA クライアント) が IMS アプリケーションを開始します。
3. IMS アプリケーションが IMS DL/I ICAL 呼び出しを発行し、コールアウト要求メッセージがキューに入れられる宛先 TPIPE 名を含む OTMA 宛先記述子を指定します。要求が TPIPE 保留キューに入れられると、その要求に相関トークンが付加されます。
4. 要求の時点で取得可能なコールアウト要求がない場合、IMS TM リソース・アダプターはブロックされ、Java アプリケーションは次に取得可能なコールアウト・メッセージを待機します。アプリケーション自体がコールアウト・メッセージの有無に関して IMS Connect へのポーリングを実施する EJB アプリケーションの場合、IMS TM リソース・アダプターはタイムアウトが発生するまで待機します。TPIPE でコールアウト要求が取得可能になるとすぐに、IMS Connect はそのコールアウト・メッセージを IMS TM リソース・アダプターに配信します。
5. IMS TM リソース・アダプターはコールアウト要求メッセージを受信し、そのコールアウト要求を Java アプリケーションに送信します。このアプリケーションがコールアウト要求を処理します。

6. Java アプリケーションは、相関トークンが付加された応答を IMS TM リソース・アダプターに送信します。
7. IMS TM リソース・アダプターはその応答を IMS Connect に転送します。

   肯定応答送信専用プロトコルが有効な場合、IMS TM Resource Adapter は、IMS が同期コールアウト応答メッセージを受信したときに IMS から肯定応答を受け取ります。
8. 相関トークンに基づき、応答が IMS からの開始側要求に相関されます。

非同期コールアウト・メッセージ・フロー:

非同期コールアウト・メッセージは、保留キューに入れられ、外部アプリケーションまたはサービスによって取り出されるのを待機します。応答データが予期されている場合、外部アプリケーションは、通常の IMS トランザクションを適切な IMS アプリケーションに出力データとともに発行します。

次の図に、一般的な非同期コールアウトのメッセージ・フローを示します。

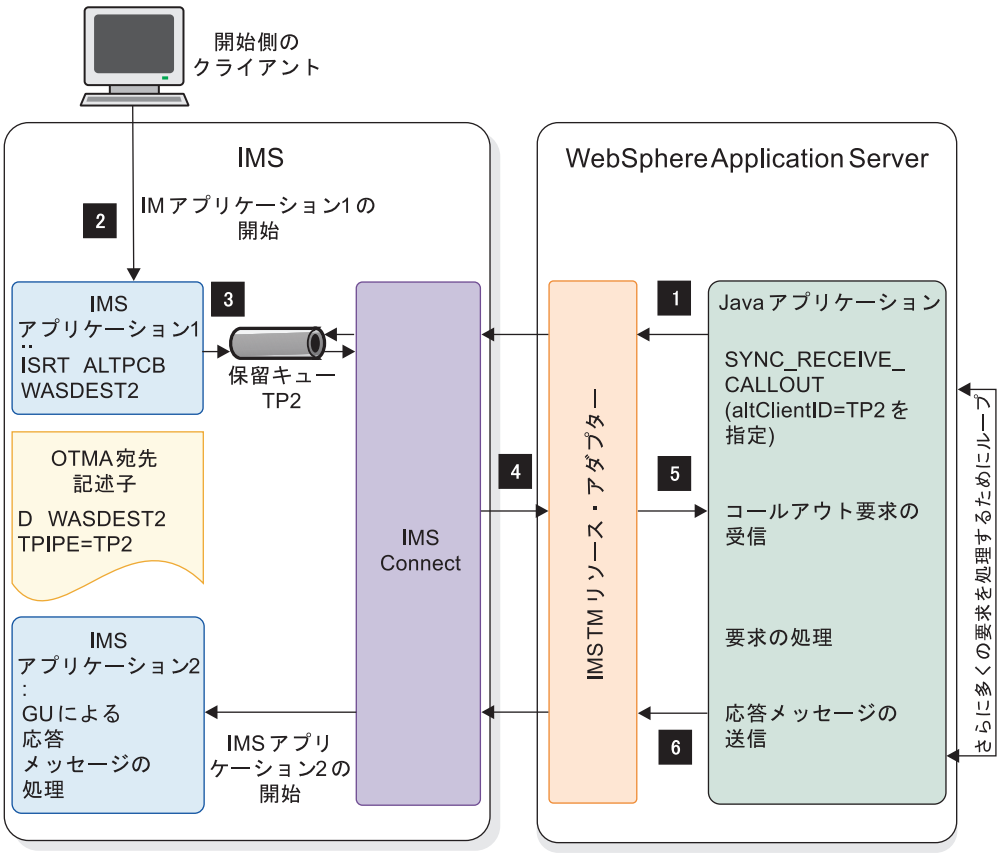


図 *117.* 非同期コールアウト・メッセージ・フロー

1. WebSphere Application Server で Java アプリケーションが開始し、IMS TM リソース・アダプターを介して IMS Connect への共用可能永続接続を取得します。このアプリケーションは SYNC_RECEIVE_CALLOUT 対話を発行し、代替クライアント ID の値として TPIPE 名を指定し、タイムアウト値を設定しま

す。 IMS TM リソース・アダプターは、それに対し、TPIPE に対して RESUME TPIPE 要求を発行し、IMS Connect からのコールアウト要求を待機します。

2. 開始側クライアント (例えば、端末や、IMS Connect または OTMA クライアント) が IMS アプリケーションを開始します。
3. その IMS アプリケーションは、宛先 TPIPE 名を含む OTMA 宛先記述子に ISRT ALTPCB 呼び出しを発行します。コールアウト要求メッセージはこの TPIPE でキューに入れられます。
4. SYNC_RECEIVE_CALLOUT 要求の時点で取得可能なコールアウト要求がない場合、IMS TM リソース・アダプターはブロックされ、Bean は次に取得可能なコールアウト・メッセージを待機するか、タイムアウトになるまで待機します。TPIPE でコールアウト要求が取得可能になると、IMS Connect はそのコールアウト・メッセージを IMS TM リソース・アダプターに配信します。
5. IMS TM リソース・アダプターはコールアウト要求メッセージを受信し、そのコールアウト要求を Bean に返します。 Bean がそのコールアウト要求を処理します。
6. Bean は IMS に返す応答データを受信すると、通常の IMS トランザクション要求を適切な IMS アプリケーションに出力データとともに発行します。

管理コールアウト・プログラミング・モデル:

管理コールアウト・プログラミング・モデル を使用すると、アウトバウンド・メッセージを IMS アプリケーションから送信して、外部のメッセージ駆動型 Bean (MDB) にサービスまたはデータを要求でき、応答データを同期または非同期で、同じトランザクション内の同じ IMS アプリケーションで受け取ることができます。

MDB は、ステートレスなサーバー・サイドの Java EE コンポーネントであり、エンタープライズ情報システム (EIS) から到着するインバウンド・メッセージを処理するための、J2EE コネクター・アーキテクチャー (JCA) 1.5 リソース・アダプター上の listener として構成できます。 MDB は、メッセージを取り込んで処理するためにメッセージ配信によって活動化できる EJB コンポーネントです。

JCA 1.5 仕様では、MDB は一般にメッセージ・エンドポイント、または単にエンドポイント と呼ばれます。MDB の主な利点として、メッセージ処理とビジネス処理がより明確に分離されていることや、他の着信メッセージによるビジネス処理の再利用がより広範囲に行われることなどがあります。

このプログラミング・モデルでは JCA 1.5 標準を利用しているため、IMS TM リソース・アダプターが、コールアウト・プロトコルと、適切な要求への応答の相関をユーザーに代わって管理します。そのため、管理コールアウト・プログラミング・モデルは、その使いやすさ、よりクリーンなコード、より高い再使用率、そしてより優れたスケーラビリティーという点で、非管理コールアウト・プログラミング・モデルより推奨されています。

次の図は、同期コールアウト・メッセージがどのように MDB によって処理されるかを示しています。

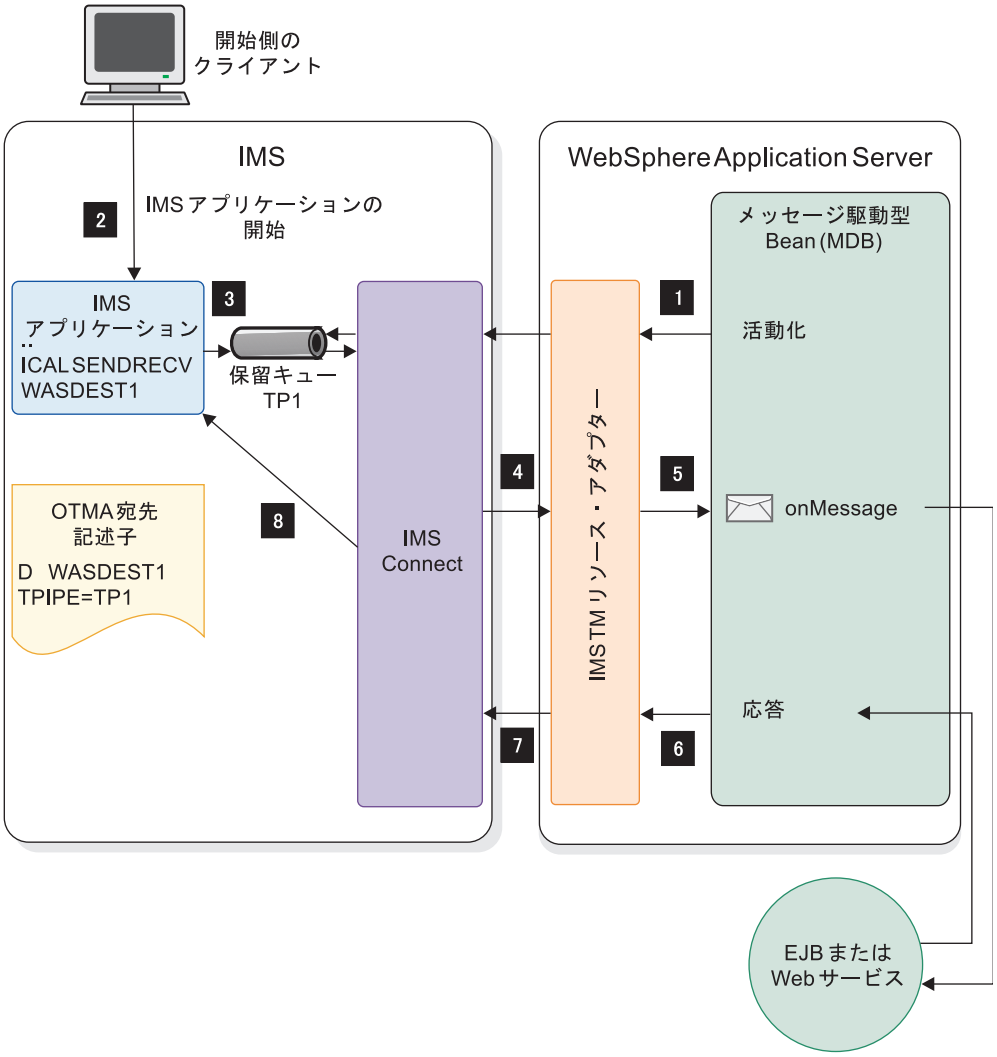


図 *118.* メッセージ駆動型 *Bean* を使用した *IMS* からの同期コールアウト・メッセージの処理

次の図は、非同期コールアウト・メッセージがどのように MDB によって処理されるかを示しています。

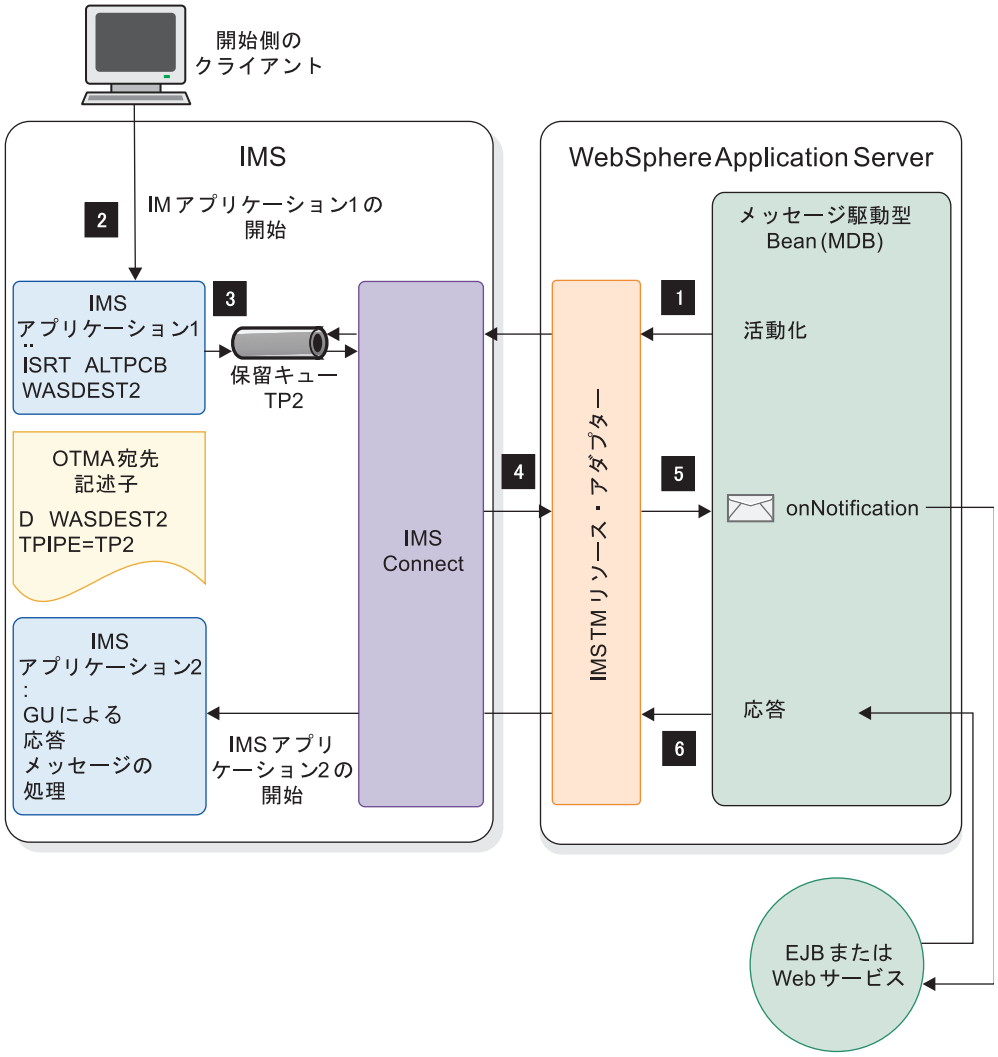


図 *119.* メッセージ駆動型 *Bean* を使用した *IMS* からの非同期コールアウト・メッセージの処理

MDB コールアウト・プログラミング・モデルでは、MDB は以下を実装する必要があります。

- com.ibm.j2ca.base.ExtendedInboundListener インターフェース
- 同期コールアウト要求を受け取るための onMessage() メソッド
- 非同期コールアウト要求メッセージを受け取るための onNotification() メソッド

これらのメソッドは、コールアウト・メッセージが非同期か同期かに応じて、特定の MDB に対するメッセージが到着したときに IMS TM リソース・アダプターによって呼び出されます。 IMS TM リソース・アダプターは自動的に、コールアウト・プロトコルを処理します。同期コールアウト・メッセージの場合、IMS TM リソース・アダプターは相関トークンの引き渡しも実施します。

肯定応答送信専用プロトコルを有効にすることができます。このプロトコルを使用すると、IMS TM Resource Adapter は、IMS が同期コールアウト応答メッセージを受信したときに IMS から肯定応答を受け取ります。バージョン 13 以前では、この肯定応答は使用できません。

IMS に返される同期コールアウト・メッセージの接続プーリングを有効にすることができます。バージョン 13 以前では、IMS TM Resource Adapter は、コールアウト応答が送信されるたびに新規接続を作成し、その後に破棄する必要があります。接続プーリングを有効にすると、複数の接続を開いたり閉じたりするオーバーヘッドを回避できます。

メッセージ駆動型 ***Bean*** を使用した ***IMS*** コールアウト・メッセージのリトリーブ***:***

MDB を使用して IMS コールアウト・メッセージをリトリーブするには、以下のようにします。

1. IBM Rational Application Developer for WebSphere Software、または J2C ツールを含むその他の統合開発環境 (IDE) を使用して、データ・バインディングおよび MDB を生成します。
   a. IMS アプリケーション・ソース・ファイルを IDE にインポートします。
   b. J2C ウィザードを使用して、コールアウト要求およびコールアウト・メッセージを処理するためにデータ・バインディングを生成します。
2. コールアウト・メッセージを listen するために com.ibm.j2ca.ExtendedInboundListener インターフェースを実装します。
   - 同期コールアウト要求のための onMessage() メソッドを実装します。
   - 非同期コールアウト要求メッセージのための onNotification() メソッドを実装します。
3. WebSphere Application Server でJ2C アクティベーション・スペック (IMSActivationSpec クラスのインスタンス) を構成し、コールアウト要求を listen するための以下のプロパティーの値を、少なくとも 1 つ指定します。
   - ホスト名 (Host name)
   - ポート番号 (Port number)
   - データ・ストア名 (Data store name)
   - キュー名 (Queue names) (TPIPE のコンマ区切りのリスト)
4. WebSphere Application Server 管理コンソールを使用して、WebSphere Application Server に MDB をデプロイします。
   a. 管理コンソールの 「**Bind listeners for message-driven beans**」パネルで、事前に構成した J2C アクティベーション・スペックの JNDI 名を入力します。
   b. MDB アプリケーションを開始します。

J2C アクティベーション・スペックの構成***:***

メッセージ駆動型 Bean の IMS からのインバウンド・メッセージングを記述するように、J2C アクティベーション・スペックを構成する必要があります。

J2C アクティベーション・スペックは、JCA 1.5 リソース・アダプターのインバウンド・メッセージング・サポートの構成の一部です。JCA 1.5 リソース・アダプターは、その独自のメッセージ・リスナー・インターフェースを実装することにより、インバウンド・メッセージングをサポートします。メッセージ・リスナーは、リソース・アダプターがインバウンド・メッセージをメッセージ・エンドポイントに伝えるために使用するインターフェースです。メッセージ駆動型 Bean は、メッ

セージ・エンドポイントであり、リソース・アダプターによって提供されるメッセージ・リスナー・インターフェースの 1 つを実行します。

メッセージ駆動型 Bean を含むアプリケーションがデプロイされる場合、デプロイヤーは、メッセージ駆動型 Bean が実装するのと同じタイプのメッセージ・リスナーをサポートするリソース・アダプターを選択する必要があります。メッセージ駆動型 Bean のデプロイメントの一環として、デプロイヤーは、J2C アクティベーション・スペックに設定するプロパティーを指定する必要があります。後で、アプリケーションの開始時に、アクティベーション・スペック・インスタンスが作成されるため、それらのプロパティーが設定され、エンドポイントの活動化に使用されます。

IMSActivationSpec クラスは、IMS TM リソース・アダプターでの J2C アクティベーション・スペックの実装です。WebSphere Application Server でメッセージ駆動型 Bean をデプロイする場合は、IMS TM リソース・アダプターを選択し、IMS からのインバウンド通信を記述する IMSActivationSpec プロパティーの値を指定することによってアクティベーション・スペックを構成する必要があります。

複数のデータ・ストアと複数の TPIPE がある場合、IMS TM リソース・アダプター・スレッド・プールは短時間で使い果たされ、複数のデータ・ストアと TPIPE が関係していると、コールアウト処理が停止する可能性があります。この場合、独自のスレッド・プールを追加するよう WebSphere Application Server を構成することも考えられます。スレッド・プールの追加後、IMS TM リソース・アダプターの「Thread pool alias」フィールドのスレッド・プールを参照してください。

関連タスク:

942 ページの『WebSphere Application Server でのスレッド・プールの構成』

WebSphere Application Server* での *J2C* アクティベーション・スペックの構成*:

IMS TM リソース・アダプターから新規 J2C アクティベーション・スペックを構成してから、IMS との通信の値を指定する必要があります。

WebSphere Application Server で J2C アクティベーション・スペックを作成および構成するには、次のようにします。

1. IMS TM リソース・アダプターの J2C スペックを作成します。
   a. IMS TM リソース・アダプターの J2C スペックを作成します。WebSphere Application Server の管理コンソール (Integrated Solutions Console とも呼ばれます) では、ナビゲーション・ペインの「**Resources**」 > 「**Resource Adapters**」 > 「**J2C activation specification**」を選択します。

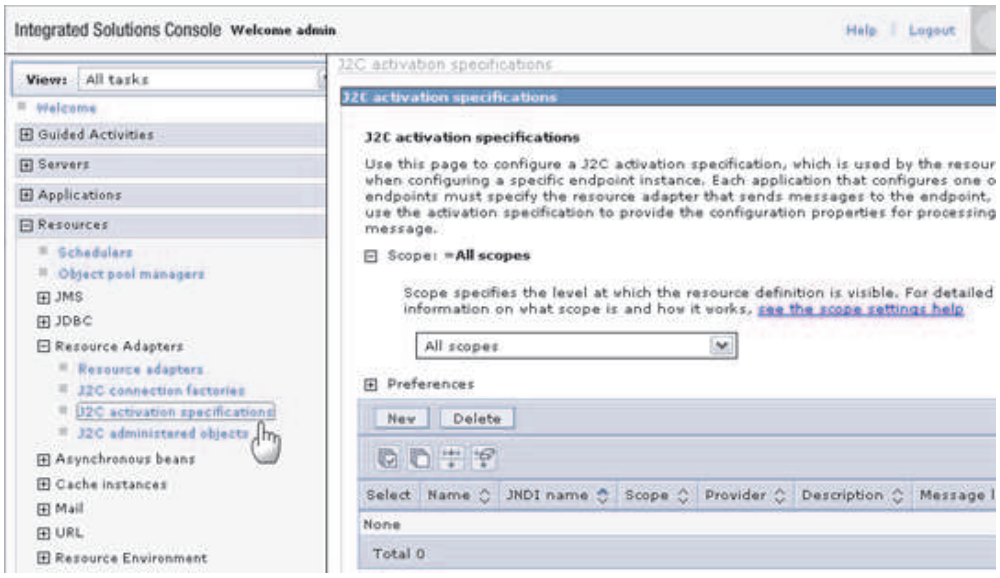


図 *120. WebSphere Application Server* での *J2C* アクティベーション・スペック構成

b. コンテンツ・ペインで「**New**」をクリックします。 「Configuration」タブが表示されます。

c. 「Provider」のリストから「IMS TM リソース・アダプター」を選択します。 このページ下部の「Message listener type」フィールドは、その選択に基づいて自動的に設定されます。

d. この J2C アクティベーション・スペックの名前を指定します。 例: `IMSActivationSpec`

e. JNDI 名を入力します。 例: `eis/IMSActivationSpec`

f. 「**Apply**」をクリックします。 J2C アクティベーション・スペックのページに戻ります。

g. 上部にあるメッセージ・ボックスの「**Save**」をクリックして、マスター構成に対する変更を保管します。 J2C アクティベーション・スペックが作成されます。

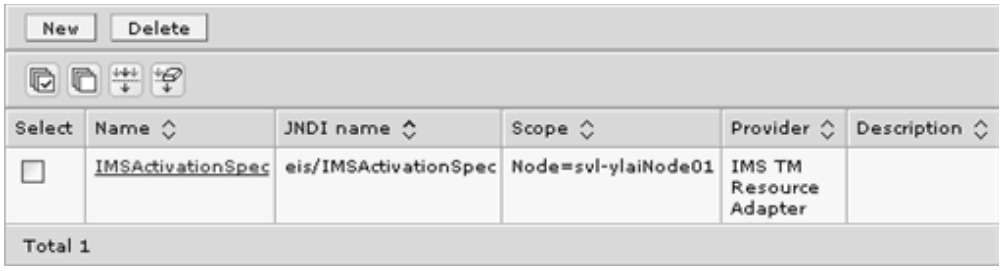


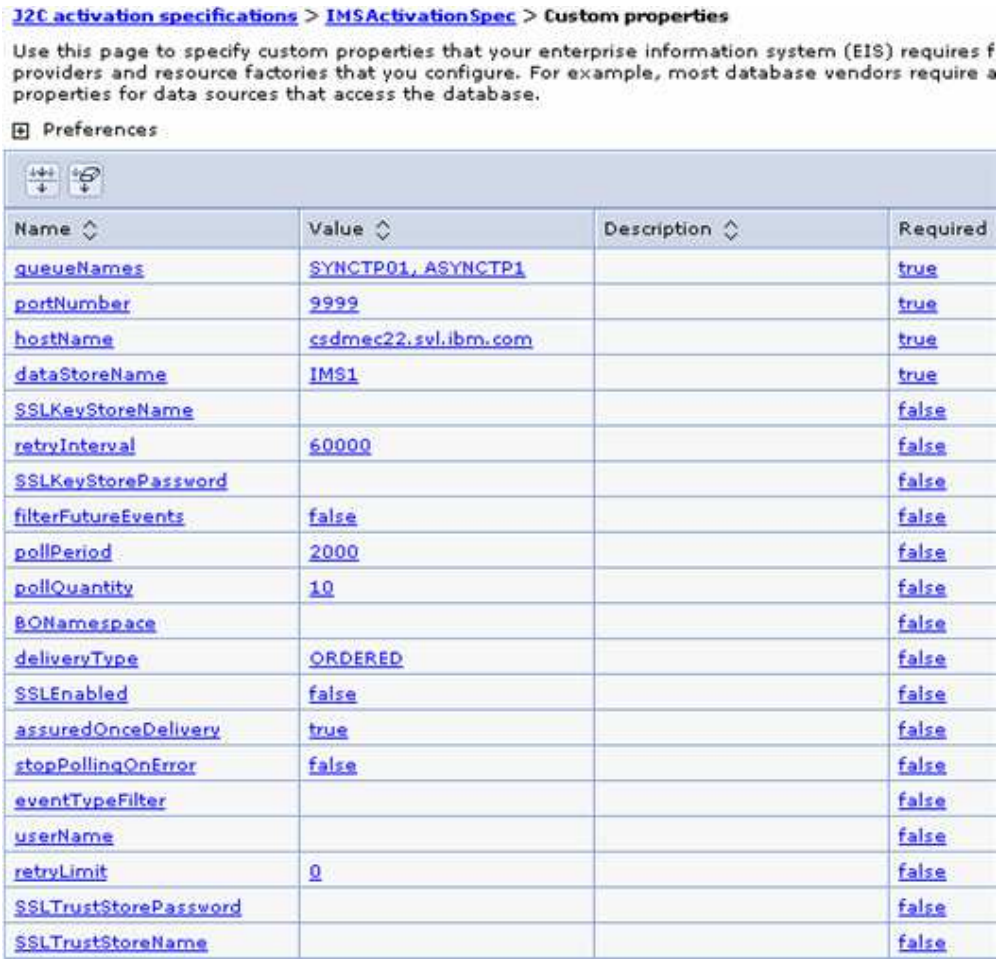
図 *121. IMS TM* リソース・アダプターから新規作成した *J2C* アクティベーション・スペック

2. IMS TM リソース・アダプターで使用するように J2C アクティベーション・スペックを構成するには、以下のようにします。

a. IMS TM リソース・アダプターの J2C アクティベーション・スペックの名前をクリックします。 「Configuration」タブが表示されます。

b. サイドにある「Additional Properties」セクションで、「**J2C activation specification custom properties**」リンクをクリックします。 この J2C アクティベーション・スペックのインスタンスのカスタム・プロパティーが表示されます。

注: 次の図のプロパティーは、ご使用の IMS TM リソース・アダプターのバージョン次第で、表示されるプロパティーと一致しない場合があります。



J2C activation specifications > IMSActivationSpec > Custom properties

Use this page to specify custom properties that your enterprise information system (EIS) requires f providers and resource factories that you configure. For example, most database vendors require a properties for data sources that access the database.

Preferences

Name	Value	Description	Required
queueNames	SYNCTP01, ASYNCTP1		true
portNumber	9999		true
hostName	csdmec22.svl.ibm.com		true
dataStoreName	IMS1		true
SSLKeyStoreName			false
retryInterval	60000		false
SSLKeyStorePassword			false
filterFutureEvents	false		false
pollPeriod	2000		false
pollQuantity	10		false
BONamespace			false
deliveryType	ORDERED		false
SSLEnabled	false		false
assuredOnceDelivery	true		false
stopPollingOnError	false		false
eventTypeFilter			false
userName			false
retryLimit	0		false
SSLTrustStorePassword			false
SSLTrustStoreName			false

図 *122. IMS TM* リソース・アダプターの *J2C* アクティベーション・スペックのカスタム・プロパティー

c. テーブルで以下の必須プロパティーの名前をクリックし、該当する値を入力して、これらのプロパティーの値を指定します。

- queueName: IMS からのコールアウト要求が保留中になっている TPIPE 保留キュー名の、コンマ区切りのリスト。
- portNumber: IMS Connect のポート番号
- hostName: IMS Connect のホスト名
- dataStoreName: IMS データ・ストア名。

WebSphere Application Server バージョン 8 以降では、データ・ストア名は名前のコンマ区切りリストにすることができ、IMSActivationSpec インスタンスが複数のデータ・ストアからコールアウト・メッセージを引き出すことができます。

d. IMS に返送される応答用の接続プールを IMS TM リソース・アダプターで使用するようにする場合は、以下のプロパティーの値を指定します。
   - connectionPoolEnabled: IMS TM リソース・アダプターが接続プールを使用するかどうか (この機能を使用可能にするには `true` を指定します)
   - maxConnections: 接続プールが保持できる接続の最大数
   - poolCleanupFrequency: 接続プール内のアイドル状態の接続のクリーンアップ間隔 (分)
   - ddurationOfIdleConnections: 次のクリーンアップ間隔で接続がクリーンアップされる前に、接続プール内の接続をアイドル状態で保持できる分数

e. IMS が同期コールアウト応答メッセージを受信したときに IMS TM リソース・アダプターで IMS からの肯定応答を受信したい場合は、sendOnlyWithAck プロパティーに `true` を指定します。

f. 環境に基づいて、その他のオプション・プロパティーの値を指定します。

g. 「**Apply**」をクリックします。

h. 上部にあるメッセージ・ボックスの「**Save**」をクリックして、マスター構成に対する変更を保管します。

IMS TM リソース・アダプターの J2C アクティベーション・スペックが作成され、構成されます。

メッセージ駆動型 ***Bean*** のための ***IMSActivationSpec*** プロパティーの構成*:*

IMSActivationSpec オブジェクトのプロパティーの値は、メッセージ駆動型 Bean によって使用される IMS からのインバウンド通信を記述します。

IMS TM リソース・アダプターが IMS Connect と通信して同期コールアウト・メッセージをリトリーブして応答するために、メッセージ駆動型 Bean をデプロイする場合は、IMSActivationSpec オブジェクトのインスタンスを構成する必要があります。

以下の表では、IMSActivationSpec オブジェクト (IMS TM リソース・アダプター内での J2C アクティベーション・スペックの実装) のプロパティーについて説明します。WebSphere Application Server の管理コンソールを使用して、これらのプロパティーの値を指定し、構成してください。

以下の表にリストされていない J2C アクティベーション・スペックのプロパティーは、IMS TM リソース・アダプターではサポートされません。

表 115. *IMSActivationSpec* オブジェクトのプロパティー

プロパティー	説明
dataStoreName	IMS データ・ストア名を指定します。この名前は、IMS Connect のインストール時に IMS Connect 構成メンバーに指定された Datastore ステートメントの ID パラメーターと一致する必要があります。この名前は、IMS Connect と OTMA の間でシステム間カップリング・ファシリティー (XCF) 通信が行われる際に、IMS の XCF メンバー名としても使用されます。 データ・ストア名として、複数のデータ・ストア名をコンマ区切りでリストした値を設定できます。このようにすると、IMSActivationSpec クラスの 1 つのインスタンスで複数の IMS データ・ストアからコールアウト・メッセージを引き出せるようになります。
groupName	セキュリティー許可機能 (SAF) のグループ名を指定します。
hostName	IMS Connect のホスト名を指定します。
password	SAF パスワードを指定します。
portNumber	IMS Connect のポート番号を指定します。
queueNames	コールアウト (同期または非同期) メッセージ用の IMS OTMA TPIPE 名のコンマ区切りリストを指定します。キュー名は、1 から 8 文字の英数字 (A から Z、0 から 9、@、#、$) であることが必要です。
resumeTpipeNsc	IMS TM リソース・アダプターが、IMS コールアウト・メッセージのネットワーク・セキュリティー資格情報をサポートするかどうかを指定します。 **true** IMS TM リソース・アダプターは、ネットワーク・セキュリティー資格情報をサポートします。 **false** IMS TM リソース・アダプターは、ネットワーク・セキュリティー資格情報をサポートしません。
retryInterval	IMS TM リソース・アダプターが、データ・ストアが使用可能かどうかをチェックするまでの時間遅延 (ミリ秒) を指定します。
retryLimit	IMS Connect または IMS データ・ストアの可用性の問題が原因で接続が失われたときに、IMS TM リソース・アダプターが IMS Connect への再接続を試行する最大回数を指定します。
SSLEnabled	IMSActivationSpec オブジェクトに指定されたホスト名とポート番号を使用して、IMS Connect への Secure Sockets Layer (SSL) ソケット接続を作成するように、IMS TM リソース・アダプターに指示します。このプロパティーは、TCP/IP 接続の場合のみ有効です。
SSLEncryptionType	SSL 暗号化タイプを指定します。有効な値は、`strong` と `weak` です。

表 115. *IMSActivationSpec* オブジェクトのプロパティー (続き)

プロパティー	説明
SSLKeyStoreName	TCP/IP SSL 通信の鍵ストアの名前 (絶対ファイル・パスを含む) を指定します。 秘密鍵およびそれぞれの関連する公開鍵証明書は、鍵ストアと呼ばれるパスワード保護されたデータベースに保管されます。信頼証明書も鍵ストアに保管でき、トラストストア・プロパティーは空にすることも、鍵ストア・ファイルを指定することもできます。鍵ストア名の例は、`c:¥keystore¥MyKeystore.ks` です。ファイルに他のファイル拡張子を付けることもできます。
SSLKeyStorePassword	TCP/IP SSL 通信の鍵ストアのパスワードを指定します。
SSLTrustStoreName	TCP/IP SSL 通信のセキュリティー資格情報 (証明書) が含まれている鍵ストア・ファイルの名前 (絶対パスを含む) を指定します。SSLTrustStoreName プロパティーの値は、鍵ストアが使用されている場合は必須ではありません。トラストストア・ファイルは、公開鍵または証明書を含む鍵データベース・ファイルです。秘密鍵もトラストストアに保管でき、鍵ストア・プロパティーは空にすることも、トラストストア・ファイルを指定することもできます。トラストストア名の例は、`c:¥keystore¥MyTruststore.ks` です。ファイルに他のファイル拡張子を付けることもできます。
SSLTrustStorePassword	TCP/IP SSL 通信のトラストストアのパスワード。
connectionPoolEnabled	接続プーリングを使用可能にするかどうかを指定します。 **`true`** IMS TM Resource Adapter が、IMS に返送される同期コールアウト応答用の接続プールを作成できるようにします。 **`false`** 接続プーリングを使用不可にします。IMS TM Resource Adapter は、同期コールアウト応答が送信されるたびに新規接続を作成し、その後で破棄します。デフォルト値は `false` です (IMS TM Resource Adapter の旧バージョンとの互換性のため)。
durationOfIdleConnections	次のクリーンアップ間隔で接続がクリーンアップされる前に、接続プール内の接続をアイドル状態で保持できる分数を指定します。デフォルト値は `0` で、アイドル状態の接続は即時にクリーンアップされます。connectionPoolEnabled プロパティーが `false` の場合、このプロパティーは無視されます。
maxConnections	接続プールが保持できる接続の最大数を指定します。プールが満杯になると、すべての新規接続は破棄されます。デフォルト値は `1` です。connectionPoolEnabled プロパティーが `false` の場合、このプロパティーは無視されます。
poolCleanupFrequency	接続プール内のアイドル状態の接続のクリーンアップ間隔 (分) を指定します。デフォルトは `20` です。connectionPoolEnabled プロパティーが `false` の場合、このプロパティーは無視されます。

表 115. *IMSActivationSpec* オブジェクトのプロパティー (続き)

プロパティー	説明
sendOnlyWithAck	肯定応答送信専用プロトコルを使用可能にするかどうかを指定します。 **true** プロトコルを使用可能にします。IMS TM Resource Adapter は、IMS が同期コールアウト応答メッセージを受信すると、IMS からの肯定応答を受信します。 **false** プロトコルを使用不可にします。デフォルト値は `false` です (IMS TM Resource Adapter の旧バージョンとの互換性のため)。
userName	SAF ユーザー名。

関連情報:

J2C アクティベーション・スペックの構成 (WebSphere Application Server V8 インフォメーション・センター)

WebSphere Application Server でのスレッド・プールの構成*:*

同期コールアウト要求処理に複数のデータ・ストアおよび複数の TPIPE を使用する場合、WebSphere Application Server でスレッド・プールを構成します。

スレッド・プールを使用すると、サーバーのコンポーネントがスレッドを再利用できるようになり、実行時に新しいスレッドを作成する必要がなくなります。IMS TM リソース・アダプターのスレッド・プールが短時間で使い果たされ、複数のデータ・ストアおよび TPIPE が関係している場合、コールアウト処理は停止します。

複数のデータ・ストアと複数の TPIPE がある場合は、独自のスレッド・プールを追加するよう WebSphere Application Server を構成します。IMS TM リソース・アダプターのスレッド・プールが短時間で使い果たされ、複数のデータ・ストアと TPIPE が関係している場合、コールアウト処理は停止します。開始するスレッド・プール・サイズは numberOfDatastore * numberOfQueues + numberOfActiveWorkload です (例えば、1 秒当たり 20 個のメッセージ駆動型 Bean 要求の場合は 20)。ただし、実際の数には、ご使用の環境と使用状況に基づくテストが必要です。

1. WebSphere Application Server 管理コンソールに進みます。
2. 「**Application servers**」 > 「*server_name*」 > 「**Additional Properties**」 > 「**Thread Pools**」を選択します。
3. スレッド・プールの最大サイズを設定します。 開始するスレッド・プール・サイズは `number_of_data_store * number_of_queues * number_of_active_workload` です。ここで、`number_of_active_workload` は、処理したい要求の 1 秒当たりの要求数です。

   実際の番号には、ご使用の環境と使用状況に基づくテストが必要です。
4. IMS TM リソース・アダプター構成のスレッド・プールを参照します。

例えば、次の例に示すように、IMSTMRA というスレッド・プールを追加する場合は、以下のようにします。

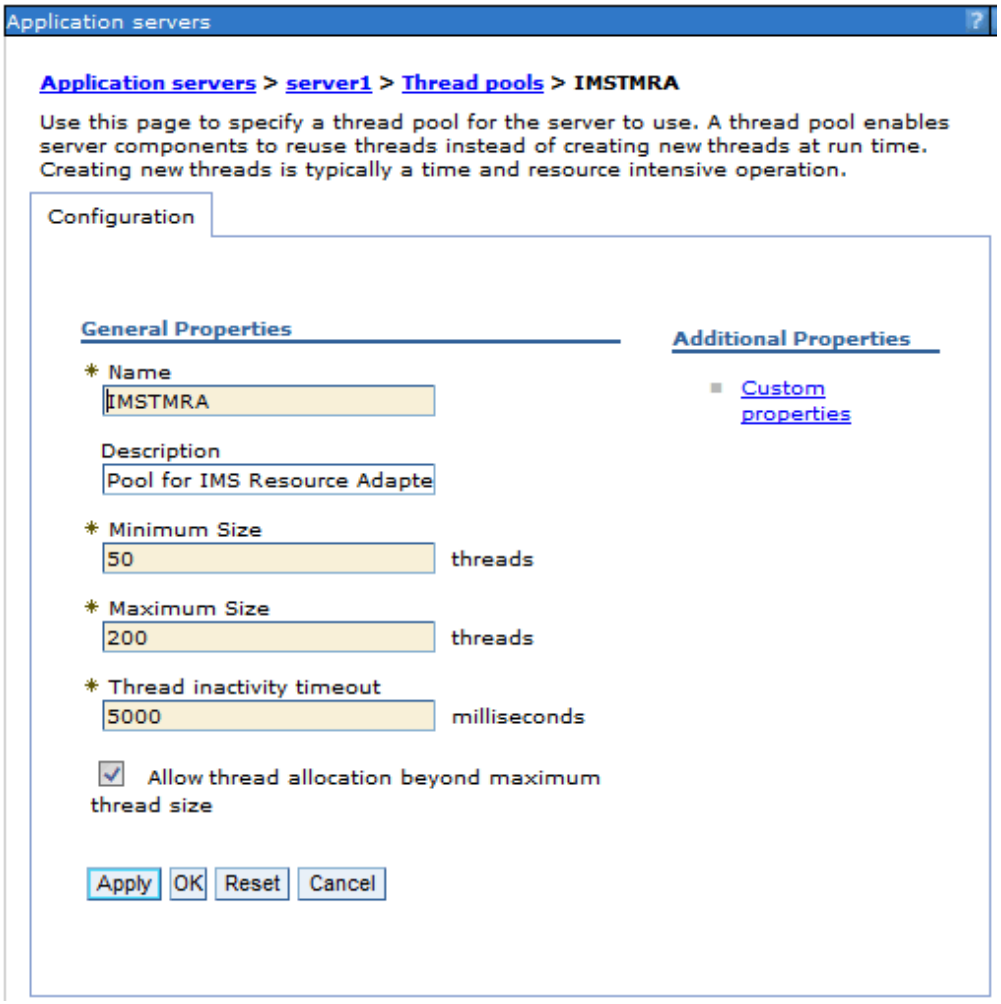


図 123. *WebSphere Application Server* でのスレッド・プール構成

スレッド・プールを、IMS TM リソース・アダプターの「Thread pool alias」フィールドに追加できます。

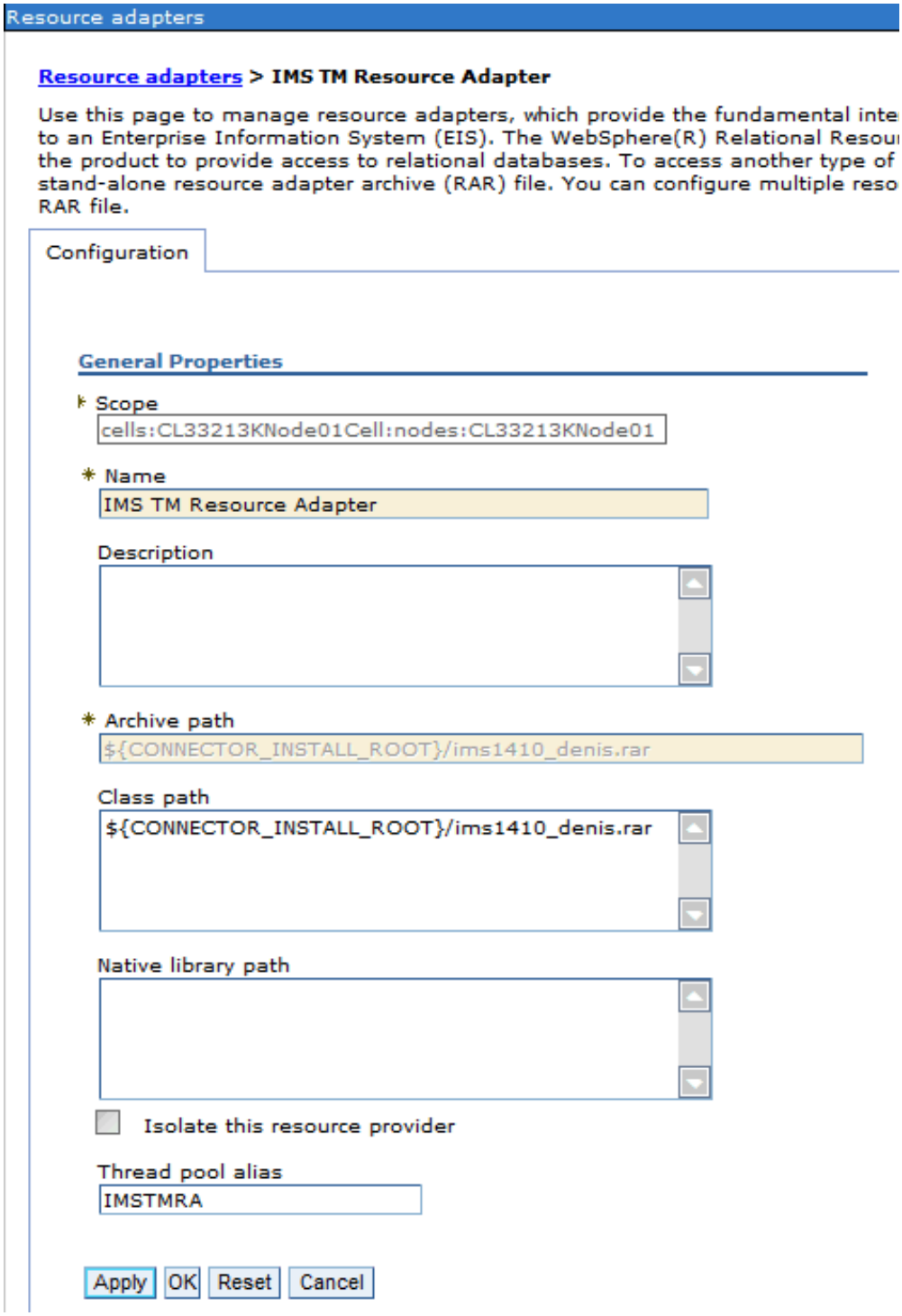


図 *124. WebSphere Application Server* でのリソース・アダプター構成

IMS からコールアウト要求を受信するためのサンプル ***MDB:***

MDB は InboundListener インターフェースを実装するので、onMessage() メソッドと onNotification() メソッド (それぞれ、同期処理および非同期処理に使用されます) を実装する必要があります。

次のサンプルは、IMS からの同期および非同期の両方のコールアウト・メッセージを listen するように構成された、MDB の Bean 実装クラス (IMSCalloutIVPMDBBean) を示しています。

```
package ejbs;

import java.io.IOException;
import javax.resource.ResourceException;
import javax.resource.cci.Record;
import com.ibm.connector2.ims.ico.IMSInputStreamRecord;
import commonj.connector.runtime.InboundInteractionSpec;

/**
 * Bean implementation class for Enterprise Bean: IMSInboundMessage
 */
public class IMSCalloutIVPMDBBean
 implements
  javax.ejb.MessageDrivenBean,
  com.ibm.j2ca.base.ExtendedInboundListener {

 private javax.ejb.MessageDrivenContext fMessageDrivenCtx;

 /**
  * getMessageDrivenContext
  */
 public javax.ejb.MessageDrivenContext getMessageDrivenContext() {
  return fMessageDrivenCtx;
 }

 /**
  * setMessageDrivenContext
  */
 public void setMessageDrivenContext(javax.ejb.MessageDrivenContext ctx) {
  fMessageDrivenCtx = ctx;
 }

 /**
  * ejbCreate
  */
 public void ejbCreate() {
 }

 /**
  * onMessage
  */
 public void onMessage(javax.jms.Message msg) {
 }

 /**
  * ejbRemove
  */
 public void ejbRemove() {
 }

 /**
   * onMessage
   */
 public javax.resource.cci.Record onMessage(
  javax.resource.cci.Record arg0,
  javax.resource.cci.InteractionSpec arg1)
  throws javax.resource.ResourceException {

  return processSyncMessageReceived(arg1);
 }

 /**
  * onNotification
```

```
 */
 public void onNotification(
  javax.resource.cci.Record arg0,
  javax.resource.cci.InteractionSpec arg1)
  throws javax.resource.ResourceException {

  processAsyncMessageReceived(arg0);
 }

 public void onNotification(Record arg0) throws ResourceException {
  processAsyncMessageReceived(arg0);
 }

 public Record onMessage(Record arg0) throws ResourceException {
  return processSyncMessageReceived(arg0);
 }

 public Record onMessage(InboundInteractionSpec arg0, Record arg1)
throws ResourceException {
  return processSyncMessageReceived(arg1);
 }

 public void onNotification(InboundInteractionSpec arg0, Record arg1)
throws ResourceException {
  processAsyncMessageReceived(arg1);
 }

 /*
  * Process synchronous callout requests from IMS and return a response
  */
 public Record processSyncMessageReceived(Object event) {

        SYNCCALLOUTREQUEST request = new SYNCCALLOUTREQUEST();
  SYNCCALLOUTRESPONSE response = new SYNCCALLOUTRESPONSE();

     try {
      // Request
      request.setBytes(((IMSInputStreamRecord)event).getBytes());

   System.out.println("Synchronous callout request from IMS: " +
     request.getSync__callout__request__str());

   // Response
   response.setSync__callout__response__str("HELLO FROM WEBSPHERE MDB");

   System.out.println("Synchronous callout response from WAS MDB: " +
     response.getSync__callout__response__str());

     } catch (IOException e) {
         System.err.println(e);
       }

  return response;
 }

 /*
  * Receive asynchonrous callout requests from IMS
  */
 public void processAsyncMessageReceived(Object event) {
        ASYNCCALLOUTREQUEST request = new ASYNCCALLOUTREQUEST();

     try {
      request.setBytes(((IMSInputStreamRecord)event).getBytes());

   System.out.println("Asynchronous callout request from IMS: " +
    request.getAsync__callout__request__str());
```

```
        } catch (IOException e) {
         System.err.println(e);
        }
 }
}
```

非管理 **(**クライアント管理**)** コールアウト・プログラミング・モデル**:**

非管理コールアウト・プログラミング・モデル を使用すると、メッセージ駆動型 Bean を使用しない任意の外部 Java アプリケーションから IMS アプリケーション・コールアウト要求をリトリーブできます。

コールアウト要求への応答データに関しては、Java アプリケーションがその応答データを、同期的に同じトランザクション内で同じ IMS アプリケーションに送信することも、非同期的に別のトランザクション内で送信することもできます。

同期コールアウト要求の場合は、管理コールアウト・プログラミング・モデル (コールアウト・プロトコルと応答の相関が、ユーザーに代わりメッセージ駆動型 Bean によって処理されるモデル) とは異なり、ユーザーが Java アプリケーションで以下の課題を処理する必要があります。

- アプリケーションでコールアウト・プロトコルを処理し、コールアウト・メッセージの保留キューをポーリングします。
- 応答が開始側要求に相関されるように、コールアウト要求に付加された相関トークンが確実に応答とともに返されるようにします。

ユーザーが同期コールアウト・プロトコルおよび要求への応答メッセージの相関を処理する必要があるため、このプログラミング・モデルはクライアント管理コールアウト・プログラミング・モデル とも呼ばれます。

一般に、クライアント管理コールアウト・プログラミング・モデルを使用する場合、以下の作業が必要です。

1. WebSphere Application Server 管理者は、Java アプリケーションによってコールアウト要求のリトリーブに使用される共用可能接続ファクトリーを構成する必要があります。
2. IMS OTMA 保留キュー (TPIPE) からコールアウト・メッセージをリトリーブします。
   - 同期コールアウト要求の場合:
     - EJB アプリケーションからの応答をコールアウト要求に相関させる必要があります。詳しくは、 948 ページの『Java アプリケーションから同期コールアウト要求への応答の相関』を参照してください。
     - 949 ページの『非 MDB アプリケーションからの同期コールアウト要求メッセージのリトリーブ』に説明されているステップに従います。
   - 非同期コールアウト要求の場合は、 951 ページの『非 MDB アプリケーションからの非同期コールアウト要求メッセージのリトリーブ』を参照してください。

肯定応答送信専用プロトコルを有効にすることができます。このプロトコルを使用すると、IMS TM Resource Adapter は、IMS が同期コールアウト応答メッセージを受信したときに IMS から肯定応答を受け取ります。バージョン 13 以前では、この肯定応答は使用できません。

IMS に返される同期コールアウト・メッセージの接続プーリングを有効にすることができます。バージョン 13 以前では、IMS TM Resource Adapter は、コールアウト応答が送信されるたびに新規接続を作成し、その後に破棄する必要があります。接続プーリングを有効にすると、複数の接続を開いたり閉じたりするオーバーヘッドを回避できます。

Java アプリケーションから同期コールアウト要求への応答の相関*:*

Java アプリケーションが出力応答を生成していて、その応答を最初の同期コールアウト要求と相関させる必要がある場合、その相関を行うのは IMS アプリケーションの役目です。

非同期コールアウトの場合、相関を行うには、コールアウト要求内で特定のデータ (例えば、メッセージ ID や固有の要求 ID) を定義します。このデータを最初の入力メッセージと相関させることができます。

同期コールアウトの場合は、コールアウト・メッセージで渡される相関関係子トークンを使用します。

次の図は、非 MDB アプリケーションで、IMS コールアウト要求をリトリーブし、応答を返すために行われるステップの概要を示したものです。

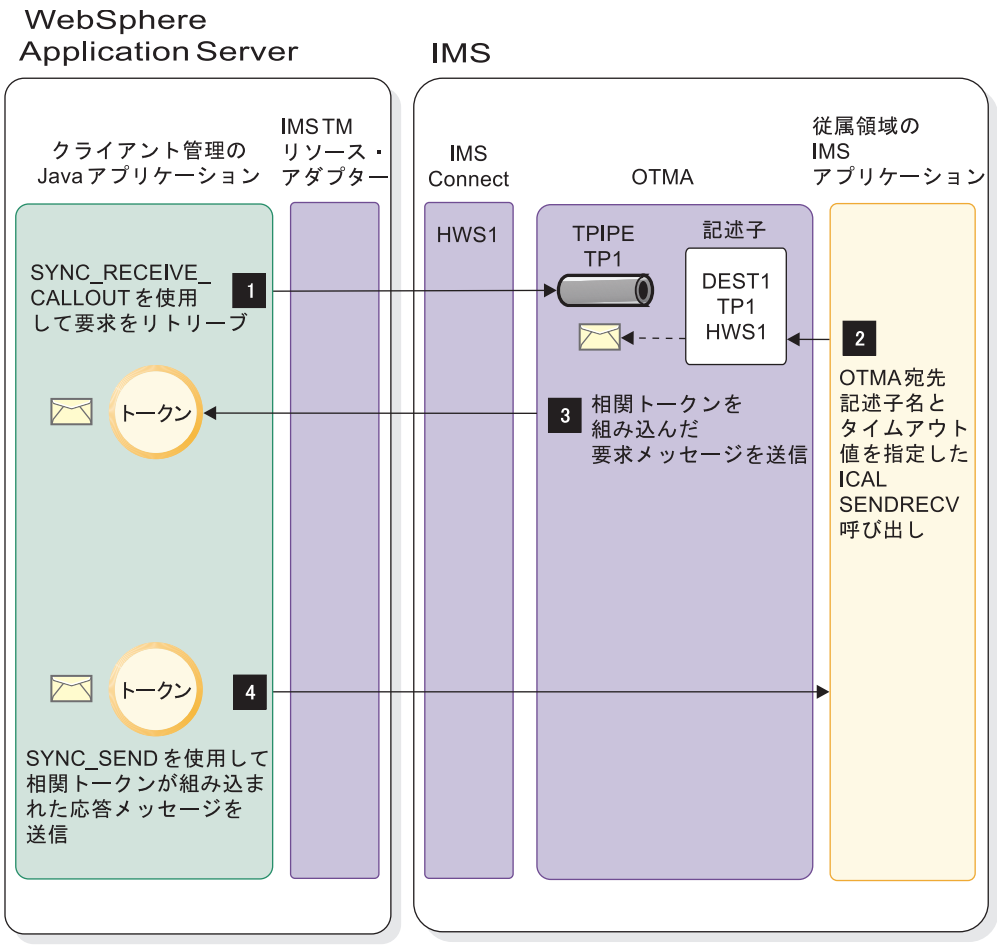


図 *125.* 非 *MDB Java* アプリケーションでの *IMS* からの同期コールアウト要求のリトリーブとその要求に対する応答のプロセス・フロー

1. Java アプリケーションが SYNC_RECEIVE_CALLOUT 要求を OTMA TPIPE TP1 に発行します。
2. IMS アプリケーションが SENDRECV 副次機能を指定して DL/I ICAL 呼び出しを発行し、OTMA 宛先記述子名とタイムアウト値を指定します。
3. IMS OTMA が、IMS Connect を介して IMS TM リソース・アダプターに要求メッセージを配信します。コールアウト要求とともに相関トークンが送信されます。
4. Java アプリケーションがコールアウト要求メッセージと相関トークンを受信します。要求を処理した後、Java アプリケーションは SYNC_SEND 要求を使用して応答メッセージと相関トークンを IMS に返します。IMS は、その相関トークンを使用して応答メッセージを対応する IMS トランザクション・インスタンスに相関させます。

非 ***MDB*** アプリケーションからの同期コールアウト要求メッセージのリトリーブ*:*

保留キュー内のコールアウト要求をリトリーブし、適切な要求への応答の相関を処理するように、Java アプリケーションを変更する必要があります。

前提条件: WebSphere Application Server 管理者は、Java アプリケーションによってコールアウト要求のリトリーブに使用される共用可能接続ファクトリーを構成する必要があります。

以下のステップは、IMS からの同期コールアウト要求をリトリーブおよび応答するための、Java アプリケーションでの主要ステップです。

1. 同期コールアウト要求をリトリーブするには、次のようにします。
   a. 対話コミット・モードを 0 (COMMIT_THEN_SEND) に設定します。

      ```
      interactionSpec.setCommitMode(IMSInteractionSpec.COMMIT_THEN_SEND);
      ```

   b. 対話同期レベルを CONFIRM に設定します。

      ```
      interactionSpec.setSyncLevel(IMSInteractionSpec.SYNC_LEVEL_CONFIRM);
      ```

   c. コールアウト・キュー名を OTMA TPIPE (保留キュー) 名に指定します。

      ```
      // An 8-character name of the OTMA asynchronous hold queue that the
      // messages are to be retrieved from
      String calloutQueueName = new String ("CALLOUTQ");
      // Set the queue name for the callout messages
      interactionSpec.setAltClientID(calloutQueueName);
      ```

   d. 対話 Verb を SYNC_RECEIVE_CALLOUT に設定します。

      ```
      interactionSpec.setInteractionVerb(IMSInteractionSpec.SYNC_RECEIVE_CALLOUT);
      ```

   e. リトリーブするコールアウト要求のタイプを指定します。同期のみ (CALLOUT_REQUEST_SYNC)、非同期 (CALLOUT_REQUEST_ASYNC)、またはその両方 (CALLOUT_REQUEST_BOTH)。 次の例では、同期コールアウト要求メッセージのみをリトリーブします。

      ```
      // Specify to retrieve only synchronous callout request messages
      interactionSpec.setCalloutRequestType(IMSInteractionSpec.CALLOUT_REQUEST_SYNC);
      ```

   f. 実行タイムアウト値を指定します。 次の例では、タイムアウト値を 5 秒に設定します。

      ```
      interactionSpec.setExecutionTimeout(5000);
      ```

   g. 対話を実行します。 次の例では対話が実行され、要求が calloutRequestMsg として返されます。

      ```
      iteraction.execute(interactionSpec, null, calloutRequestMsg);
      ```

2. IMSInteractionSpec インスタンスから相関トークンを取得します。 次の例では、タイムアウト値を 5 秒に設定します。

   ```
   byte[] corrToken = interactionSpec.getSyncCalloutCorrelationToken();
   ```

3. 要求を処理します。 前の例では、要求は calloutRequestMsg として返されます。
4. 応答を送信します。
   a. 対話 Verb を SYNC_SEND に設定します。

      ```
      interactionSpec.setInteractionVerb(IMSInteractionSpec.SYNC_SEND);
      ```

   b. 相関トークンが要求とともに元どおりに送信されるよう設定します。

      ```
      interactionSpec.setSyncCalloutCorrelationToken(corrToken);
      ```

   c. 対話を実行します。 次の例は、対話を実行し、コールアウト応答メッセージ (calloutRespondMsg) を送信します。

      ```
      iteraction.execute(interactionSpec, calloutRespondMsg, null);
      ```

同期コールアウト要求をリトリーブおよび処理するためのサンプルの非 ***MDB Java*** アプリケーション***:***

IMS アプリケーション・プログラムからのインバウンド要求に対応するように Java アプリケーションを準備するには、Java アプリケーションで適切な対話 Verb、TPIPE 名、およびタイムアウト値を指定します。

```
// JNDI lookup returns connFactory
InitialContext initCntx = new InitialContext();
ConnectionFactory connFactory =
(ConnectionFactory) initCntx.lookup("java:comp/env/ibm/ims/IMSTarget");

IMSConnectionSpec connSpec = new IMSConnectionSpec();
Connection connection = connFactory.getConnection(connSpec);
Interaction interaction = connection.createInteraction();
IMSInteractionSpec interactionSpec = new IMSInteractionSpec();

//set the commit mode and sync level
interactionSpec.setCommitMode(IMSInteractionSpec.COMMIT_THEN_SEND);
interactionSpec.setSyncLevel(IMSInteractionSpec.SYNC_LEVEL_CONFIRM);

// An 8-character name of the OTMA asynchronous hold queue that the
// messages are to be retrieved from
String calloutQueueName = new String ("CALLOUTQ");
// Set the queue name for the callout message
interactionSpec.setAltClientID(calloutQueueName);

// Set InteractionVerb for retrieving callout request with a timeout value
interactionSpec.setInteractionVerb(IMSInteractionSpec.SYNC_RECEIVE_CALLOUT);
// Specify to retrieve only synchronous callout request messages
interactionSpec.setCalloutRequestType(IMSInteractionSpec.CALLOUT_REQUEST_SYNC);
interactionSpec.setExecutionTimeout(5000);


// Execute the interaction
iteraction.execute(interactionSpec, null, calloutRequestMsg);

// Get correlation token
byte[] corrToken = interactionSpec.getSyncCalloutCorrelationToken();

// Further processing on the request (calloutRequestMsg)
:

// Send back the response (calloutRespondMsg) by using the SYNC_SEND interaction
interactionSpec.setInteractionVerb(com.ibm.connector2.ims.ico.IMSInteractionSpec.SYNC_SEND);
// SYNC_SEND does not support alternate client ID
interactionSpec.setAltClientID(null);
interactionSpec.setSyncCalloutCorrelationToken(corrToken);

// Execute the interaction
iteraction.execute(interactionSpec, calloutRespondMsg, null);

iteraction.close();
connection.close();
```

完全なコード・サンプルを取得するには、ここを右クリックし、「**Save Link As**」(FireFox の場合) または「対象をファイルに保存」 (Microsoft Internet Explorer の場合) を選択して Java サンプル・ファイルをダウンロードします。

非 ***MDB*** アプリケーションからの非同期コールアウト要求メッセージのリトリーブ***:***

保留キュー内のコールアウト要求をリトリーブし、応答メッセージが予期される場合は、応答を適切な要求へ相関させるように、Java アプリケーションを変更する必要があります。

前提条件**:** WebSphere Application Server 管理者は、Java アプリケーションによってコールアウト要求のリトリーブに使用される共用可能接続ファクトリーを構成する必要があります。

Java アプリケーションを変更して、以下のプロパティー値を設定してください。

1. コールアウト要求がキューに入れられる OTMA 非同期保留キュー (TPIPE) の名前を値として持つ代替クライアント ID を指定します。 コールアウト Java アプリケーションが listen する TPIPE は、コールアウト要求を受信するために予約されている必要があります。OTMA 非同期保留キューの名前は、次のとおりです。
   - コールアウト要求が挿入される ALTPCB の名前。または
   - OTMA 宛先記述子に指定された TPIPE の名前。
2. interactionVerb プロパティーを SYNC_RECEIVE_CALLOUT に設定します (これは、IMS Connect Resume Tpipe Single Wait 要求に相当)。
3. 保留キュー内のコールアウト・メッセージを待機する実行タイムアウト値を指定します。
4. セキュリティーを高めるために、権限があるユーザーのみが非同期保留キューからコールアウト要求メッセージをリトリーブできるように IMS セキュリティーを構成している場合、オプションで Java アプリケーション内にユーザー ID を指定できます。 このユーザー ID は、ご使用のアプリケーション内、またはアプリケーションが使用する接続ファクトリー内の connectionSpec オブジェクトに指定する必要があります。

非同期コールアウト要求をリトリーブするためのサンプルの ***Java*** アプリケーション*:*

IMS アプリケーション・プログラムからのインバウンド要求に対応するように非 MDB Java アプリケーションを準備するには、Java アプリケーションで適切な対話 Verb、非同期保留キュー名、およびタイムアウト値を指定します。

- 非同期保留キュー名を指定します。

  ```
  interactionSpec.setAltClientID(calloutQueueName);
  ```

- interactionVerb プロパティーを SYNC_RECEIVE_CALLOUT に設定します。

  ```
  interactionSpec.setInteractionVerb(com.ibm.connector2.ims.ico.
  IMSInteractionSpec.SYNC_RECEIVE_CALLOUT);
  ```

- 実行タイムアウト値 (ミリ秒) を指定します。

  ```
  interactionSpec.setExecutionTimeout(3600000);
  ```

  保留キュー内のコールアウト要求メッセージを待機するタイムアウト値を指定します。このサンプルでは、実行タイムアウト値に大きな値が指定されています。実行タイムアウト値を大きく設定すると、コールアウト要求が行われない期間が長くなると思われるときにコールアウト EJB コンポーネントで必要になるループを最小限に抑えることができます。タイムアウトの長さは、コールアウト要求間で経過すると予想される最大時間、およびその時間が過ぎた場合の EJB コンポーネントの動作によって異なります。

次のサンプル・コードは、コールアウト要求を受信するように構成された Java アプリケーションの例を示しています。

```
// JNDI lookup returns connFactory
InitialContext initCntx = new InitialContext();
ConnectionFactory connFactory =
      (ConnectionFactory) initCntx.lookup("java:comp/env/ibm/ims/IMSTarget");

IMSConnectionSpec connSpec = new IMSConnectionSpec();
```

```
Connection connection = connFactory.getConnection(connSpec);
Interaction interaction = connection.createInteraction();
IMSInteractionSpec interactionSpec = new IMSInteractionSpec();

//set the commit mode and sync level
interactionSpec.setCommitMode(0);
interactionSpec.setSyncLevel(1);

// An eight-character queue name that the asynchronous messages to be retrieved from
String calloutQueueName = new String ("CALLOUTQ");
// Set the asynchronous queue name for the callout message
interactionSpec.setAltClientID(calloutQueueName);

// Set interactionVerb to retrieve async output with a timeout
interactionSpec.setInteractionVerb(com.ibm.connector2.ims.ico.
IMSInteractionSpec.SYNC_RECEIVE_CALLOUT);
interactionSpec.setExecutionTimeout(3600000);

for (;;) {
      try {
            // Execute the interaction
            iteraction.execute(interactionSpec, null, calloutMsg);

            // Further processing on the calloutMsg
            :
      } catch (Exception e) {
            // if the exception is an execution timeout error,
            // you can either do nothing and continue to loop
            // or process the error and then break the loop
            break;
      }
}

iteraction.close();
connection.close();
```

コミット・モードおよび同期レベルの処理

選択するコミット・モードによって IMS が実行するコミット・モード処理のタイプが決まります。

Java クライアントは、そのクライアントがトランザクション要求を IMS にサブミットするときに使用されるコミット・モード・プロトコルを指定します。IMS Connect と IMS は、以下の 2 つのタイプのコミット・モード処理をサポートします。

- コミット・モード 0 (コミット後送信)。この場合、IMS は IMS データベース変更をコミットしてから、出力をクライアントに送信します。
- コミット・モード 1 (送信後コミット)。この場合、IMS は出力をクライアントに送信してから、データベース変更をコミットします。

コミット・モード・プロトコルに関連して、IMS Connect と IMS は、NONE、CONFIRM、および SYNCPT (同期点プロトコル) の 3 つの同期レベル (sync レベル) もサポートします。コミット・モード 1 では 3 つすべての同期レベルを使用できます。コミット・モード 0 では CONFIRM のみを使用できます。Java クライアントは同期レベルを明示的に SYNCPT に設定できません。IMS TM リソース・アダプターでは、以下の組み合わせのみがサポートされます。

コミット・モード	同期レベル **NONE** のサポートの有無	同期レベル **CONFIRM** のサポートの有無
コミット・モード 0	なし	あり

コミット・モード	同期レベル **NONE** のサポートの有無	同期レベル **CONFIRM** のサポートの有無
コミット・モード 1	あり	あり

IMS TM リソース・アダプターにより同期レベル SYNCPT でコミット・モード 1 が使用されるのは、このアダプターが IMS での 2 フェーズ・コミット処理に参加する場合です。

コミット・モードと同期レベルの組み合わせによってサポートされる対話

IMS TM リソース・アダプターでサポートされる対話は、選択するコミット・モードと同期レベルの組み合わせによって異なります。

IMS TM リソース・アダプターでサポートされるコミット・モードと同期レベルの組み合わせは、以下のとおりです。

- コミット・モード 1 と同期レベル NONE

  この組み合わせは、非トランザクション型の対話に使用されます。非会話型アプリケーションの場合は、SYNC_SEND_RECEIVE 対話を使用します。会話型アプリケーションの場合は、SYNC_SEND_RECEIVE を使用するか、オプションで SYNC_END_CONVERSATION 対話を使用します。
- コミット・モード 1 と同期レベル CONFIRM

  この組み合わせは、非トランザクション型の対話に使用されます。非会話型アプリケーションの場合は、SYNC_SEND_RECEIVE 対話を使用します。会話型アプリケーションの場合は、SYNC_SEND_RECEIVE を使用するか、オプションで SYNC_END_CONVERSATION 対話を使用します。
- コミット・モード 0 と同期レベル CONFIRM

  この組み合わせは、非トランザクション型対話 SYNC_SEND_RECEIVE、SYNC_SEND、 SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT、SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT、および SYNC_RECEIVE_CALLOUT に使用されます。

  ヒント: コミット・モード 0 は、TCP/IP 接続で実行されている非会話型アプリケーションでのみサポートされます。

コミット・モード 1 の対話の場合、IMS TM リソース・アダプターが IMS Connect との通信時に自動的に同期レベルを指定します。また、同期レベルは、Java クライアントにより setSyncLevel(int) メソッドを使用して明示的に NONE または CONFIRM に設定することもできます。デフォルトの同期レベルは NONE です。同期レベルを設定せずに対話 Verb を SYNC_SEND_RECEIVE に設定できます。

コミット・モード 0 の対話の場合は、CONFIRM 同期レベルのみがサポートされます。

無効な組み合わせのコミット・モードと同期レベルを使用すると、setSyncLevel(int) メソッドが例外をスローします。

関連概念:

1025 ページの『出力メッセージを含む Java 例外』

トランザクション・パイプ、クライアント ID、および対話 Verb の指定

対話 Verb は、使用するコミット・モードに関係なく指定する必要があります。トランザクション・パイプ (TPIPE) とクライアント ID の指定は、使用するコミット・モードまたはソケット接続のタイプによって異なります。

IMS と OTMA の用語では、トランザクション・パイプ (TPIPE) は、クライアント (IMS Connect) とサーバー (IMS OTMA) の間の論理接続です。IMSConnectionSpec クラスには、その IMSConnectionSpec に関連付けられた接続を識別する clientID プロパティーがあります。対話中にリカバリー可能出力メッセージが入れられる IMS OTMA 非同期出力キューまたは TPIPE の名前は、その対話で使用されているコミット・モードに応じて、以下の 2 つの方法のいずれかで特定されます。

- コミット・モード 0 の対話では、TPIPE はその対話に使用されているクライアント ID によって識別されます。コミット・モード 0 のトランザクションに使用されるクライアント ID ごとに、それぞれ独自の TPIPE が使用されます。
  - 共用可能ソケット接続での対話では、クライアント ID は自動的に生成されます。
  - 専用ソケット接続での対話では、クライアント ID はユーザーが設定する必要があります。

  通常は、専用ソケット接続ではなく共用可能ソケット接続を使用してください。共用可能ソケット接続を使用すると、接続をより効率的に使用できます。さらに、共用可能ソケット接続は、Sysplex Distributor 環境や WebSphere Application Server for z/OS クローン環境では問題が発生しにくい傾向にあります。これらの環境では、複数のインスタンスの WebSphere Application Server が同じ IMS Connect と通信し、接続用に同じクライアント ID を使用しようとすると考えられます。
- コミット・モード 1 の対話の場合、TPIPE は、その対話に使用される IMS Connect ポート番号によって識別されます。そのため、各ポートには、そのポートでコミット・モード 1 の対話を実行するすべてのクライアントで使用される TPIPE があります。

Java クライアントがコミット・モード 1 またはコミット・モード 0 のどちらで IMS トランザクションを実行している場合でも、その Java クライアントが対話の interactionVerb プロパティーの値を指定します。コミット・モード 0 の対話が指定されていて、Java クライアントによって専用ソケット接続が使用されている場合、Java クライアントはさらに、その接続のクライアント ID の値を指定する必要があります。

ソケット接続

IMS TM リソース・アダプターによって作成されるすべてのソケット接続は、永続的です。また、共用可能や専用にすることができます。

共用可能永続ソケットは、コミット・モード 0 またはコミット・モード 1 の対話を実行するアプリケーションによって共用できます。専用永続ソケットは、コミット・モード 0 の対話を実行するアプリケーションのみが使用できます。

IMS TM リソース・アダプターと IMS Connect 間の 1 つのソケット接続を、IMS Connect との複数の対話用に順次に再使用できます。対話と対話の間で、ソケット接続が閉じて再び開くという動作は行われません。

専用永続ソケット

専用永続ソケットとは、ソケット接続が特定のクライアント ID に割り当てられており、接続が切断されるまでその特定クライアント ID に占有されることを意味します。

通常、専用永続ソケット接続は単一のアプリケーションによって順次に使用 (または再使用) されます。ただし、専用永続ソケット接続が複数のアプリケーションによって順次に使用される場合もあります。これらのアプリケーションは、同じクライアント ID を使用してソケットを取得できます。

複数の異なるアプリケーションが同じクライアント ID を使用して同時に専用永続ソケット接続を使用しようとした場合、最初のアプリケーションはその接続に成功します。最初のアプリケーションがまだ接続を使用しているときに他のアプリケーションが続けてその接続を使用しようとすると、IMS Connect から重複クライアント・エラー (DUPCLNT) を受け取ります。

重複クライアント・エラーは、特定のクライアント ID を使用して専用永続ソケット接続を取得するアプリケーションが、複数の WebSphere Application Server インスタンスを接続元として専用永続ソケット接続が行われる環境で実行される場合にも発生する可能性があります。例えば、ワークロード・マネージャーによって管理される複数の WebSphere Application Server インスタンスを含む環境では、重複クライアント・エラーが発生する可能性があります。

推奨事項: アプリケーションが専用永続ソケット接続を使用している場合は、そのアプリケーションをこのタイプの環境にデプロイしないでください。

専用永続ソケット接続は、単一の WebSphere Application Server インスタンスが単一の IMS Connect に接続される環境での使用を目的としています。状況によっては、単一の WebSphere Application Server インスタンスが複数のインスタンスの IMS Connect に接続されることもあります。この構成では、WebSphere Application Server と IMS Connect の間でシスプレックス・ディストリビューターが使用される場合、以下の対話はサポートされません。

- 2 フェーズ・コミット・リカバリー
- IMS 会話型トランザクション

サポートされるコミット・モードおよび対話

専用永続ソケットは、コミット・モード 0 の対話を実行する Java アプリケーションでのみ使用できます。

表 *116.* ソケット・タイプ別のサポートされるコミット・モード、対話 *Verb*、および同期レベル：

コミット・モード	ソケット・タイプ	対話 **Verb**	クライアント **ID**	同期レベル
CM1	共用可能永続	• SYNC_END_CONVERSATION (会話型トランザクションの場合) • SYNC_SEND_RECEIVE	自動生成および自動管理	NONE または CONFIRM
CM0	共用可能永続	• SYNC_SEND • SYNC_SEND_RECEIVE • SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT • SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT • SYNC_RECEIVE_CALLOUT (非管理コールアウト・プログラミング・モデルの場合)	自動生成および自動管理	確認
	専用永続	• SYNC_SEND • SYNC_SEND_RECEIVE • SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT • SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT	クライアント・アプリケーションによって指定 (clientID プロパティー)	確認

表 *117.* ソケット・タイプ別のサポートされるコミット・モード、対話 *Verb*、および同期レベル：

コミット・モード	ソケット・タイプ	対話 **Verb**	クライアント **ID**	同期レベル
CM1	共用可能永続	• SYNC_END_CONVERSATION (会話型トランザクションの場合) • SYNC_SEND_RECEIVE	自動生成および自動管理	NONE または CONFIRM

表 *117.* ソケット・タイプ別のサポートされるコミット・モード、対話 *Verb*、および同期レベル *(*続き*)*:

コミット・モード	ソケット・タイプ	対話 **Verb**	クライアント **ID**	同期レベル
CM0	共用可能永続	• SYNC_SEND • SYNC_SEND_RECEIVE • SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT • SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT • SYNC_RECEIVE_CALLOUT (非管理コールアウト・プログラミング・モデルの場合)	自動生成および自動管理	確認
	専用永続	• SYNC_SEND • SYNC_SEND_RECEIVE • SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT • SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT	クライアント・アプリケーションによって指定 (clientID プロパティー)	確認

SYNC_RECEIVE_ASYNCOUTPUT 対話 Verb は非推奨になり、SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT 対話 Verb に置き換えられました。

メッセージのリトリーブ

専用永続ソケットで SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT 対話および SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT 対話 (さらに非推奨の SYNC_RECEIVE_ASYNCOUTPUT 対話) を使用すると、クライアント・アプリケーションが以下のソースからメッセージをリトリーブできます。

- コミット・モード 0 の対話が失敗したために、IMS OTMA 非同期出力キューに入れられたメッセージ
- 代替プログラム連絡ブロック (ALTPCB) への挿入を実行した IMS アプリケーションから送信されたメッセージ
- 共用可能ソケット接続で実行されたトランザクションからの出力の転送によるメッセージ

これらのメッセージをリトリーブするには、クライアント・アプリケーションがクライアント ID を指定する必要があります。クライアント ID は、非同期出力メッセージがキューに入れられる TPIPE を表します。専用永続ソケット上では対話からの配信されなかった出力メッセージは、転送もパージもできません。

OTMA スーパー・メンバーを使用せずに複数の IMS システムを使用する場合、SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT 対話を使用しても、取得可能な出力メッセージをすべてリトリーブできないことがあります。

専用永続ソケット接続

専用永続ソケット接続は、最低でも以下のカスタム・プロパティーの値を使用して IMS 接続ファクトリーによって作成されます。

- ホスト名: IMS Connect を実行しているシステムの TCP/IP ホスト名
- ポート番号 : 関連するポート番号
- データ・ストア名: ターゲット IMS の名前
- CM0 専用: `true`

CM0Dedicated プロパティーの値が `true` に設定されていると、接続ファクトリーは専用永続ソケット接続を作成します。

複数の接続ファクトリーが、同じ IMS Connect インスタンスへの専用永続ソケットを作成するように構成されている場合、特定のクライアント ID へのソケットを専用にできるのは一度に 1 つの接続ファクトリーのみです。例えば、接続ファクトリーがクライアント ID CLIENT01 への専用のソケット接続を正常に作成したとします。2 番目の接続ファクトリーも、CLIENT01 への専用のソケット接続を作成しようとしています。最初の接続ファクトリーによって作成されたソケット接続がまだ IMS Connect に接続されている場合、2 番目の接続では次の例外を受け取ります。

```
javax.resource.spi.EISSystemException: ICO0001E:
com.ibm.connector2.ims.ico.IMSTCPIPManagedConnection@23766050.processOutputOTMAMsg
(byte [], InteractionSpec,Record) error. IMS Connect returned error: RETCODE=[8],
REASONCODE=[DUPECLNT].
Duplicate client ID was used;  the client ID is currently in use.
```

関連タスク:

967 ページの『専用永続ソケット接続の確立』

関連資料:

1059 ページの『クライアント ID (clientID)』

共用可能永続ソケット

共用可能永続ソケットは、コミット・モード 1 (CM1) またはコミット・モード 0 (CM0) の対話を実行する、複数のアプリケーションによって共用 (連続再利用) できるソケットです。

共用可能永続ソケットでコミット・モード 0 の対話を実行するアプリケーションの場合、IMS TM リソース・アダプターが自動的に接頭部 HWS を付けてクライアント ID を生成します。このクライアント ID は、ソケット接続と関連する OTMA TPIPE を表すとともに、これらを識別します。このタイプのソケットには、IMS TM リソース・アダプターによって生成されたクライアント ID のみを使用できます。共用可能永続ソケットで CM0 として定義されていない接続にクライアント ID を指定すると、例外がスローされてクライアントに返されます。

制約事項**:** 代替 PCB にメッセージを挿入する IMS アプリケーション・プログラムでは、代替 PCB に HWS で始まる名前を使用しないでください。IMS TM リソース・アダプターは HWS メッセージをリトリーブできません。

ヒント**:** クライアント ID は、代替クライアント ID とは異なります。代替クライアント ID は、IMSInteractionSpec クラスのプロパティーです。代替クライアント ID は OTMA 保留キュー (TPIPE) から非同期出力メッセージをリトリーブするために使用されます。

サポートされるコミット・モードと対話

共用可能永続ソケットは、コミット・モード 1 またはコミット・モード 0 の対話を実行する Java アプリケーションで使用できます。

表 *118.* ソケット・タイプ別のサポートされるコミット・モード、対話 *Verb*、および同期レベル：

コミット・モード	ソケット・タイプ	対話 **Verb**	クライアント **ID**	同期レベル
CM1	共用可能永続	• SYNC_END_CONVERSATION (会話型トランザクションの場合) • SYNC_SEND_RECEIVE	自動生成および自動管理	NONE または CONFIRM
CM0	共用可能永続	• SYNC_SEND • SYNC_SEND_RECEIVE • SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT • SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT • SYNC_RECEIVE_CALLOUT (非管理コールアウト・プログラミング・モデルの場合)	自動生成および自動管理	確認
	専用永続	• SYNC_SEND • SYNC_SEND_RECEIVE • SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT • SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT	クライアント・アプリケーションによって指定 (clientID プロパティー)	確認

表 *119.* ソケット・タイプ別のサポートされるコミット・モード、対話 *Verb*、および同期レベル：

コミット・モード	ソケット・タイプ	対話 **Verb**	クライアント **ID**	同期レベル
CM1	共用可能永続	• SYNC_END_CONVERSATION (会話型トランザクションの場合) • SYNC_SEND_RECEIVE	自動生成および自動管理	NONE または CONFIRM

表 *119.* ソケット・タイプ別のサポートされるコミット・モード、対話 *Verb*、および同期レベル *(*続き*)*:

コミット・モード	ソケット・タイプ	対話 **Verb**	クライアント **ID**	同期レベル
CM0	共用可能永続	• SYNC_SEND • SYNC_SEND_RECEIVE • SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT • SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT • SYNC_RECEIVE_CALLOUT (非管理コールアウト・プログラミング・モデルの場合)	自動生成および自動管理	確認
	専用永続	• SYNC_SEND • SYNC_SEND_RECEIVE • SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT • SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT	クライアント・アプリケーションによって指定 (clientID プロパティー)	確認

SYNC_RECEIVE_ASYNCOUTPUT 対話 Verb は非推奨になり、SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT 対話 Verb に置き換えられました。

メッセージのリトリーブ

共用可能永続ソケットで CM0 の対話を実行する Java クライアントに配信できない出力メッセージは、キューに入れて後でリトリーブできます。 2 次出力は常に CM0 出力として扱われ、再キューイングされて後でリトリーブできます。

共用可能永続ソケットで CM1 または CM0 の対話によってプログラム間通信が実行される状況について考えてみます。両方のプログラムが出力を返す場合、2 番目に返される出力メッセージが CM0 出力として扱われます。そのメッセージは再びキューに入れられ、後でリトリーブできます。

共用可能永続ソケットでの CM1 または CM0 の対話によってプログラム間通信が作成され、その通信によって別の CM0 の対話が起動されたことで 2 次出力 (常に CM0) が返された場合、その出力は再びキューに入れられ、後でリトリーブできます。

IMS OTMA 非同期保留キューまたは TPIPE でキューに入れられた配信されなかった出力メッセージが、代替クライアント ID を使用してリトリーブされない場合、元の対話とリトリーブは、同じソケット接続を使用して同じクライアント・アプリケーション内で起動する必要があります。元の対話と、その対話からの非同期出力のリトリーブの両方に、同じ生成済みクライアント ID (共用可能ソケット接続および関連する OTMA TPIPE を識別するクライアント ID) を使用する必要があります。

共用可能永続ソケットでは、配信されなかった出力メッセージはいくつかの方法で処理できます。

- 非同期出力メッセージを含む OTMA 保留キューの名前を IMSInteractionSpec オブジェクトの代替クライアント ID プロパティー値として指定することにより、配信不能な出力をリトリーブします。
- 配信されなかった出力をパージします。配信されなかった出力メッセージをパージするには、IMSInteractionSpec の purgeAsyncOutput プロパティーの値を `true` に設定する必要があります。この入力プロパティーが、配信されなかった入出力 PCB 出力を IMS Connect がパージするかどうかを決定します。purgeAsyncOutput プロパティーは、SYNC_SEND_RECEIVE 対話 Verb でのみ有効です。このプロパティーが SYNC_SEND_RECEIVE に指定されていない場合、デフォルトは `true` になります。
- メッセージを別の宛先に転送します。配信されなかった出力メッセージを別の宛先に転送するには、IMSInteractionSpec の reRoute プロパティーを `true` に設定します。このプロパティーは、SYNC_SEND_RECEIVE 対話 Verb および SYNC_SEND 対話 Verb でのみ有効です。reRoute プロパティーが `true` に設定されている場合、配信されなかった出力メッセージは指定された宛先 (通常はクライアント・アプリケーションが指定) でキューに入れられます。この宛先は、IMSInteractionSpec の reRouteName プロパティーで指定します。reRoute プロパティーが `true` に設定されていて、reRouteName が指定されない場合、reRouteName プロパティーの値は、IMS Connect 構成ファイルで指定された値になります。IMS Connect 構成ファイルで値が指定されていない場合は、デフォルト値の HWS$DEF が使用されます。

共用可能永続ソケット接続

共用可能永続ソケット接続は、以下のカスタム・プロパティーに必要な値を使用して IMS 接続ファクトリーによって作成されます。

- ホスト名: IMS Connect を実行しているシステムの TCP/IP ホスト名
- ポート番号 : 関連するポート番号
- データ・ストア名: ターゲット IMS の名前
- CM0 専用: `false`

CM0Dedicated プロパティーのデフォルト値は `false` です。これにより、CM0Dedicated プロパティーが明示的に `true` に設定されていない限り、接続ファクトリーによって共用可能永続ソケット接続が作成されます。

関連タスク:

967 ページの『共用可能永続ソケット接続の確立』

永続ソケットの解放および再接続

IMS TM リソース・アダプターと IMS Connect との間の TCP/IP 接続は永続的なものであり、エラーが発生しない限り、共用可能永続ソケット接続および専用永続ソケット接続のいずれの場合も開いたままになります。

IMS TM リソース・アダプターまたは IMS Connect のいずれかでエラーが発生した場合、または WebSphere Application Server が接続プールの管理のためにソケットを切断した場合は、ソケットが閉じられることがあります。

専用永続ソケット接続の場合、接続の確立に使用されたものと同じクライアント ID を持つ対話のみが、ソケット接続を使用できます。専用永続ソケット接続では対話に新しいクライアント ID が使用されると、ソケット接続の数が増えていきます。

WebSphere Application Server 管理コンソールで最大接続プロパティーと接続タイムアウト・プロパティーの両方をゼロ以外の値に設定した場合、最大接続の値に達して、すべての接続が使用中であると、接続タイムアウトの経過後にアプリケーションは ConnectionWaitTimeoutException 例外を受け取ります。この動作は、WebSphere Application Server の標準動作です。ConnectionWaitTimeoutException は、専用永続ソケットと共用可能永続ソケットの両方に適用されます。

ただし、最大接続の値に達していて、永続ソケット接続のいずれかが使用されていない場合、WebSphere Application Server は、新しい永続ソケット接続の作成要求に応えるために、そのソケットを切断します。 WebSphere Application Server のこの標準動作は、専用永続ソケットと共用可能永続ソケットの両方に適用されます。

ヒント: WebSphere Application Server には、プールの管理および永続セッションの再配分に役立つ接続プール設定がいくつか用意されています。物理接続が破棄されるまでの間隔 (秒単位) を指定するには、経過時間タイムアウト設定を使用します。

重要: コミット・モード 0 (CM0 またはコミット後送信) トランザクションで、接続の管理にアプリケーション・サーバーを使用しない (非管理対象 環境と呼ばれます) 場合、ユーザー自身が接続プーリングを管理する必要があります。 CM0 トランザクションはリカバリー可能であるため、IMS Connect は CM0 を使用するクライアントごとに別個の TPIPE を作成します。接続プールの管理にアプリケーション・サーバーを使用しない場合、多くの TPIPE が作成され、最終的にシステムが過負荷になります。

ソケット再接続

IMS TM リソース・アダプター にはソケット再接続機能があります。この機能は、IMS Connect への要求の送信プロセスまたは IMS Connect からの応答の受信プロセス中にいずれかの接続で通信上の問題が発生した場合に、接続プール内の失効した接続の再確立を試みます。

要求の受信時に、IMS TM リソース・アダプターが失効した接続を検出すると、アダプターは例外をスローします。次の要求が到着し、その失効した接続を使用しようとすると、IMS TM リソース・アダプターは IMS Connect が稼働状態になっているかどうかを確認します。 IMS Connect が稼働状態であれば、IMS TM リソース・アダプターは再接続を行ってから対話要求をサブミットします。接続が復元されると、ICO0140I 通知メッセージがログに記録されます。

このソケット再接続機能を使用すると、一時的なネットワーク接続問題が発生したときの、IMS TM リソース・アダプターのフォールト・トレランスが向上します。さらに重要な点は、この機能を使用することで、クライアント・アプリケーションから IMS TM リソース・アダプターの対話を再サブミットしなくても、システム保守の一環として IMS Connect をリサイクルできることです。

IMS TM リソース・アダプターと WebSphere Application Server の間でシスプレックス・ディストリビューターが使用される場合、WebSphere Application Server 接続プールに、複数の IMS Connect に接続された接続が含まれる可能性があります。そのいずれかの IMS Connect インスタンスがリサイクルされる場合、新しい要求は他の IMS Connect インスタンスに送信されます。新しい要求がリサイクルされた IMS Connect に送られるようになるのは、その IMS Connect インスタンスが復帰した場合、および他の IMS Connect インスタンスがワークロードを処理できなくなった場合のみです。

IMS Connect インスタンスの間でのワークロードの再配分に役立つよう、WebSphere Application Server 接続プール設定の経過時間タイムアウト・プロパティーを使用できます。期限切れの接続は破棄されるため、新しい接続を利用して、リサイクルされた IMS Connect インスタンスへの接続を再配分できます。

コールアウト要求では、IMS TM リソース・アダプターは IMSActivationSpec の retryLimit プロパティーと retryInterval プロパティーの値に基づいて再接続を試みます。retryLimit プロパティーは、IMS TM リソース・アダプターが IMS Connect への再接続 (接続が失われた場合)、または IMS への再接続 (IMS データ・ストアへの接続が失われた場合) を試みる最大回数を指定します。retryInterval プロパティーは、次に IMS Connect の状況確認を試みるまでの遅延時間を指定します。

関連情報:

WebSphere Application Server V8 の資料にある経過時間タイムアウトの情報

IMSInteractionSpec プロパティー構成

IMS と対話するには、Java アプリケーションで構成済み IMSInteractionSpec オブジェクトを提供する必要があります。

IMSInteractionSpec プロパティーの値は、IMS との対話についての記述 (例えば、IMS TM リソース・アダプターが IMS Connect からの応答を待つ最大時間や、対話のコミット・モードと同期レベル) を行います。

IMSInteractionSpec プロパティーの値は、Common Client Interface を使用するアプリケーションによって、`set` メソッドを使用して直接指定できます。これらの値は、アプリケーションのコードを生成する IDE のウィザードにも渡されます。

一部のプロパティーは入力専用プロパティーであり、一部は入力と出力のプロパティー、一部は出力専用プロパティーです。入力専用プロパティーでは IMS TM リソース・アダプターに入力が提供され、それらの値が変更されることはありません。出力専用プロパティーは、対話に関する追加情報を特定するために Java アプリケーション (アプリケーション・コンポーネントとも呼ばれます) によって問い合わせできます。

関連資料:

1063 ページの『IMS 相互作用仕様プロパティー』

IMS へのコマンド送信

IMS TM リソース・アダプターは、基本的にユーザーがホスト IMS システムで Java EE アプリケーションを介してトランザクションを実行するためのものですが、Java アプリケーションから IMS OTMA がサポートする IMS コマンドを発行することもできます。

IMS TM リソース・アダプターは、ホスト製品である IMS Connect を使用して IMS にアクセスします。IMS Connect はシステム間カップリング・ファシリティー (XCF) を使用して OTMA 経由で IMS にアクセスします。

IMS OTMA インターフェースを通じてサブミットできる IMS コマンドは特定のものに限られます。IMS TM リソース・アダプターは OTMA を通じて IMS にアクセスするので、OTMA でサポートされる IMS コマンドは、IMS TM リソース・アダプターを使用するアプリケーションによって IMS にサブミットできるコマンドのみです。

IMS コマンドでは、1 つ以上のデータ・セグメントで構成されるメッセージが出力されます。一部の IMS コマンドでは DFS メッセージが出力されます。例えば、ほとんどの /START コマンドの出力は、通常、メッセージ DFS058I `START COMMAND COMPLETED` です。他の IMS コマンドでは DFS メッセージは返されません。例えば、/DISPLAY コマンドでは、表示情報の行を表す複数のデータ・セグメントが返されます。両方のタイプの出力を同じものとして扱うには、IMSInteractionSpec クラスの imsRequestType プロパティーを 2 (IMS_REQUEST_TYPE_IMS_COMMAND) に設定する必要があります。この値は、IMS TM リソース・アダプターに対し、対話が IMS コマンドであること、および DFS メッセージを Java 例外ではなく通常の出力として扱うことを示します。

IMS TM リソース・アダプターを使用するアプリケーションによって IMS にサブミットできるコマンドは IMS コマンド解説書に記載されています。

IMS 接続ファクトリーの構成

アプリケーションで IMS 接続ファクトリーを使用して、IMS TM リソース・アダプターと対話します。IMS 接続ファクトリーを介して、IMS Transaction Manager (IMS TM) への事前構成済み接続を作成します。

IMS TM リソース・アダプターと IMS Connect との間の接続は、管理対象でも非管理対象でもかまいません。

管理対象接続 は、Java EE アプリケーション・サーバー (WebSphere Application Server など) で Connection Manager によって管理される接続です。管理対象環境では、アプリケーションは EIS 接続を作成する必要はありません。代わりに、Java EE Connection Manager からの接続を要求します。ただし、非管理対象接続 は、IMS TM リソース・アダプターからアプリケーションによって直接取得されます。

TCP/IP ソケットを使用して、IMS TM リソース・アダプターを IMS Connect と接続します。アプリケーションは、Java EE Connection Architecture (JCA) 接続ファクトリーを使用して、IMS Connect への接続を取得します。

アプリケーションが管理接続を使用する場合、IMS TM リソース・アダプターと IMS Connect との間の接続はすべて、永続的です。接続は、対話ごとに開いたり閉じたりしません。そうではなく、接続は開いたままにしておくことができ、複数の対話が順次に再使用することができます。これにより、CPU およびメモリー・リソースの使用効率が高くなります。Java EE アプリケーション・サーバーの管理者は、デプロイメント時に接続ファクトリーを構成します。

非管理対象接続の開閉は、それを使用するアプリケーションが行う必要があります。アプリケーションは、その実行時に必要な接続ファクトリーを作成し、構成する必要があります。

推奨事項: 管理対象接続を使用し、JNDI 検索を使用して適切な接続ファクトリーへの参照を取得してください。

IMS TM リソース・アダプターは、IMS Connect への永続接続のみをサポートします。管理対象環境では、永続接続は、アプリケーション・コンポーネントによって順次に再使用され、使用のたびに切断して再接続する必要はありません。TCP/IP ソケット接続の使用時に、アプリケーション・コンポーネントは、IMS TM リソース・アダプターと IMS Connect との間で専用永続ソケット接続または共用可能永続ソケット接続のどちらも使用できます。共用可能永続ソケット接続では、IMS TM リソース・アダプターが生成するクライアント ID が使用されますが、この ID は、共用可能ソケットを使用するアプリケーションからは見えません。

推奨事項: 専用ソケット接続またはローカル・オプション接続を使用する必要がある指定変更がある場合を除き、常に、共用可能 TCP/IP ソケットを使用してください。専用永続ソケット接続では、アプリケーションが指定するクライアント ID が使用されます。ローカル・オプション接続を使用する場合は、共用可能永続接続のみがサポートされます。

IMS Connect への TCP/IP 接続

TCP/IP は、分散環境および z/OS 環境の両方で IMS TM リソース・アダプターと IMS Connect の間の接続に使用することができます。

ご使用のアプリケーションが分散プラットフォーム上にある場合、IMS TM リソース・アダプターと IMS Connect の間の接続は TCP/IP 接続でなければなりません。例えば、IMS TM リソース・アダプターが Windows、AIX、Solaris、Linux、Linux for System z、または HP-UX 上の WebSphere Application Server にインストールされている場合、IMS TM リソース・アダプターは TCP/IP 接続を使用して IMS Connect と接続する必要があります。

IMS Connect との TCP/IP 接続は、クライアント *ID* と呼ばれる ID に関連付けられています。IMS Connect では、それ自体へのすべてのソケット接続で、それらのクライアント ID の固有性が確保されます。

IMS Connect への TCP/IP 接続は、以下の 2 つのタイプのいずれかです。

- 専用永続ソケット接続
- 共用可能永続ソケット接続

クライアント ID は、この 2 つのタイプのソケット接続ではそれぞれ異なる方法で決定されます。

関連概念:

959 ページの『共用可能永続ソケット』

956 ページの『専用永続ソケット』

専用永続ソケット接続の確立

WebSphere Application Server を使用して、専用永続ソケット接続を作成する J2C 接続ファクトリーを構成します。

専用永続ソケット接続を確立するには、以下のようにします。

1. WebSphere Application Server 管理コンソールで、J2C 接続ファクトリーを構成します。
2. 接続ファクトリーの CM0Dedicated プロパティーに値 `TRUE` を指定します。管理接続の場合、アプリケーションは、JNDI を使用してこの接続ファクトリーを見つけます。
3. アプリケーションで、IMSInteractionSpec クラスの commitMode プロパティーを 0 に設定します。
4. 接続の取得に使用する IMSConnectionSpec クラスの clientID プロパティーに値を指定します。アプリケーションが生成されたコードを使用する場合、clientID プロパティーをデータとして公開すると、この値を設定できます。アプリケーションが IMS TM リソース・アダプターの Common Client Interface (CCI) を使用する場合は、getConnection メソッドで渡される IMSConnectionSpec オブジェクトの setClientID メソッドを使用してこの値を設定します。

関連概念:

956 ページの『専用永続ソケット』

共用可能永続ソケット接続の確立

WebSphere Application Server を使用して、共用可能永続ソケット接続を作成する J2C 接続ファクトリーを構成します。

共用可能永続ソケット接続を確立するには、以下のようにします。

1. WebSphere Application Server 管理コンソールで、J2C 接続ファクトリーを構成します。
2. 接続ファクトリーの CM0Dedicated プロパティーに値 `FALSE` を指定します。管理接続の場合、アプリケーションは、JNDI を使用してこの接続ファクトリーを見つけます。
3. 共用可能永続ソケット接続の場合、クライアント ID は、ユーザーに代わり、IMS TM リソース・アダプター によって自動的に設定されます。アプリケーションが生成されたコードを使用する場合、共用可能永続ソケット接続はその生成されたコードによって取得されます。アプリケーションが IMS TM リソース・アダプターの Common Client Interface を使用する場合は、getConnection メソッドで渡される IMSConnectionSpec オブジェクトの clientID プロパティーに値を指定しないでください。

関連概念:

959 ページの『共用可能永続ソケット』

IMS 接続ファクトリー

IMS TM リソース・アダプター接続ファクトリーのプロパティーは、ターゲットのエンタープライズ情報システム (EIS) の特性と一致するように正しく構成する必要があります。

IMS サービス定義を作成する際、または WebSphere Application Server に対して IMS 接続ファクトリーを定義する際 (管理対象接続) に、IMS TM リソース・アダプターと IMS Connect の間の接続の特定のプロパティーに値を指定する必要があります。接続ファクトリーは、EIS への接続に必要な情報 (例えば、ホスト名やデータ・ストア名) を提供します。セキュリティー設定と接続タイプによっては、他の接続情報が必要になる場合があります。

非管理対象接続を使用する必要がある場合は、IMSConnectionSpec クラスが接続プロパティーを設定するための API を提供します。

関連タスク:

895 ページの『WebSphere Application Server での接続ファクトリーの作成』

関連資料:

1059 ページの『IMS 接続ファクトリーのプロパティー』

入出力メッセージのフォーマット

IMS TM リソース・アダプターを使用するアプリケーションを開発する際は、ターゲット・ホスト・アプリケーションと Java クライアントが実行されるプラットフォーム、およびご使用の入出力メッセージが持つ特性について考慮する必要があります。

作成する Java アプリケーションは、エンタープライズ情報システム (EIS) プラットフォームとは異なるプラットフォーム (例えば、Windows、AIX、Solaris) で実行されます。 IMS Transaction Manager (IMS TM) は、z/OS 上の IMS アプリケーション・プログラムです。プラットフォームに違いがあるため、メッセージのテキスト・データを、クライアント側の UNICODE のテキスト・データから IMS アプリケーションによって使用されている EBCDIC のテキスト・データに変換する必要があります。

ターゲット・ホスト・アプリケーションが実行されるプラットフォームのエンディアン値および IMS ホスト・システムによって使用されているコード・ページによっては、フォーマットの変換が必要になる場合があります。デフォルトは米国英語 (037) です。

推奨事項**:** フォーマット変換の処理は、統合開発環境 (IDE) によって行ってください。例えば、さまざまな WebSphere または Rational 開発環境の J2C ウィザードを使用して、インポートされた C、COBOL、および PL/I データ構造から Java データ・バインディングを作成します。データ・バインディングでは、実行時にメッセージのすべてのフォーマット変換が行われます。これらのデータ・バインディングは、インポーター・ウィザードで選択したオプションに基づいて作成されます。

IMS では必ず、すべてのトランザクション・メッセージに接頭部として 2 バイトの LL フィールド、2 バイトの ZZ フィールドを付け、その後にトランザクション・コードを続けます。LL フィールドはメッセージの長さを指定します。 ZZ は 2 進ゼロのフィールドです。IMS への入力メッセージを作成する際は、LL 値を入力メッセージの長さに設定し、ZZ を 0 に設定する必要があります。

複数セグメントおよび可変長の出力

IMS アプリケーションが複数セグメントの出力、または可変長の出力を返す場合、COBOL 定義 OUTPUT-MSG を使用してトランザクションの出力を定義するか、トランザクションの出力用の出力メッセージを作成できます。

次の COBOL アプリケーションでは、IMS によって返される出力メッセージは 3 つの固定長セグメントで構成されます。この IMS アプリケーションによって返される出力メッセージの合計サイズは、固定サイズの 99 バイトであり、COBOL 01 構造 OUTPUT-MSG によって表されます。長さは、OUTPUT-SEG1 が 16 バイト、OUTPUT-SEG2 が 31 バイト、OUTPUT-SEG3 が 52 バイトです。

```
LINKAGE SECTION.

        01  INPUT-MSG.
            02  IN-LL          PICTURE S9(3) COMP.
            02  IN-ZZ          PICTURE S9(3) COMP.
            02  IN-TRCD        PICTURE X(5).
            02  IN-DATA1       PICTURE X(6).
            02  IN-DATA2       PICTURE X(6).

        01  OUTPUT-MSG.
            02  OUT-ALLSEGS  PICTURE X(99) VALUE SPACES.

        01  OUTPUT-SEG1.
            02  OUT-LL       PICTURE S9(3) COMP VALUE +0.
            02  OUT-ZZ       PICTURE S9(3) COMP VALUE +0.
            02  OUT-DATA1    PICTURE X(12) VALUE SPACES.

        01  OUTPUT-SEG2.
            02  OUT-LL       PICTURE S9(3) COMP VALUE +0.
            02  OUT-ZZ       PICTURE S9(3) COMP VALUE +0.
            02  OUT-DATA1    PICTURE X(13) VALUE SPACES.
            02  OUT-DATA2    PICTURE X(14) VALUE SPACES.

        01  OUTPUT-SEG3.
            02  OUT-LL       PICTURE S9(3) COMP VALUE +0.
            02  OUT-ZZ       PICTURE S9(3) COMP VALUE +0.
            02  OUT-DATA1    PICTURE X(15) VALUE SPACES.
            02  OUT-DATA2    PICTURE X(16) VALUE SPACES.
            02  OUT-DATA3    PICTURE X(17) VALUE SPACES.
```

J2C ウィザードを使用すると、以下のようにして、IMS トランザクションを実行する J2C Bean を作成できます。

- IMS アプリケーションから返される出力メッセージを保管するメッセージ・バッファー・クラスを作成します。
- 入力メッセージと出力メッセージの両方について COBOL から Java へのマッピングを行うための COBOLファイルをインポートします。
  - 入力メッセージ構造 (COBOL 01 構造 INPUT-MSG によって表されます) を選択して、入力バインディング操作を作成します。

– データ・マッピング・ウィザード (「**File**」 > 「**New**」 > 「**Other**」 > 「**CICS/IMS Java Data Binding**」を選択して使用できます) を使用して、出力メッセージのセグメントの出力バインディング操作を作成します。各セグメントで、データ・バインディング・ウィザードを使用して、データ構造として正しいセグメントを選択し、コード・ページ、エンディアン名、Quote 名、Trunc 名などのパラメーターを指定します。

- IMS トランザクションを実行する J2C Bean メソッドを呼び出して、IMS トランザクションによって返されたデータのバッファーから出力セグメントにデータを取り込みます。

```
package sample.ims;

import com.ibm.etools.marshall.util.MarshallIntegerUtils;
import sample.ims.data.*;

public class TestMultiSeg
{
 public static void main (String[] args)
 {
  byte[] segBytes = null;
  int srcPos = 0;
  int dstPos = 0;
  int totalLen = 0;
  int remainLen = 0;
  byte[] buff;
  short LL = 0;
  short ZZ = 0;

  try
  {
   // ------------------------------------------------
   // Populate the IMS transaction input message with
   // data.  Use the input message format handler method
   // getSize() to set the LL field of the input message.
   // ------------------------------------------------
   InputMsg input = new InputMsg();
   input.setIn__ll((short) input.getSize());
   input.setIn__zz((short) 0);
   //--------------------------------------------
   // find out the transaction code from your IMS
   // administrator
   //--------------------------------------------
   input.setIn__trcd("SKS6 ");
   input.setIn__data1("M2 SI1");
   input.setIn__data2("M3 SI1");

   // ------------------------------------------------
   // Run the IMS transaction.  The multi-segment output
   // message is returned.
   // ------------------------------------------------
   MSOImpl proxy = new MSOImpl();

   sample.ims.CCIBuffer output = proxy.runMultiSegOutput(input);

   // ------------------------------------------------
   // Retrieve the multi-segment output message as a
   // byte array using the output message format
   // handler method getBytes().
   // ------------------------------------------------
   System.out.println(
    "\nSize of output message is: " + output.getSize());
   segBytes = output.getBytes();

   srcPos = 0;
```

```
dstPos = 0;
totalLen = segBytes.length;
remainLen = totalLen;

// ------------------------------------------------
// Populate first segment object from buffer.
// ------------------------------------
buff = null;
// Get length of segment.
LL =
 MarshallIntegerUtils.unmarshallTwoByteIntegerFromBuffer(
  segBytes,
  srcPos,
  true,
  MarshallIntegerUtils.SIGN_CODING_TWOS_COMPLEMENT);

// Put segment in byte array.
buff = new byte[LL];
System.arraycopy(segBytes, srcPos, buff, dstPos, LL);
remainLen -= LL;

// Create and populate segment object from byte array.
OutputSeg1 S1 = new OutputSeg1();
S1.setBytes(buff);
System.out.println(
 "\nOutSeg1 LL is:    "
  + S1.getOut__ll()
  + "\nOutSeg1 ZZ is:    "
  + S1.getOut__zz()
  + "\nOutSeg1_DATA1 is: "
  + S1.getOut__data1());

// ------------------------------------------------
// Populate second segment object from buffer.
// ------------------------------------------------
srcPos += LL;
buff = null;
// Get length of segment.
LL =
 MarshallIntegerUtils.unmarshallTwoByteIntegerFromBuffer(
  segBytes,
  srcPos,
  true,
  MarshallIntegerUtils.SIGN_CODING_TWOS_COMPLEMENT);

// Put segment in byte array.
buff = new byte[LL];
System.arraycopy(segBytes, srcPos, buff, dstPos, LL);
remainLen -= LL;

// Create and populate segment object from byte array.

OutputSeg2 S2 = new OutputSeg2();
S2.setBytes(buff);
System.out.println(
 "\nOutSeg2 LL is:    "
  + S2.getOut__ll()
  + "\nOutSeg2 ZZ is:    "
  + S2.getOut__zz()
  + "\nOutSeg2_DATA1 is: "
  + S2.getOut__data1()
  + "\nOutSeg2_DATA2 is: "
  + S2.getOut__data2());
// ------------------------------------------------
// Populate third segment object from buffer.
// ------------------------------------------------
srcPos += LL;
```

```
   buff = null;
   // Get length of segment.
   LL =
    MarshallIntegerUtils.unmarshallTwoByteIntegerFromBuffer(
     segBytes,
     srcPos,
     true,
     MarshallIntegerUtils.SIGN_CODING_TWOS_COMPLEMENT);

   // Put segment in byte array.
   buff = new byte[LL];
   System.arraycopy(segBytes, srcPos, buff, dstPos, LL);
   remainLen -= LL;

   // Create and populate segment object from byte array.
   OutputSeg3 S3 = new OutputSeg3();
   S3.setBytes(buff);
   System.out.println(
    "\nOutSeg3 LL is:    "
     + S3.getOut__ll()
     + "\nOutSeg3 ZZ is:    "
     + S3.getOut__zz()
     + "\nOutSeg3_DATA1 is: "
     + S3.getOut__data1()
     + "\nOutSeg3_DATA2 is: "
     + S3.getOut__data2()
     + "\nOutSeg3_DATA3 is: "
     + S3.getOut__data3());
  }
  catch (Exception e)
  {
   System.out.println("\nCaught exception is: " + e);
  }
 }
}
```

可変長メッセージ、複数セグメントのメッセージ、および配列を含むメッセージについては、WebSphere または Rational 開発環境のオンライン・ヘルプまたはインフォメーション・センターにある IMS チュートリアルとサンプルを参照してください。

- IMS チュートリアルは、「**Tutorials**」 > 「**Do and Learn**」から利用できます。
- IMS サンプルは、「**Samples**」 > 「**Technology samples**」 > 「**J2C samples**」から取得できます。

これらのチュートリアルとサンプルは、特殊なメッセージ・フォーマットの処理に関するガイダンス情報とサンプル・コードを示します。

関連情報:

Rational Application Developer V9 の Knowledge Center にあるチュートリアル

Rational Application Developer V9 の Knowledge Center にあるサンプル

IMS Transaction Manager との対話の保護

IMS Connect との対話、および IMS との対話を保護するには、 IMS TM リソース・アダプターのさまざまなセキュリティー機能を使用します。

IMS TM リソース・アダプターのセキュリティー

Java EE コネクター・アーキテクチャー (JCA) では、アプリケーション・サーバーとエンタープライズ情報システム (EIS) 内の情報を連携させて、認証済みのユーザーのみが EIS にアクセスできるようにする必要があることを指定しています。

JCA セキュリティー・アーキテクチャーでは、Java EE ベースのアプリケーションのエンドツーエンド・セキュリティー・モデルが拡張され、EIS との統合を含まれるようになっています。 IMS TM リソース・アダプターは Java EE コネクター・アーキテクチャーのセキュリティー・アーキテクチャーに従っており、WebSphere Application Server Java 2 セキュリティー・マネージャーと連動します。

EIS サインオン

JCA セキュリティー・アーキテクチャーは、EIS 固有のユーザー IDとパスワードの認証メカニズムをサポートしています。ターゲット EIS へのサインオンに使用されるユーザー ID とパスワードは、アプリケーション・コンポーネント (コンポーネント管理サインオン)、またはアプリケーション・サーバー (コンテナー管理サインオン) のいずれかによって提供されます。

IMS TM リソース・アダプターの場合、IMS がターゲット EIS です。アプリケーション・コンポーネントまたはアプリケーション・サーバーにより提供されるセキュリティー情報は、IMS TM リソース・アダプターに渡されます。次に IMS TM リソース・アダプターはこの情報を IMS Connect に渡します。 IMS Connect は、この情報を使用してユーザー認証を実行し、その情報を IMS OTMA に渡します。 IMS OTMA は次に、この情報を使用して特定の IMS リソースにアクセスするための許可を検証します。

標準的な環境では、IMS TM リソース・アダプターは、受け取ったセキュリティー情報 (ユーザー ID、パスワード、およびオプションのグループ名) を、IMS OTMA メッセージに組み込んで IMS Connect に渡します。IMS Connect はその後、そのセキュリティー構成に応じてホスト上のセキュリティー許可機能 (SAF) を呼び出すことがあります。

- TCP/IP を使用し、コンポーネント管理サインオンまたはコンテナー管理サインオンのいずれかを使用している、分散プラットフォームまたは z/OS 上の WebSphere Application Server の場合、以下のようになります。
  - IMS Connect 構成メンバーで RACF=Y が設定されている場合、または IMS Connect コマンド SETRACF ON が発行されている場合、IMS Connect は SAF を呼び出して、OTMA メッセージ内で IMS TM リソース・アダプターによって渡されたユーザー ID とパスワードを使用して認証を行います。認証が成功すると、SAF への IMS Connect 呼び出しから返されたユーザー ID、オプションのグループ名、および UTOKEN が IMS OTMA に渡され、IMS リソースにアクセスするための許可が検証されます。
  - IMS Connect 構成メンバーで RACF=N が設定されている場合、または IMS Connect コマンド SETRACF OFF が発行されている場合、IMS Connect は SAF を呼び出しません。ただし、ユーザー ID とグループ名 (指定されている場合) が IMS OTMA に渡され、IMS リソースにアクセスするための許可が行われます。

- ユーザー ID は、以下の 2 つの方法でアプリケーション・サーバーに渡すことができます。
  - ユーザー ID とパスワードは、Java 認証・承認サービス (JAAS) 別名に入れて渡すことができます。JAAS 別名は、IMS にアクセスするアプリケーションによって使用される接続ファクトリーと関連付けられるか、または WebSphere Application Server のバージョンによっては、アプリケーションによって使用される EJB リソース参照と関連付けられます。アプリケーション・サーバーは、ユーザー ID を表すユーザー・トークンを作成し、別名に入れて IMS TM リソース・アダプターに渡します。
  - WebSphere Application Server for z/OS は、アプリケーションの実行スレッドに関連付けられたユーザー ID を取得するように構成できます。アプリケーション・サーバーは、このユーザー ID を表すユーザー・トークンを作成して IMS TM リソース・アダプターに渡します。

IMS が実行する許可検査のレベルは、IMS コマンド /SECURE OTMA によって制御されます。

分散ネットワーク・セキュリティー資格情報

IMS TM リソース・アダプターで、Java EE アプリケーションから入力されたネットワーク・セキュリティー資格情報を Java EE アプリケーションと IMS の間で受け渡しできるように、WebSphere Application Server または WebSphere Liberty を構成できます。ネットワーク・セキュリティー資格情報には、ネットワーク・セッション ID とネットワーク・ユーザー ID を含めることができます。分散ネットワーク・セキュリティー資格情報は IMS TM リソース・アダプターによって IMS Connect に伝搬され、IMS Connect はその資格情報を、Java EE アプリケーションが IMS トランザクションにアクセスするときに IMS に渡します。

IMS TM リソース・アダプターでネットワーク・セキュリティー資格情報を Java EE アプリケーションから IMS に渡せるようにするには、IMS TM リソース・アダプターで提供される Java 認証・承認サービス (JAAS) ログイン・モジュールを構成し、ご使用のアプリケーションにリンクする必要があります。アプリケーションを JAAS ログイン・モジュールにリンクしたら、ユーザーは、外部ユーザー・アカウント・レジストリーによる認証を行うために IMS トランザクションを呼び出すときに、ユーザー自身のセキュリティー資格情報を入力する必要があります。外部ユーザー・アカウント・レジストリーとしては、WebSphere Application Server または WebSphere Liberty によってサポートされるいずれかのユーザー・アカウント・レジストリー (例えば、LDAP サーバー) が考えられます。資格情報が正常に認証されると、IMS TM リソース・アダプターは、OTMA メッセージ接頭部のセキュリティー・データ・セクションを使用して、その分散資格情報を IMS Connect に送信します。

また、IMS 従属領域で実行されている IMS アプリケーションが外部 Java EE アプリケーションに同期または非同期のコールアウト要求を行う際に、IMS TM リソース・アダプターがネットワーク・セキュリティー資格情報をサポートできるようにすることも可能です。

IMS TM リソース・アダプターによって抽出されるネットワーク・セッション ID は、デフォルトでは、使用されている認証サーバーの IP アドレスとポートです。ネットワーク・セッション ID の最大長は 254 バイトです。

IMS TM リソース・アダプターによって抽出されるネットワーク・ユーザー ID は、認証サーバーで定義されている識別名のいずれかです。ネットワーク・ユーザー ID の最大長は 246 バイトです。

Secure Sockets Layer (SSL) 通信

IMS TM リソース・アダプターと IMS Connect はユーザーが構成できます。これらは、適切に構成されていれば、TCP/IP の SSL プロトコルを使用して互いの間の通信を保護できます。

SSL 接続は、非 SSL の TCP/IP 接続より安全性が高く、IMS Connect サーバーの認証、およびオプションで IMS TM リソース・アダプター・クライアントの認証を行います。 SSL 接続を流れるメッセージも暗号化される場合があります。

Null 暗号化による SSL は、認証は行われるがメッセージは暗号化されないという中間レベルのセキュリティーを提供します。非暗号化 SSL 通信では、IMS TM リソース・アダプターと IMS Connect の間を流れる各メッセージを暗号化するために必要なオーバーヘッドが除去されるため、より高いスループットが実現します。

関連概念:

『コンテナー管理 EIS サインオン』

983 ページの『Secure Sockets Layer (SSL) サポート』

関連タスク:

976 ページの『コンテナー管理 EIS サインオンの構成』

コンテナー管理 EIS サインオン

コンテナー管理 EIS サインオンを使用した場合は、アプリケーション・サーバーのセキュリティー・マネージャーがアプリケーションのセキュリティー情報を管理します。

アプリケーションのデプロイメント記述子にディレクティブ `<res-auth>Container</res-auth>` を指定すると、コンテナー管理 EIS サインオンが使用されます。この場合、アプリケーション・サーバー (コンテナー) がセキュリティー情報 (ユーザー ID とパスワード) を提供します。

TCP/IP の場合、アプリケーション・サーバーは別名に含まれるセキュリティー情報を IMS TM リソース・アダプターに渡します。IMS TM リソース・アダプターは、そのセキュリティー情報を認証用に IMS Connect に渡します。IMS Connect はユーザーを認証し、そのセキュリティー情報を IMS へのサインオン用に渡します。IMS Connect がユーザーを認証できない場合、IMS TM リソース・アダプターにセキュリティー障害が返され、それを受けて IMS TM リソース・アダプターはアプリケーションに例外を返します。

コンテナー管理サインオンでは、アプリケーションが IMSConnectionSpec クラスのプロパティー userName、password、または groupName を使用して IMS TM リソース・アダプターにセキュリティー情報を渡した場合、その情報は無視されま

す。ただし、IMSConnectionSpec オブジェクトの他の情報 (例えば、コミット・モード 0 の対話でのクライアント ID) を渡した場合、その情報は IMS TM リソース・アダプターによって使用されます。

コンテナー管理 EIS サインオンの構成

WebSphere Application Server バージョン 6 以降では、コンテナー管理認証が、デプロイメント時のリソース参照マッピング上のログイン構成の仕様で置き換えられます。

Rational 開発環境または WebSphere 開発環境で設定したアプリケーション・デプロイメント記述子ファイル内の `<res-auth>Container</res-auth>` ディレクティブは、アプリケーションによって使用されるリソース参照の設定に置き換えられます。

コンテナー管理 EIS サインオンの構成は、以下の作業からなります。

1. EIS サインオンに使用されるユーザー ID とパスワードを提供するために使用される Java 認証・承認サービス (JAAS) 認証データ・エントリー別名の作成。
2. 適切な JAAS 認証別名を使用するための、アプリケーションまたは接続ファクトリーの構成。

コンテナー管理 EIS サインオン用に構成するには、次のようにします。

1. JAAS - J2C 認証データ・エントリー別名を作成し、ユーザー ID とパスワードを指定します。
   a. WebSphere Application Server 管理コンソールで、「**Security**」をクリックします。
   b. 「**Global security**」をクリックします。
   c. 「Global Security」ページの「Authentication」セクションの下で、「**Java Authentication and Authorization Service**」をクリックして展開し、さらに、「**J2C authentication data**」をクリックします。
   d. 「JAAS - J2C 認証データ」ページで、「**New**」をクリックします。
   e. 別名を指定し、IMS への接続時にアプリケーションがサインオンのために使用するユーザー ID とパスワードを指定します。

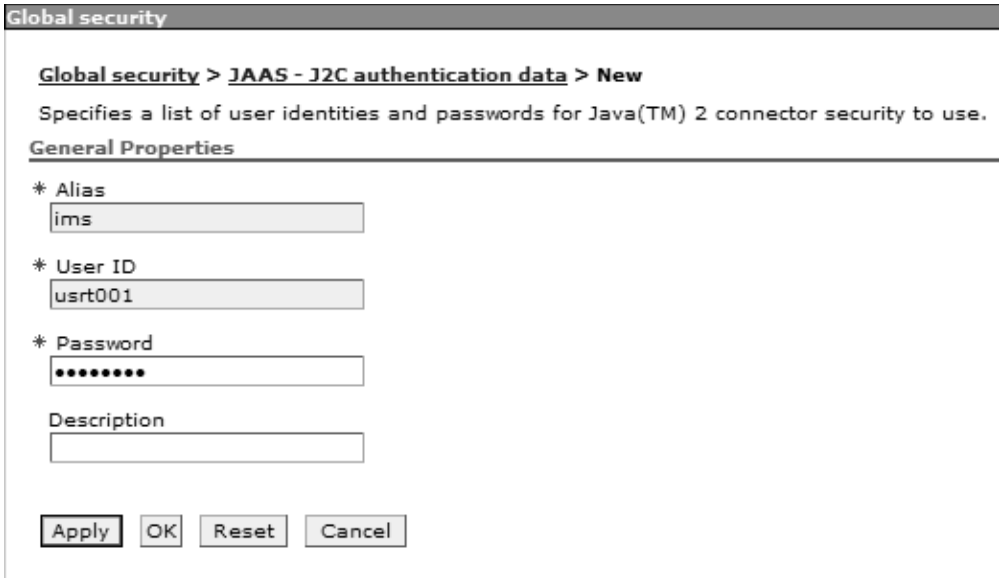


図 *126. J2C* 認証に使用するセキュリティー別名の指定

f. 「**OK**」をクリックし、ページの上部にある「**Save**」をクリックします。

2. 適切な JAAS 認証別名を使用するよう、アプリケーションまたは接続ファクトリーを構成します。
   - アプリケーションを構成するには、管理コンソールで以下を実行します。
     a. 「**Applications > Application Types**」 (例えば、「**WebSphere enterprise applications**」) をクリックします。
     b. アプリケーションのリストから、ご使用のアプリケーションの名前をクリックします。
     c. 「References」セクションで、「**Resource references**」をクリックします。
     d. 「**Modify Resource Authentication Method**」をクリックします。
     e. 認証方式を選択し、その認証方式が適用されるモジュールを表から選択します。

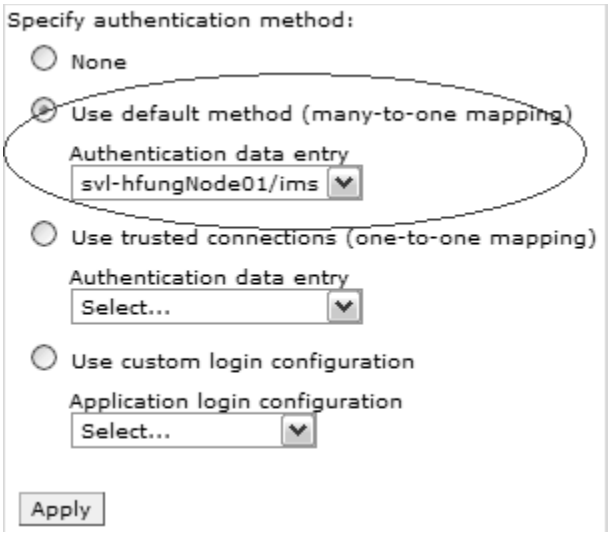


図 *127.* ご使用のアプリケーションのための認証方式の指定

ヒント**:** 認証が指定されない場合、データ・ソースまたは接続ファクトリー上の非推奨コンテナー管理認証別名が使用されます。オンライン・ヘルプを参考にして構成してください。

f. 「**Apply**」をクリックし、「**Save**」をクリックして変更を保管します。

- 接続ファクトリーを構成するには、管理コンソールで以下を実行します。
  a. 「**Resources**」 > 「**Resource adapters**」 > 「**J2C connection factories**」をクリックします。
  b. 接続ファクトリーの名前をクリックします。
  c. コンポーネント管理認証別名、またはコンテナー管理認証別名を指定します。

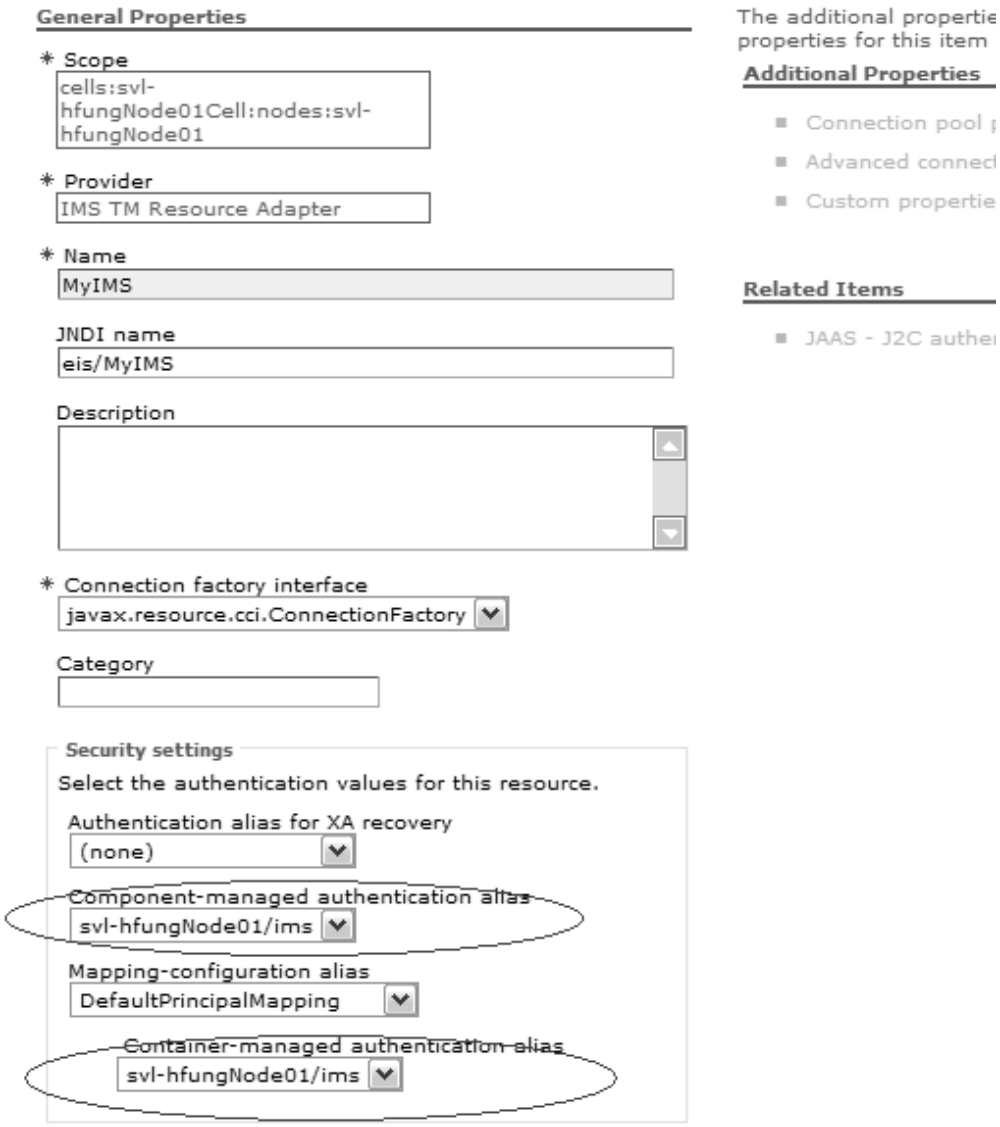


図 *128. J2C* 接続ファクトリーのためのセキュリティー設定および認証別名の指定

ヒント**:** コンテナー管理認証別名フィールドは、コンポーネント・リソース参照にログイン構成がない場合にのみ使用されます。オンライン・ヘルプを参考にして構成してください。

d. 「**OK**」をクリックし、ページの上部にある「**Save**」をクリックして変更を保管します。

ヒント**:** スタンドアロン型の WebSphere Application Server でのコンテナー管理サインオンを構成するプロセスは、単体テスト環境での WebSphere Application Server のプロセスと同じです。

関連情報:

管理コンソールでの Java EE コネクター接続ファクトリーの構成 (WebSphere Application Server バージョン 8)

J2C 接続ファクトリーの設定 (WebSphere Application Server バージョン 8)

リソース参照 (WebSphere Application Server バージョン 8)

コンポーネント管理 EIS サインオン

コンポーネント管理 EIS サインオンの場合、アプリケーションは IMS Connect および IMS に提供されるセキュリティー情報を管理します。通常、アプリケーションは、EIS サインオンに使用されるセキュリティー情報を提供します。

アプリケーションにコンポーネント管理 EIS サインオンを指定するには、そのアプリケーションのデプロイメント記述子のリソース参照で <res-auth> エレメントに値 `application` を入力します。

EIS サインオンに使用されるセキュリティー情報 (ユーザー ID、パスワード、およびオプションのグループ名) はアプリケーション (コンポーネント) が提供します。

- アプリケーションは、Java EE コネクター・アーキテクチャーの Common Client Interface (CCI) を使用する場合、以下のメソッドを使用してコンポーネント管理 EIS サインオンを実行します。
  - IMSConnectionSpec.setUserName()
  - IMSConnectionSpec.setPassword()
  - IMSConnectionSpec.setGroupName()

  これらのメソッドは、IMSConnectionSpec オブジェクトにセキュリティー情報を取り込みます。アプリケーションは IMS との接続を確立した後に、IMSConnectionSpec オブジェクトを IMSConnectionFactory.getConnection メソッドのパラメーターとして渡します。IMS TM リソース・アダプターは、このセキュリティー情報を IMS へのサインオン (認証および許可) で使用するために IMS Connect に渡します。
- アプリケーションが Rational または WebSphere 開発環境によって生成された場合、セキュリティー情報はアプリケーション入力データとして渡されます。セキュリティー情報を入力データとして渡すには、IMSConnectionSpec クラスのプロパティー userName、password、および groupName を公開する必要があります。

アプリケーションがセキュリティー情報を提供するメソッドをいずれも使用しない場合、WebSphere Application Server は J2C 接続ファクトリーのカスタム・プロパティーからセキュリティー情報を取得します。

ヒント: 接続ファクトリーのセットアップ時にコンポーネント管理の JAAS 別名を指定した場合、WebSphere Application Server の始動時にその別名のユーザー名とパスワードが接続ファクトリーのカスタム・プロパティーの userName 値と password 値をオーバーライドします。

コンポーネント管理 EIS サインオンの構成

アプリケーション開発時にコンポーネント管理 EIS サインオンを指定し、アプリケーションの配置時にそれを構成します。

Rational 開発環境または WebSphere 開発環境を使用して Java EE アプリケーションを作成する場合、コンポーネント管理またはコンテナー管理の EIS サインオンを選択できます。ご使用のアプリケーション・デプロイメント記述子のリソース参

照内の認証ディレクティブを「Application」に設定すると、ご使用のアプリケーションはコンポーネント管理 EIS サインオン用に構成されます。同様のステップが、他のリソースおよび他の IDE に使用されます。

以下の例は、Rational 開発環境または WebSphere 開発環境での EJB プロジェクトについてこの設定を検証または変更する方法を説明しています。

1. エレメントの値を「**Application**」に設定します。
   a. Java EE パースペクティブの「Project Explorer」ビューで、「**EJB Projects**」の「**EJB project**」を展開します。
   b. 「**Deployment Descriptor: *your_EJB_project***」を右クリックして、「**Open With**」 > 「**Deployment Descriptor Editor**」を選択します。
   c. 「EJB Deployment Descriptor」ビューで、「**References**」タブをクリックし、アプリケーションが使用する EJB コンポーネントの名前を展開して、EJB のリソース参照を選択します。EJB のリソース参照を選択すると、「EJB Deployment Descriptor」ビューの右側のフィールドが値とともに表示されます。
   d. 「Authentication」フィールドで「**Application**」をまだ選択していない場合は、ここで選択します。このフィールドは、`<res-auth>` エレメントにマップされます。
   e. EJB Deployment Descriptor Editor を閉じ、「**Yes**」をクリックして変更を保管します。EJB アプリケーションのデプロイメント記述子に以下のコードが追加されます。

      ```
      <res-auth>Application</res-auth>
      ```

   通常、コンポーネント管理サインオンでは、これ以上の構成は不要です。セキュリティー情報が、IMSConnectionSpec オブジェクトでアプリケーションによって提供されるためです。ただし、アプリケーションが IMSConnectionSpec オブジェクトを提供しない場合、または提供された IMSConnectionSpec オブジェクトにユーザー ID が指定されない場合、IMS TM リソース・アダプターは、アプリケーションが使用する接続ファクトリーからデフォルトのセキュリティー値を取得します。
2. アプリケーション・コンポーネントがユーザー ID を提供しない場合、または指定されたユーザー ID が NULL またはブランクである場合は、デフォルトのセキュリティー値が使用されます。デフォルト値は、接続ファクトリーで指定するものであり、以下の 2 つの方法で指定できます。
   - コンポーネント管理認証別名を指定する。
     a. コンポーネント管理認証別名を使用するには、JAAS 認証別名を定義する必要があります。
        1) 「Servers」ビューで、該当のサーバーを右クリックして、「**Run administrative console**」を選択します。
        2) 「Resources」を展開して、「**Resource Adapters**」を選択します。
        3) 変更するリソース・アダプターをクリックします。
        4) 「Additional Properties」の下で、「**J2C Connection factories**」をクリックします。

5) 「Related Items」で、「**J2C Authentication Data Entries**」をクリックします。
6) 別名のリストの上にある「**New**」をクリックします。
7) 別名、ユーザー ID、パスワード、およびオプションの説明を入力します。「**OK**」を選択します。

b. アプリケーションで使用される J2C 接続ファクトリーのコンポーネント管理認証別名プロパティー用の JAAS 認証別名を選択します。JAAS 認証別名の選択は、接続ファクトリーを最初に作成する際に、または後で接続ファクトリーを編集することにより、可能です。接続ファクトリーを編集するには、以下のステップを実行します。
1) サーバーの管理コンソールで、「**Resource Adapters**」 > 「***server_name***」 > 「**J2C connection factories**」 > 「***connection_factory_name***」を選択することにより、変更する接続ファクトリーにナビゲートします。
2) 「Component-managed Authentication Alias」ドロップダウン・リストで、その接続ファクトリーを使用するアプリケーションがコンポーネント管理認証に使用する JAAS 認証別名を選択します。
3) 「**OK**」を選択します。

コンポーネント管理認証別名に関連付けられているユーザー ID およびパスワードは、アプリケーション・サーバーの始動時に関連接続ファクトリーのカスタム・プロパティーのデフォルト値を設定 (または適用可能な場合はオーバーライド) するために使用されます。

- 接続ファクトリーのカスタム・プロパティーでデフォルト値を定義します。
  - ご使用の J2C 接続ファクトリーの「Component-managed Authentication Alias」フィールドに有効な JAAS 認証別名を割り当てない場合は、J2C 接続ファクトリーの「Custom Properties」ページの「userName」、「password」、および「groupName」の各フィールドに値を割り当てることができます。
  - 接続ファクトリーを作成するために、IMSConnectionSpec API を使用して、接続プロパティーを指定します。コンポーネント管理認証別名を使用する場合は、J2C 接続ファクトリーのカスタム・プロパティーで値を指定することをお勧めします。コンポーネント管理認証別名により、ユーザー ID とパスワードのセキュリティーが強化されます。これは、JAAS 認証別名のユーザー名およびパスワード値がサーバー管理者にのみ可視であるためです。

IDE のテスト環境でコンポーネント管理サインオンを構成するためのプロセスとスタンドアロン型 WebSphere Application Server でコンポーネント管理サインオンを構成するためのプロセスは似ています。

関連タスク:

895 ページの『WebSphere Application Server での接続ファクトリーの作成』

関連資料:

1059 ページの『IMS 接続ファクトリーのプロパティー』

Secure Sockets Layer (SSL) サポート

SSL は、IMS TM リソース・アダプターと IMS Connect 間の TCP/IP 接続を保護することにより、対話に対するセキュリティーを提供します。

SSL プロトコルを使用すると、インターネット上での機密情報の転送を安全に行うことができます。 SSL は、以下のものから情報を保護します。

- インターネットの盗聴
- データ盗難
- トラフィック分析
- データ変更
- トロイの木馬ブラウザー/サーバー

IMS TM リソース・アダプターは TCP/IP ソケットを介して IMS Connect と通信できます。IMS TM リソース・アダプターで TCP/IP が使用される場合は、SSL を使用してその 2 つのエンティティー間の TCP/IP 通信を保護できます。 IMS TM リソース・アダプターと IMS Connect によって提供される SSL サポートでは、公開鍵と秘密鍵の組み合わせを対称鍵暗号化方式とともに使用して、クライアントとサーバーの認証、データの機密性、および保全性が実現します。 SSL は TCP/IP 通信プロトコルの上の層で実行され、SSL を使用することで、SSL 対応サーバーがそれ自体を SSL 対応クライアントに対して認証でき、その逆の認証も可能になります。

IMS TM リソース・アダプターと IMS Connect との間の SSL 接続では、IMS TM リソース・アダプターがクライアントとみなされ、IMS Connect がサーバーとみなされます。認証が完了すると、サーバーとクライアントは暗号化された接続を確立でき、さらにそれによってデータの保全性も維持されます。

WebSphere Application Server 環境での SSL サポートの場合、IMS TM リソース・アダプターでは Java Secure Socket Extension の IBM による実装 (IBM JSSE) を使用します。SSL ライブラリーは WebSphere または Rational 開発環境内、および WebSphere Application Server 内に含まれています。

重要:

- IMS TM リソース・アダプターは、分散プラットフォーム (Linux for System z を含む) では X.509 証明書および JKS 鍵ストア・タイプのみをサポートし、z/OS では JKS 鍵ストア・タイプまたは RACF 鍵リングをサポートします。
- Transport Layer Security バージョン 1 (TLS V1) は、SSL 3.0 プロトコルの後継です。IMS TM リソース・アダプターは TLS V1 をサポートします。TLS 3.0 には以前の SSL 3.0 プロトコルとの互換性があります。

SSL の概念

SSL プロトコルには、証明書、認証局、証明書管理、鍵ストア、トラストストアなどの重要な概念が含まれます。

証明書
: デジタル証明書とは、証明書の所有者の ID を検証するデジタル文書です。

デジタル証明書には、個人に関する情報 (名前、会社、公開鍵など) が含まれます。証明書には、信頼できる管轄局である認証局 (CA) によりデジタル署名で署名されています。

認証局
: 認証局 (CA) とは、デジタル証明書を作成し、ユーザーおよびシステムに対して発行する信頼できる当事者です。CA は、有効な資格情報として、証明書の信頼の基盤を確立します。

証明書管理
: 証明書および秘密鍵は、鍵ストアと呼ばれるファイルに保管されます。鍵ストアは、鍵資料のデータベースです。鍵ストア情報は、鍵項目とトラステッド証明書項目という 2 つのカテゴリーにグループ分けすることができます。この 2 つの項目は、同じ鍵ストアに保管することも、セキュリティー目的で鍵ストアとトラストストアに別個に保管することもできます。鍵ストアおよびトラストストアは、SSL クライアントの IMS TM リソース・アダプターと、SSL サーバーの IMS Connect の両方が使用します。

鍵ストア
: 鍵ストアは、IMS TM リソース・アダプター や SSL クライアントの秘密鍵などの鍵項目を保持します。

トラストストア
: トラストストアは、ユーザーが信頼する証明書のみを保持する鍵ストアです。トラストストアへの項目の追加は、ユーザーがそのエンティティーを信頼する決定を下した場合にのみ行ってください。IMS TM リソース・アダプター (クライアント) トラストストア項目の例として、ターゲット SSL サーバーである IMS Connect の証明書があります。

鍵項目およびトラステッド証明書項目は、鍵ストアに保管することも、トラストストアに保管することもできます。それらを別々に保管することもできます。IMS TM リソース・アダプターは、分散プラットフォーム (Linux for System z を含む) では X.509 証明書および JKS 鍵ストア・タイプのみをサポートし、z/OS では JKS 鍵ストア・タイプまたは RACF 鍵リングをサポートします。

SSL プロトコル

SSL プロトコルは、サーバー認証とクライアント認証、その後に続く暗号化された会話 (SSL ハンドシェーク) で構成されます。

サーバー認証

SSL サーバー認証を使用して、クライアントはサーバーの身元を確認できます。SSL 対応のクライアント・ソフトウェアは、標準手法である公開鍵暗号方式を使用して、サーバーの証明書とパブリック ID が有効であること、およびその証明書と ID が信頼できる認証局 (CA) のクライアントのリストのいずれかから発行されたことを確認します。

クライアント認証

SSL クライアント認証を使用して、サーバーはクライアントの身元を確認できます。SSL 対応のサーバー・ソフトウェアは、サーバー認証の場合と同じ手法を使用して、クライアントの証明書とパブリック ID が有効であること、およびその証明

書と ID が信頼できる認証局 (CA) のサーバーのリストのいずれかによって発行されたことを確認します。

Null 暗号化

Null 暗号化を使用すると、SSL ハンドシェーク時に認証を実行できます。 SSL ハンドシェークが完了すると、すべてのメッセージはそのソケットを介して暗号化されずに流れます。

SSL ハンドシェーク

クライアントである IMS TM リソース・アダプターと、サーバーである IMS Connect の両方が、証明書と秘密鍵を鍵ストアに保管します。 IMS TM リソース・アダプターと IMS Connect の間の SSL セッションは、クライアントとサーバーの間のハンドシェーク・シーケンスに従って確立されます。このシーケンスは、サーバーがサーバー証明書のみを提供するように構成されているか、それともサーバー証明書を提供し、クライアント証明書を要求するように構成されているか、およびどの暗号スイートが使用可能であるかに応じて異なります。暗号は暗号化アルゴリズムです。SSL プロトコルは、クライアントとサーバーがどのように、使用する暗号スイートをネゴシエーションし、相互に認証し、証明書を伝送し、セッション鍵を設定し、メッセージを送信するのかを決定します。暗号スイートで使用されるアルゴリズムには、以下のようなものがあります。

- DES - データ暗号化規格
- DSA - デジタル署名アルゴリズム
- KEA - 鍵交換アルゴリズム
- MD5 - メッセージ要約アルゴリズム
- RC2 および RC4 - Rivest の暗号を解く鍵
- RSA - 暗号化と認証の両方に対する公開鍵アルゴリズム
- RSA 鍵交換 - RSA アルゴリズムに基づく SSL の鍵交換
- SHA-1 - セキュア・ハッシュ・アルゴリズム
- SKIPJACK - FORTEZZA 準拠ハードウェアに実装された、分類されている対称鍵アルゴリズム
- Triple-DES - DES を 3 重に適用したアルゴリズム。

SSL ハンドシェークおよび認証プロセス

SSL を使用するには、クライアント (IMS TM リソース・アダプター) とサーバー (IMS Connect) の両方を SSL ハンドシェーク用に構成する必要があります。

実行時、Java クライアント・アプリケーションが IMS との対話を実行する場合、その対話は、SSL クライアント (IMS TM リソース・アダプター) と SSL サーバー (IMS Connect) の間のセキュア (SSL) 接続上を流れます。クライアントとサーバーとの間の SSL 接続を開くために、SSL ハンドシェーク・プロセスが行われます。この SSL ハンドシェークは、Java クライアント・アプリケーションに対して透過的に行われ、以下の手順で進みます。

1. SSL クライアント (IMS TM リソース・アダプター) は、クライアントの hello メッセージを送信して接続を開始します。サーバー (IMS Connect) は、サーバーの hello メッセージと、サーバーの公開鍵を含む証明書によって応答します。
2. この証明書がサーバーによって正常に認証されると、双方の側でセッション鍵が設定され、接続で使用される暗号化のタイプを決定する暗号仕様がネゴシエーションされます。暗号は、STRONG、WEAK、ENULL のいずれかに設定できます。サーバーがクライアントの認証を要求しない場合、SSL ハンドシェークはこれで完了します。
3. サーバーがクライアント認証を要求する場合、クライアントは、サーバーの証明書にあるサーバーの公開鍵を使用して、その証明書を認証します。この認証が成功すると、クライアント証明書がクライアントの鍵ストアから送信されます。この証明書がサーバーによって正常に認証されると、双方の側でセッション鍵が設定され、接続で使用される暗号化のタイプを決定する暗号仕様がネゴシエーションされます。これで、SSL ハンドシェークが完了します。
4. クライアントとサーバーで、暗号化されたデータを送受信する準備ができました。

重要**:** クライアント・アプリケーションが管理対象環境 (SSL 接続で特に推奨される環境) で実行される場合、IMS TM リソース・アダプターは永続ソケット接続を使用して IMS Connect と通信する必要があります。ただし、非管理対象環境では、これらの永続接続は別のアプリケーションによって再使用できるようにならず、使用後に毎回アプリケーションによって切断されます。

WebSphere Application Server Connection Manager が使用される場合、接続は他のクライアント・アプリケーションによって順次に再使用できます。接続マネージャーは、必要に応じて接続を作成し、それを必要に応じてアプリケーションに提供します。アプリケーションが接続の使用を終了すると、接続マネージャーはその接続を空きプールに返し、そのタイプの接続を必要とする他のアプリケーションが再使用できるようにします。ただし、クライアントとサーバーの認証はソケットごとに 1 回のみ (そのソケットが SSL ソケットとして最初に作成および初期設定されるときに行われるハンドシェーク中に) 行われます。ソケットが再使用される場合、SSL クライアント (IMS TM リソース・アダプター) とサーバー (IMS Connect) は変更されません。そのため、ソケットの再使用時に、クライアントとサーバーの再認証 (ハンドシェーク・プロセスのやり直し) は必要ありません。ソケットが再使用されるたび、ソケットを識別するクライアント ID は同じままです。

SSLEncryption 値が ENULL に設定された場合、STRONG や WEAK の暗号化を使用する SSL 接続よりパフォーマンスは速くなります。パフォーマンスがどの程度向上するかは、ハードウェア暗号化とソフトウェア暗号化のどちらを使用するかといった、いくつかの要因によって異なります。一般に、ハードウェア暗号化はソフトウェア暗号化より高速です。

クライアントおよびサーバーの SSL サポート用の構成

SSL を使用するには、クライアント (IMS TM リソース・アダプター) とサーバー (IMS Connect) の両方を構成する必要があります。

IMS TM リソース・アダプターおよび IMS Connect を SSL サポート用に構成するには、以下のようにします。

1. SSL サーバー (IMS Connect) でクライアント認証が必要かどうかを決定します。クライアント認証が必要でない場合は、ステップ 3 にスキップしてください。

   推奨事項: IMS Connect への無許可アクセスから保護するためには、クライアント認証を使用してください。
2. クライアント認証が必要な場合、クライアントでは、サーバーのトラストストアまたは鍵リングに署名付き証明書が入っている必要があります。
   a. クライアント用の署名済み証明書と秘密鍵を取得します。
   b. クライアントで、鍵ストアを作成し、クライアントの秘密鍵および証明書を挿入します。
   c. サーバー IMS Connect) で、クライアントの公開鍵証明書を鍵リングに挿入します。詳しくは、*IMS バージョン 14 コミュニケーションおよびコネクション IMS バージョン 14 コミュニケーションおよびコネクション*を参照してください。
3. クライアントで、トラストストア (別のオプションの鍵ストア) を作成し、サーバーの公開鍵証明書を挿入します。あるいは、トラステッド証明書および非トラステッド証明書がクライアントの鍵ストアに保管されている場合は、公開鍵証明書を同じ鍵ストアに挿入します。
4. 使用する IMS Connect SSL ポートを決定します。適切な値を使用して、IMS Connect および SSL 構成メンバーをセットアップします。 これらの構成メンバーのセットアップについて詳しくは、「*IMS バージョン 14 コミュニケーションおよびコネクション*」を参照してください。
5. ステップ 4 からのポート番号を含め、適切な SSL パラメーターを使用して、接続ファクトリーをセットアップします。
6. アプリケーションを SSL 接続ファクトリーにバインドします。

ヒント: SSLEncryption 値が ENULL に設定された場合、STRONG や WEAK の暗号化を使用する SSL 接続よりパフォーマンスは速くなります。パフォーマンスがどの程度向上するかは、ハードウェア暗号化とソフトウェア暗号化のどちらを使用するかといった、いくつかの要因によって異なります。一般に、ハードウェア暗号化はソフトウェア暗号化より高速です。

関連タスク:

1026 ページの『IMS TM リソース・アダプター情報のロギングとトレース』

クライアントの鍵ストアまたはトラストストアの作成:

SSL クライアントで、クライアント証明書を保管するための鍵ストアを作成し、サーバー証明書を保管するためのクライアント用トラストストアを作成します。

鍵ストアを管理するのに、いくつかのツールが使用可能です。以下のステップを実行すると、クライアント鍵ストアに保管するクライアント証明書と、サーバー証明書を保管するクライアント・トラストストアが作成されます。

1. クライアント IMS TM リソース・アダプターのための証明書を作成し、その証明書に、認証局 (例えば VeriSign) による署名を付けます。また、OpenSSL などのソフトウェアを使用して独自の認証局を作成し、独自の (自己署名付き) 証明書に署名することもできます。

2. Ikeyman や Keytool などの鍵管理ツールを使用して鍵ストアを作成します。
3. クライアント証明書 (使用可能な場合) を鍵ストアにインポートします。
4. 別の鍵ストアを作成して、クライアント用にトラストストアを作成します。
5. サーバー (IMS Connect) 用の証明書をクライアント・トラストストアにインポートします。

SSL 接続の構成**:**

Java クライアント・アプリケーションと IMS Connect との間のセキュア SSL 接続は、Java クライアント・アプリケーションによって使用される接続ファクトリーで、必ずその SSL プロパティーに適切な値を指定することにより作成されます。

WebSphere 開発環境および Rational 開発環境では、以下の複数の方法で SSL プロパティーをセットアップすることができます。

- 統合開発環境 (IDE) 内のテスト環境で Java クライアント・アプリケーションを実行している場合は、テスト環境に合わせて IDE または管理コンソールでツールを使用します。統合 WebSphere Application Server にインストールされた Java クライアント・アプリケーション内の接続ファクトリー・リソース参照を、リソース参照内の接続ファクトリーの JNDI 名を設定することにより、その統合 WebSphere Application Server にもデプロイされた SSL 構成済み接続ファクトリーにマップまたはバインドします。
- WebSphere Application Server で Java アプリケーションを実行している場合は、SSL 接続を作成する接続ファクトリーを構成できます。

WebSphere Application Server で SSL 接続を作成する接続ファクトリーを設定するには、次のようにします。

1. WebSphere Application Server 管理コンソールで、「**Resources**」 > 「**Resource Adapters**」 > 「***yourIMSTMResourceAdapter***」 > 「**J2C Connection Factories**」 > 「***yourJ2CConnectionFactory***」 > 「**Custom Properties**」と移動します。
2. ターゲット接続ファクトリーの JNDIName を「**Applications**」 > 「**Enterprise Applications**」 > 「***yourApplication***」 > 「**Map resource references to resources**」に指定することにより、Java クライアント・アプリケーション内の接続ファクトリー・リソース参照を、SSL 接続ファクトリーにマップします。

関連資料:

1059 ページの『IMS 接続ファクトリーのプロパティー』

RACF パスワードの変更

IMS Connect が RACF 認証を実行するように構成されている場合、RACF パスワードを変更するには、Java アプリケーションから IMS Connect にパスワード変更要求を発行します。

メッセージに指定されているユーザー ID の RACF パスワードを Java アプリケーションから変更するには、以下のようにします。

パスワード変更要求キーワード HWSPWCH を含むメッセージを次の形式で送信します。

HWSPWCH *old-password/new-password/new-password*

HWSPWCH キーワードはメッセージのアプリケーション・データ・セクションの先頭に指定し、その後に 1 つ以上の空白スペース、現在のパスワード、スラッシュ、新規パスワード、もう 1 つのスラッシュ、その後にもう一度新規パスワードを続ける必要があります。

- キーワード HWSPWCH はすべて大文字にする必要があります。
- HWSPWCH と古いパスワードの間には、1 つ以上の空白スペースを入れる必要があります。
- 古いパスワードは、そのユーザー ID で有効なパスワードでなければなりません。
- すべてのパスワードは RACF 標準に従っている必要があります。

パスワードが正常に変更されると、「HWSC0031I PASSWORD CHANGE SUCCESSFUL」というメッセージが返されます。操作が失敗した場合、IMS TM リソース・アダプターは、エラーを示す理由コードとともに例外 ICO0001E をスローします。

関連概念:

975 ページの『コンテナー管理 EIS サインオン』

980 ページの『コンポーネント管理 EIS サインオン』

関連資料:

1024 ページの『コールアウト要求での問題の診断』

関連情報:

965 ページの『IMS へのコマンド送信』

IMS 保留キューからのメッセージのリトリーブの保護

ユーザー ID とパスワード情報を指定して、権限があるユーザーのみが IMS 保留キューから非同期出力メッセージまたはコールアウト要求メッセージをリトリーブできるようにすることができます。

IMS TM リソース・アダプターの非同期出力プログラミング・モデルとコールアウト・プログラミング・モデルの両方を使用して、保留キューから非同期出力メッセージまたはコールアウト要求メッセージをリトリーブできます。権限があるユーザーのみが保留キューから非同期出力メッセージまたはコールアウト要求をリトリーブできるようにする場合、SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT または SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT コマンド・メッセージに含まれる TPIPE 名とともに、オプションでユーザー名も指定できます。許可は、メッセージが保留キューからリトリーブされるときに IMS OTMA によって行使されます。

IMS Connect または IMS OTMA でセキュリティー検査が有効になっている場合、SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT または SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT 要求のために適切なユーザー ID を指定して IMS TM リソース・アダプターを構成し、IMS 非同期保留キューから非同期出力メッセージまたはコールアウト要求をリトリーブする必要があります。IMSConnectionSpec クラス内、あるいは IMSConnectionFactory またはアプリケーションのデプロイメント記述子によって使用される認証別名内に、ユーザー ID とパスワードの情報を指定します。

OTMA のセキュリティーについて詳しくは、「*IMS* バージョン *14* コミュニケーションおよびコネクション」のトピック『非同期保留キュー内のメッセージの保護』を参照してください。

関連概念:

921 ページの『非同期出力プログラミング・モデル』

927 ページの『コールアウト・プログラミング・モデル』

1025 ページの『出力メッセージを含む Java 例外』

分散ネットワーク・セキュリティー資格情報に対するサポートの使用可能化

IMS TM リソース・アダプターは、Java EE アプリケーションと IMS の間で、ネットワーク・セッション ID およびネットワーク・ユーザー ID を含む、元の分散ネットワーク・セキュリティー資格情報を渡すことができます。IMS TM リソース・アダプターによって渡されたネットワーク・セキュリティー資格情報は、IMS ログ・レコードに書き込まれます。

IMS TM リソース・アダプターが渡すことのできるネットワーク・セッション ID は、デフォルトでは、使用されている認証サーバーの IP アドレスとポートです。ネットワーク・セッション ID の最大長は 254 バイトです。

IMS TM リソース・アダプターが渡すことのできるネットワーク・ユーザー ID は、認証サーバーで定義されたいる識別名です。ネットワーク・ユーザー ID の最大長は 246 バイトです。

ネットワーク・セキュリティー資格情報をサポートするために IMS TM リソース・アダプターで提供されるのは、拡張可能な Java 認証承認サービス (JAAS) ログイン・モジュールです。WebSphere Application Server または WebSphere Liberty のいずれかのアプリケーション・サーバーに JAAS ログイン・モジュールをインストールして、このログイン・モジュールを Java EE アプリケーションにリンクする必要があります。以下のプロセスは、ログイン・モジュールがインストールされ、構成された後で、ネットワーク・セキュリティー資格情報が JAAS ログイン・モジュールおよび IMS TM リソース・アダプターによってどのように渡されるかを示しています。

1. JAAS ログイン・モジュールにより、ネットワーク・セキュリティー資格情報を入力し、資格情報を取り込むよう求めるプロンプトがユーザーに出されます。
2. ネットワーク・セキュリティー資格情報は、認証のために、WebSphere Application Server または WebSphere Liberty によって、LDAP サーバーなどの外部ユーザー・アカウント・レジストリーに渡されます。
3. セキュリティー資格情報がユーザー・アカウント・レジストリーによって正常に認証されると、資格情報は Java プリンシパル・オブジェクトに設定され、IMS TM リソース・アダプターに渡されます。
4. IMS TM リソース・アダプターは、ネットワーク・ユーザー ID およびネットワーク・セッション ID を抽出し、そのセキュリティー資格情報を、OTMA メッセージ接頭部のセキュリティー・データ・セクションに取り込みます。次に、OTMA メッセージが、ネットワーク。・セキュリティー資格情報と一緒に IMS Connect に送信されます。

5. IMS Connect が、OTMA メッセージをネットワーク・セキュリティー資格情報と一緒に IMS に渡すと、その資格情報は IMS ログ・レコードに書き込まれます。

IMS TM リソース・アダプターが同期コールアウト・メッセージおよび非同期コールアウト・メッセージで IMS からのネットワーク・セキュリティー資格情報をサポートできるようにすることもできます。

分散ネットワーク・セキュリティー資格情報に対応した WebSphere Application Server の構成

WebSphere Application Server 上で実行される Java EE アプリケーションとの間でやりとりされるネットワーク・セキュリティー資格情報を IMS が監査できるようにすることができます。

Java EE アプリケーションから渡される分散ネットワーク・セキュリティー資格情報のサポートを有効にする前に、以下の項目が有効であることを確認してください。

- ご使用のアプリケーション用のコンテナー管理セキュリティー。ご使用のアプリケーションにコンテナー管理セキュリティーがない場合は、そのアプリケーションの `web.xml` ファイルの変更が必要になる可能性があります。
- 許可ユーザーを含む、LDAP サーバーなどの外部ユーザー・アカウント・レジストリー。

以下の手順では、次のことを前提としています。

- 外部ユーザー・レジストリーとして LDAP サーバーを使用していること。ただし、WebSphere Application Server でサポートされている他のいずれのユーザー・レジストリーも使用することができます。
- 作成するログイン・モジュールの別名として IMS_Login を使用していること。
- ログイン・モジュールが、/STA OTMA コマンドを送信する単純な imsicoivp.ear アプリケーションにリンクされていること。

1. ユーザー・レジストリーをセットアップします。
   a. 左側のメニューで、「**Security**」、次に「**Global Security**」をクリックします。さらに、「**User account repository**」域で、「**Available realm definitions**」リストから「**Standalone LDAP Registry**」を選択します。「**Configure**」をクリックします。
   b. ご使用の LDAP サーバーの構成メニューで、1 次管理ユーザー名を指定します。このユーザー名は、LDAP サーバーで定義されている必要があります。 管理ユーザー名は、固有のユーザー ID、組織単位、および 1 つ以上のドメインで構成されます。次の管理ユーザー名の例では、固有のユーザー ID は `admin`、組織単位は `users`、ドメインは `security` および `com` です。

      ```
      uid=admin,ou=users,dc=security,dc=com
      ```

   c. ご使用の LDAP サーバーのホストおよびポート・アドレスを指定します。
   d. ご使用の LDAP サーバーの基本識別名を指定します。 基本識別名は、権限があるユーザーを保持するディレクトリーです。例えば、以下の基本識別名が使用される場合があります。

      ```
      ou=users,dc=security,dc=com
      ```

上記の基本識別名が使用されている場合は、次の例のユーザーが許可されます。

```
uid=admin,ou=users,dc=security,dc=com
cn=Bob,ou=users,dc=security,dc=com
```

e. 「**Test connection**」をクリックして、LDAP サーバーへの接続をテストします。次に、「**Apply**」をクリックします。

f. 「Global Security」ページで、「**Available realm definitions**」リストから「**Standalone LDAP registry**」を選択し、さらに、「**Set as current**」をクリックします。次に、「**Apply**」をクリックします。 ユーザー・アカウント・レジストリーがセットアップされます。ログイン・モジュールは、このユーザー・アカウント・レジストリーを使用して、入力された分散ネットワーク・セキュリティー資格情報を認証します。

2. WebSphere Application Server のセキュリティーを有効にし、カスタム JAAS ログイン・モジュールをインストールします。

a. ご使用のサーバーで管理セキュリティー (Administrative Security) およびアプリケーション・セキュリティー (Application Security) が有効になっていることを確認します。「WebSphere Application Server」ウィンドウの左側で、「**Security**」 > 「**Global Security**」をクリックし、「**Enable administrative security**」および「**Enable application security**」を選択します。

b. 右側のパネルで、「**Authentication**」をクリックして、「**Java Authentication and Authorization Service**」を展開します。「**Application logins**」をクリックしてから、「**New**」をクリックします。

c. ログイン・モジュールの別名を作成します。この手順の以下のステップでは、IMS_Login という別名を作成することを前提としています。次に、表「**JAAS login modules**」で、「**New**」をクリックします。

d. モジュール・クラス名 `com.ibm.ims.login.IMSLoginModule` を指定します。これは、IMS TM リソース・アダプターにバンドルされているログイン・モジュールです。

e. 「**Authentication strategy**」リストから、「**REQUIRED**」を選択します。

f. 「**Custom properties**」セクションで、新しいカスタム・プロパティーを追加します。名前として `propIdentity`、値として `Caller` を入力します。

g. Web アプリケーションがあり、その URI が無保護の場合は、WebSphere Application Server が無保護 URI で認証を求めるプロンプトを出すことを許可する必要があります。認証を求めるプロンプトを WebSphere Application Server が出せるようにするには、「Global Security」ページで、「**Web and SIP Security**」をクリックします。「**General Settings**」を選択します。

h. 「**Web authentication behavior**」域で、「**Authenticate when any URI is accessed**」を選択します。

3. ログイン・モジュールを、ご使用のアプリケーションにマップします。

a. 左側のメニューで、「**Applications**」をクリックします。次に、「**Application Types**」をクリックしてから、imsicoivp.ear アプリケーションをクリックします。

b. 「**Resource References**」をクリックし、ご使用のアプリケーションを選択して、「**Modify Resource Authentication Method**」をクリックします。

c. 「**Use custom login configuration**」を選択したら、「**Application login configuration**」リストから「**IMS_Login**」を選択します。「**Apply**」 > 「**OK**」をクリックします。

ログイン・モジュールをアプリケーションにマップした後、Web ブラウザーでアプリケーションを開始して開くと、ユーザー名とパスワードの入力を求めるプロンプトが出されます。ユーザー名として識別名を入力します。以下のログ・ファイル・スニペットの例は、IMS TM リソース・アダプターが分散ネットワーク・セキュリティー資格情報を正常に受信したことを示しています。

```
********************
TMRA has received the following credentials:
Security Realm: '0.0.0.0:10389'
Distinguished Name: 'uid=admin,ou=users,dc=security,dc=com'
Authenticated?: 'true'
********************
```

資格情報が有効であり、LDAP レジストリーに存在する場合、識別名とレルムは IMS に伝搬されます。

関連概念:

975 ページの『コンテナー管理 EIS サインオン』

分散ネットワーク・セキュリティー資格情報に対応した WebSphere Liberty の構成

WebSphere Liberty 上で実行される Java EE アプリケーションとの間でやりとりされるネットワーク・セキュリティー資格情報を IMS が監査できるようにすることができます。

Java EE アプリケーションから渡される分散ネットワーク・セキュリティー資格情報のサポートを有効にする前に、以下の項目が有効であることを確認してください。

- ご使用のアプリケーション用のコンテナー管理セキュリティー。
- 許可ユーザーを含む、LDAP サーバーなどの外部ユーザー・アカウント・レジストリー。

この手順では、外部ユーザー・レジストリーとして LDAP サーバーを使用していることを前提としています。ただし、WebSphere Liberty でサポートされている他のいずれのユーザー・レジストリーも使用することができます。

1. 分散ネットワーク・セキュリティー資格情報が Web アプリケーションから送信される場合は、必ず、アプリケーション URI を保護します。WebSphere Liberty では、無保護 URI を認証しないためです。Web アプリケーション URI を保護するには、Web アプリケーションの `web.xml` ファイルでセキュリティー役割および許可を定義してください。 以下の `web.xml` ファイルの例では、テストと呼ばれるセキュリティー役割が定義されており、<security-constraint> タグ内で、テストは、そのアプリケーションへのアクセスを許可された唯一の役割です。アプリケーションにアクセスするユーザーには、「テスト」役割が割り当てられている必要があります。

```
<security-constraint>
<display-name>SampleWebApp</display-name>
 <web-resource-collection>
  <web-resource-name>SampleWebApp</web-resource-name>
  <url-pattern>/*</url-pattern>
 </web-resource-collection>
 <auth-constraint>
  <role-name>Testing</role-name>
 </auth-constraint>
</security-constraint>

<security-role>
 <role-name>Testing</role-name>
</security-role>
```

2. server.xml ファイルにレジストリー・エレメントを作成することによって、ご使用のアプリケーションのユーザー・レジストリーをセットアップします。 レジストリー・タグで、以下の項目を指定します。
   - LDAP タイプ、ポート、およびホスト・アドレス
   - 基本識別名の baseDN 属性
   - カスタム・フィルターの <customFilters> エレメント

   この例では、ユーザー・レジストリーとして LDAP サーバーが使用されるため、<ldapRegistry> エレメントが使用されます。

```
<ldapRegistry ldapType="Custom" port="9999" host="0.0.0.0"
    baseDN="ou=users,dc=test,dc=test">
    <customFilters userFilter="(&amp;(uid=%v)(objectclass=inetOrgPerson))"
        groupFilter="(&amp;(cn=%v)(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames)))"
        groupMemberIdMap="ibm-allGroups:member;ibm-allGroups:uniqueMember;groupOfNames:member;groupOfUniqueNames:uniqueMember"></customFilters>
</ldapRegistry>
```

3. カスタム・ログイン・モジュールを server.xml ファイルに構成します。
   a. <fileset> エレメントを使用する <library> エレメントを作成します。このエレメントは、IMSLogin.jar ファイルの場所を示します。IMSLogin.jar ファイルは、IMS TM リソース・アダプター .rar ファイルに圧縮されています。ただし、WebSphere Liberty は.rar ファイルをスキャンすることができないため、.rar ファイルを、WebSphere Liberty でスキャンできるディレクトリーに解凍する必要があります。この例では、**id** は customLoginLib です。
   b. <jaasLoginModule> エレメントを作成して、以下の属性を設定します。
      - **id** 属性。この例では、**id** は IMS_Login です。
      - **className** 属性で、com.ibm.ims.login.IMSLoginModule クラスを指定します。
      - **controlFlag** 属性で、分散ネットワーク・セキュリティー資格情報が正常に認証されることを必要とするように REQUIRED を指定します。
      - <libraryRef> タグで、前のサブステップで構成した <library> エレメントの ID である customLoginLib を指定します。
   c. <jaasLoginContextEntry> エレメントを作成し、システム定義 JAAS 構成の ID および固有名を指定します。次の例では、IMS_Login_Entry は、名前と ID として使用されます。**loginModuleRef** 属性で、前のサブステップで作成した <jaasLoginModule> エレメントの ID である IMS_Login を指定します。

```
<jaasLoginContextEntry id="IMS_Login_Entry"
    loginModuleRef="IMS_Login" name="IMS_Login_Entry" />

<jaasLoginModule className="com.ibm.ims.login.IMSLoginModule"
    controlFlag="REQUIRED" id="IMS_Login" libraryRef="customLoginLib">
    <options propIdentity="RunAs" />
</jaasLoginModule>

<library id="customLoginLib">
    <fileset dir="${server.config.dir}/libraries" includes="IMSLogin.jar" />
</library>
```

サーバー構成は、ログイン・モジュールを使用するようにセットアップされました。

4. ご使用のアプリケーションを、カスタム JAAS ログイン・モジュールにリンクします。Web アプリケーション・バインド・ファイルで、<resRefBindings> エレメントの loginConfigurationName 属性を使用して、アプリケーションをログイン・モジュールに手動でリンクします。loginConfigurationName 属性の値については、前のステップで定義した <jaasLoginContextEntry> エレメントの ID を指定します。

```
<?xml version="1.0" encoding="UTF-8"?>
<com.ibm.ejs.models.base.bindings.webappbnd:WebAppBinding xmi:version="2.0" xmlns
  <webapp href="WEB-INF/web.xml#WebApp"/>

  <resRefBindings xmi:id="ResourceRefBinding_1264197557812" jndiName="ims_cf2"
        loginConfigurationName="IMS_Login_Entry">
    <bindingResourceRef href="WEB-INF/web.xml#ResourceRef_1264197557812"/>
  </resRefBindings>

</com.ibm.ejs.models.base.bindings.webappbnd:WebAppBinding>
```

関連概念:

975 ページの『コンテナー管理 EIS サインオン』

IMS TM リソース・アダプター・クライアント・アプリケーションでのコールアウト・メッセージのネットワーク・セキュリティー資格情報のサポートの使用可能化

IMS TM リソース・アダプター・クライアント・アプリケーションで IMS コールアウト・メッセージのネットワーク・セキュリティー資格情報をサポートできるようにする場合、外部 Web サーバーで、その資格情報を使用して、コールアウト要求の認証および許可を実行することができます。

同期コールアウト・メッセージ内のネットワーク・セキュリティー資格情報
: IMS TM リソース・アダプター・クライアント・アプリケーションが同期コールアウト・メッセージのネットワーク・セキュリティー資格情報をサポートできるようにするには、IMSActivationSpec オブジェクトの resumeTpipeNsc プロパティーを true に設定します。

非同期コールアウト・メッセージのネットワーク・セキュリティー資格情報
: IMS TM リソース・アダプター・クライアント・アプリケーションで非同期コールアウト・メッセージのネットワーク・セキュリティー資格情報をサポートできるようにするには、IMSInteractionSpec オブジェクトについて setResumeTpipeNSC(int resumeTpipeNSC) メソッドを呼び出し、setResumeTpipeNSC プロパティーの値を 1 に設定してください。setResumeTpipeNSC プロパティーの値が 1 に設定されている場合、IMS

TM リソース・アダプターは、IMS に送信される OTMA メッセージ接頭部に、ネットワーク・セキュリティー資格情報がコールアウト・メッセージに含まれることを示すフラグ・バイトを設定します。

関連資料:

メッセージ駆動型 Bean のための IMSActivationSpec プロパティーの構成

RESUME TPIPE ネットワーク・セキュリティー資格情報 (resumeTpipeNSC)

IMS TM リソース・アダプターのタイムアウト

タイムアウトのタイプによっては、対話の実行時に発生する障害が原因でアプリケーションがハングしないようにするために使用できるものがあります。

ネットワークの問題が、クライアントと IMS との間のデータ伝送に影響することがあります。そのような予期せぬ問題に対処する仕組みがないと、アプリケーションは停止するか、無限ループで実行する可能性があります。その例として、IMS TM リソース・アダプターと IMS Connect との間の TCP/IP パスでのルーター障害があります。

そのような障害から回復するために使用できるタイムアウトがいくつかあります。

- IMSInteractionSpec クラスの executionTimeout プロパティーは、IMS Connect と IMS との間の問題を処理するために使用されます。そのような問題の例として、XCF 通信リンク障害や、IMS でターゲット・トランザクションが実行しない、またはその出力が IMS Connect に返されないという障害があります。
- IMSInteractionSpec クラスの socketTimeout プロパティーは、IMS TM リソース・アダプターと IMS Connect との間の TCP/IP 通信に影響する問題に対処するために使用されます。

対話の実行中に発生した問題を処理するために、executionTimeout プロパティーおよび socketTimeout プロパティーに値を指定できます。実行タイムアウトのタイマーは、ソケット・タイムアウト・ウィンドウ内で実行されます。

推奨事項**:** socketTimeout プロパティーは、常時、executionTimeout プロパティーの値よりも大きい値に設定してください。

実行タイムアウト

実行タイムアウトの値は、IMS Connect がメッセージを IMS に送信して、IMS からそのメッセージに対する応答を受信するまでの最大許容時間です。

実行タイムアウト設定は、IMS Connect に対し、その現在の TIMEOUT 値をオーバーライドするよう指示するために IMS TM リソース・アダプター・クライアント・アプリケーションによって使用されます。

実行タイムアウトが発生する前に対話が完了しない場合、IMS Connect は IMS TM リソース・アダプターにエラー・メッセージを返します。それにより、IMS TM リソース・アダプターは例外をクライアント・アプリケーションに返し、IMS が IMS Connect に応答する時間がタイムアウト値を超えたことを示します。このエラー・メッセージは、IMS Connect によって使用されたタイムアウト値も示します。

ヒント: IMS TM リソース・アダプターと IMS Connect の間の接続は永続であるため、実行タイムアウトの発生後に接続が既知の状態であれば、ソケットは閉じられません。代わりに、ソケットは開いたまま、再使用できる状態になります。

トランザクション有効期限

OTMA トランザクション有効期限の機能拡張を利用して、実行タイムアウト値に達したときにトランザクションを破棄またはデキューすることを OTMA に示すよう、IMS Connect に明示的に指示できます。

デフォルトでは、transExpiration プロパティーは後方互換性を確保するために false に設定され、OTMA は、実行タイムアウトになった後も引き続きトランザクションを処理します。 OTMA トランザクション有効期限機能を利用して、必要なくなったトランザクションでの不要な処理コストや CPU サイクルを節約するには、transExpiration プロパティーを true に設定します。

会話型トランザクション

会話型トランザクションの場合、実行タイムアウト値は会話の各反復に適用されます。反復 は、IMS に送信される 1 つの入力メッセージと、IMS から受信される 1 つの出力メッセージで構成されます。実行タイムアウトが原因で会話の反復がタイムアウトになると、会話は終了し、その会話中に行われたデータベース更新はすべてバックアウトされます。

例外

無効な実行タイムアウト値を指定すると、IMS Connect 構成メンバーで指定された TIMEOUT 値が使用され、javax.resource.NotSupportedException がスローされます。

関連概念:

1002 ページの『その他のタイプのタイムアウト』

有効な実行タイムアウト値

実行タイムアウト値はミリ秒で表され、1 から 3600000 まで (両端を含む) の 10 進整数でなければなりません。

つまり、実行タイムアウト値は、ゼロより大きく、1 時間以下でなければなりません。時間制限なしで対話を実行する場合は、実行タイムアウト値を -1 にすることもできます。実行タイムアウト値に非数値文字を含めることはできません。

実行タイムアウト値を指定しない場合、または指定した値が無効である場合は、次のようになります。

- SYNC_SEND_RECEIVE 対話では、IMSConnect 構成メンバー内のタイムアウト値が使用され、対話は続行します。
- SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT 対話、および SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT 対話では、IMS Connect はタイムアウト値を 2 秒に設定し、対話は続行します。

実行タイムアウトが送信専用対話に指定されている場合、実行タイムアウトは、送信専用の対話には適用されないため、無視されます。

無効値を指定し、タイムアウトが発生した場合、IMS Connect 構成メンバーに指定されたタイムアウト値が使用され、例外 javax.resource.NotSupportedException がスローされます。

ヒント: ホスト・システム管理者は、IMS Connect 構成メンバー内のグローバル・タイムアウト値を決定します。この値を表示するには、z/OS コンソールで VIEWHWS コマンドを発行します。

有効な実行タイムアウト値が設定されている場合、この値は、IMS Connect が使用できる値に変換されます。次の表は、指定された値が、IMS Connect が使用する値にどのように変換されるかを示しています。

ユーザー指定値の範囲	変換規則
1 - 250	ユーザー指定の値が 10 で割り切れない場合は、10 の増分で次に大きい数に変換されます。
251 - 1000	ユーザー指定の値が 50 で割り切れない場合は、50 の増分で次に大きい数に変換されます。
1001 - 60000	ユーザー指定値は、1000 の増分でその値に最も近い数に変換されます。1000 の増分のちょうど中間にある値は、1000 の増分で次に大きい数に変換されます。
60001 - 3600000	ユーザー指定値は、60000 の増分でその値に最も近い数に変換されます。 60000 の増分のちょうど中間にある値は、60000 の増分で次に大きい数に変換されます。

例えば、値 1 を指定した場合、この値は 10 に変換されます(1 は 10 で割り切れず、10 は 1 より大きい次の増分であるため)。以下の例は、値の範囲ごとにどのように変換が行われるかを示しています。

ユーザー指定値 (ミリ秒)	変換後の値 (ミリ秒)
1	10
11	20
251	300
401	450
1499	1000
1500	2000
60000	60000
89999	60000
3600000	3600000
3750000	3600000

関連概念:

1002 ページの『その他のタイプのタイムアウト』

関連資料:

1066 ページの『実行タイムアウト (executionTimeout)』

実行タイムアウト値の設定

実行タイムアウト値は、ご使用の Rational または WebSphere の統合開発環境 (IDE)の J2C ウィザードで、あるいは Common Client Interface (CCI) API を使用して設定することができます。

実行タイムアウト・プロパティーは、IMSInteractionSpec クラスのプロパティーです。設定した実行タイムアウト値は、IMS Connect が使用する値に変換されます。この変換は、IMS Connect の要件を満たすために行われます。

ソケット・タイムアウトなどの他のタイムアウト、またはクライアント・アプリケーションと WebSphere Application Server for z/OS との間の対話は、ご使用の対話に影響を与える可能性があります。IMS 対話に設定した実行タイムアウト値よりも他のタイムアウト値が小さい場合、それらの他のタイムアウトにより、IMS が応答を返さなかった事実が隠されてしまう可能性があります。

IDE では、新しい J2C Java Bean の IMS バインディング・プロパティーを最初に定義するときに、実行タイムアウト値を設定できます。

新しい J2C Java Bean について IDE で既に定義済みの IMS バインディング・プロパティーを編集するには、以下のステップを実行します。

1. Java エディターを使用して、IMS 用の適切な Java バインディングを開きます。
2. IMSInteractionSpec クラスのドックレット・タグを見つけます。
3. executionTimeout プロパティーを追加するようドックレット・タグを変更し、executionTimeout プロパティーがまだリストされていない場合には、そのプロパティーの値を指定します。該当のプロパティーがリストされている場合は、値を変更します。
4. エディターを閉じ、「**Yes**」をクリックして変更を保管します。

IMSInteractionSpec executionTimeout プロパティー値を設定することによって、さまざまな対話に対して個別のタイムアウト値を指定することもできます。Java クライアント・アプリケーション・コードに実行タイムアウト値を指定すると、その値は、J2C Java Bean の IMS バインディング・プロパティーに設定されたすべての実行タイムアウト値をオーバーライドします。

2 番目の方法では、setExecutionTimeout メソッドを使用して、CCI アプリケーションで実行タイムアウト値を設定します。

1. 最初に、新しい IMSInteractionSpec インスタンスを生成するか、または特定の対話から IMSInteractionSpec インスタンスを取得します。
2. setExecutionTimeout メソッドを使用して、executionTimeout 値を設定します。以下に例を示します。

   ```
   interactionSpec.setExecutionTimeout(timeoutValue);
   ```

3. この interactionSpec を特定の対話に割り当てます。

ソケット・タイムアウト

実行タイムアウトの値は、IMS TM リソース・アダプターが IMS Connect からの応答を待機する最大時間です。この時間が経過すると、ソケットが切断され、クライアント・アプリケーションに例外が返されます。

ネットワークの問題または経路指定の障害が発生した場合、ユーザーが指定したソケット・タイムアウト値により、クライアント・アプリケーションまたは IMS TM リソース・アダプターが、IMS Connect からの応答を無期限に待機する事態を回避できます。socketTimeout プロパティーは IMS Connect と IMS TM リソース・アダプターが通信に使用する TCP/IP ソケットに基づいているため、このプロパティーはローカル・オプション接続には適用されません。

socketTimeout プロパティーを使用すると、ソケットを使用して特定の対話の個々のタイムアウト値を設定できます。この値 (ミリ秒単位) は、IMSInteractionSpec クラスの socketTimeout プロパティーで設定できます。対話に socketTimeout property プロパティーが指定されていない場合、またはこのプロパティーが 0 ミリ秒に設定されている場合、ソケット・タイムアウト値は存在せず、接続は無期限に待機します。

ソケット・タイムアウト値を決定する際は、既存の他のタイムアウト値を考慮する必要があります。

特定の対話に有効なソケット・タイムアウト値が指定されていて、ソケット・タイムアウトが発生すると、java.io.IOInterruptedException および Java EE J2C 例外 javax.resource.spi.CommException がスローされます。Java EE J2C 例外メッセージは、クライアントが費やした時間が、IMS Connect との通信用に socketTimeout 値によって割り振られた時間を超えたことを示します。

関連概念:

1002 ページの『その他のタイプのタイムアウト』

関連資料:

1073 ページの『ソケット・タイムアウト (socketTimeout)』

ソケット・タイムアウト値の設定

ソケット・タイムアウト値は、実行タイムアウト値よりも大きな数値に設定します。

executionTimeout プロパティーは、IMS Connect が IMS にメッセージを送信し、IMS から応答を受信するまでの最大許容時間を設定するためのものです。ソケット・タイムアウト値は実行タイムアウト値をカプセル化するため、ソケット・タイムアウト値は実行タイムアウト値より大きくなければなりません。ソケットの値が実行タイムアウト値より小さい値に設定されている場合、ソケットが不必要にタイムアウトになる可能性があります。

以下の表は、実行タイムアウト値に基づくソケット・タイムアウトの推奨値をリストしています。

表 120. 実行タイムアウト値に基づくソケット・タイムアウトの推奨値

実行タイムアウト値 (ミリ秒)	実行タイムアウトの動作	推奨されるソケット・タイムアウト値
0 (または値なし)	IMS Connect 構成メンバーからのデフォルト値が使用されます。	ソケット・タイムアウト値は、IMS Connect 構成ファイルに指定された実行タイムアウトのデフォルト値より大きくなければなりません。
1 - 36000000	待機応答は、指定されたミリ秒値の経過後にタイムアウトになります。	ソケット・タイムアウト値は、実行タイムアウト値よりも大きくなければなりません。
-1	待機応答は無期限です。	ソケット・タイムアウト値を 0 に設定すると、接続は無期限に待機します。

ソケット・タイムアウト値を設定する方法は 2 つあります。 Common Client Interface (CCI) を使用して、IMSInteractionSpec クラスで提供された getter メソッドおよび setter メソッドにアクセスする方法か、あるいはオプションの Java EE Connector (J2C) 機能がインストールされた状態で、WebSphere 開発環境または Rational 開発環境によって提供されるツールを使用する方法です。

Common Client Interface を使用したソケット・タイムアウト値の設定:

ソケット・タイムアウト値を設定するには、IMSInteractionSpec オブジェクトの setSocketTimeout メソッドを使用します。

setSocketTimeout メソッドを使用してソケット・タイムアウト値を設定するには、次のようにします。

1. 新しい IMSInteractionSpec オブジェクトをインスタンス化するか、または特定の対話から IMSInteractionSpec オブジェクトを取得します。
2. setSocketTimeout メソッドを使用して、IMSInteractionSpec のソケット・タイムアウト値を設定します。 以下に例を示します。

```
interactionSpec.setSocketTimeout(timeoutValue1);
interaction.execute(interactionSpec,input,output);

interactionSpec.setSocketTimeout(timeoutValue2);
interaction.execute(interactionSpec,input,output);
```

開発環境を使用したソケット・タイムアウト値の設定:

新しい J2C JavaBean の IMSバインディング・プロパティーを最初に定義する際に、Rational開発環境または WebSphere開発環境を使用してソケット・タイムアウト値を設定できます。

新しい J2C Java Bean に対して既に定義済みの操作バインディング・プロパティーを編集するには、以下のステップを実行します。

1. Java Editor を使用して、適切な IMS バインディング Java ファイルを開きます。
2. IMSInteractionSpec クラスのドックレット・タグを見つけます。

3. socketTimeout プロパティーを追加するようにドックレット・タグを変更し、値を指定します。このプロパティーが既にリストされている場合は、値を変更します。
4. 操作拡張性エレメントを再度選択して、変更が行われたことを示します。
5. エディターを閉じ、「**Yes**」をクリックして変更を保管します。

その他のタイプのタイムアウト

実行タイムアウトとソケット・タイムアウトのほかに、その他のタイプのタイムアウト (J2C 接続ファクトリー、Enterprise JavaBeans (EJB) トランザクション、ブラウザー、HTTP セッション、および EJB セッションのタイムアウトなど) がアプリケーションの実行に影響を与える可能性があります。

アプリケーションを開発する際は、他のタイプのタイムアウトがアプリケーションの実行にどのように影響を与えるかについて検討する必要があります。

タイムアウトは、オーバーラップしたり、場合によっては互いが互いをカプセル化したりすることもあります。このような機能により、アプリケーションのハングの原因となり得る問題に対して、何層もの保護が施されます。より長いタイムアウトが、より短いタイムアウトをマスクすることがあります。例えば、ソケット・タイムアウトが実行タイムアウトより低い値に設定されると、ソケット・タイムアウトのエラーによって実行タイムアウトが回避されてしまい、IMS からの応答がないことが知られないままになります。

IMS TM リソース・アダプターのタイムアウトと相互作用する可能性のあるその他のタイムアウト値には、以下の値があります。

- J2C 接続ファクトリーの接続タイムアウト・プロパティー
- Enterprise JavaBeans (EJB) トランザクション・タイムアウト
- ブラウザー・タイムアウト
- サーブレット HTTP セッションまたは EJB セッションのタイムアウト

アプリケーションの実行に影響を与える可能性のあるタイムアウトの一例として、WebSphere Application Server for z/OS で実行されているアプリケーションと IMS TM リソース・アダプターの間の相互作用があります。WebSphere Application Server for z/OS は 2 つの部分、すなわち、1 つのコントローラーと、1 つ以上のサーバントのセットで構成されます。デフォルトでは、アプリケーションの処理は、アプリケーションがサーバント領域にディスパッチされる場合でも時間が決められています。サーバント領域にディスパッチされたアプリケーションがそのタイムアウトに達すると、そのサーバント領域は異常終了し、再始動されます。サーバーは稼働状態のまま、処理の実行を続けます。このため、WebSphere Application Server のタイムアウト値より大きい実行タイムアウト値を選択するときは注意してください。また、実行タイムアウト値 -1 を選択する場合も注意してください。この値は、IMS Connect に IMS からの応答を無期限に待機するように指示します。

WebSphere Application Server のタイムアウトを無効にする場合は、ここで説明されていない追加のタイムアウト値に関する情報について、サーバーの資料を参照してください。

アプリケーションの実行に影響を与えるタイムアウト値の 2 つ目の例は、ブラウザー・タイムアウトです。実行タイムアウト値をブラウザー・タイムアウトより大きく構成すると、ブラウザー・タイムアウトが実行タイムアウトより先に発生するため、実行タイムアウト値は使用されません。

関連情報:

WebSphere Application Server バージョン 8 の Knowledge Center

会話型プログラム

IMS プログラムは、1 つの対話または複数の対話で構成されるトランザクションをサポートできます。複数の反復で構成されるトランザクションは、会話型トランザクション と呼ばれます。

IMS 会話型プログラムは、完了するために複数の対話を要する複雑なトランザクションを処理することを目的としています。各対話を、会話の反復と呼びます。 IMS 会話型プログラムは、複雑な処理を、クライアントからプログラムへ、プログラムからクライアントへという、一連の関連した対話 (反復 と呼びます) に分割します。会話の各反復では、IMS 会話型トランザクション・プログラムはクライアントから要求メッセージを受け取り、その要求を処理し、応答をクライアントに送信します。このプログラムは、会話の次の反復で使用する、スクラッチパッド域 (SPA) 内のトランザクションからの中間データも保管します。ユーザーは応答内のデータを変更できます。また、要求内に追加データを入力してから IMS に送信し、会話の次の反復を処理することもできます。この新しい要求のデータを前回の反復からの SPA 内のデータとともに使用して、IMS 会話型プログラムは会話の次の反復を処理します。これらの反復の処理は、クライアント・アプリケーションか IMS のいずれかが会話を終了するまで続きます。

IMS TM リソース・アダプターの会話型サポートでは、IMS 会話型トランザクションと対話するクライアントは通常、以下のいずれかのアプリケーションを使用します。

- Java アプリケーションまたは Web アプリケーション
- サービス指向アーキテクチャー (SOA) 内のサービス・コンポーネント・アーキテクチャー (SCA) に基づくビジネス・プロセス・アプリケーション

Java アプリケーションまたは Web アプリケーション

Java アプリケーションまたは Web アプリケーションを使用する場合、ユーザーは同じブラウザー・セッションを使用して、1 つの IMS 会話の複数の異なる対話によって反復する一連の要求を実施します。1 つの考えられる実装としては、Java サーブレットがブラウザーから入力要求を受け取り、IMS TM リソース・アダプター・クラスを使用して、TCP/IP 通信を介して会話型トランザクション要求を IMS Connect に送信するというものです。 IMS Connect はその後、トランザクション要求を OTMA と IMS の両方に転送し、IMS 会話型トランザクションの実行をスケジュールします。このプロセスにより、新しい IMS 会話が作成されます。IMS アプリケーションは要求を処理し、IMS Connect および IMS TM リソース・アダプターの会話型サポートを介して、Java サーブレットに出力を返します。 Java サーブレットは、該当する JavaServer Pages (JSP) ファイルをロードし、ブラウザー内でユーザーに出力を表示します。

ビジネス・プロセス・アプリケーション

ビジネス・プロセスは、特定のビジネス目標を達成するために一群のサービスを組織化 (オーケストレーション) したものです。ビジネス・プロセス・コレオグラフィーは、ビジネス・アプリケーションを柔軟で適応性のある要素で構成するための SOA の主要な実装です。 IBM WebSphere Process Server はビジネス・プロセスを実行するためのランタイム・インフラストラクチャーを提供し、IBM WebSphere Integration Developer はビジネス・プロセスや他のさまざまなコンポーネントを作成するためのモデリング・ツールを提供します。

クライアント管理と IMS Connect 管理の会話状態プログラミング・モデル

IMS プログラムは、1 つの対話または複数の対話で構成されるトランザクションをサポートできます。複数の反復で構成されるトランザクションは、会話型トランザクションと呼ばれます。

アプリケーションでビジネス・プロセス・コレオグラフィーを可能にするには、Java アプリケーション内で、会話状態が IMS Connect ではなくアプリケーションによって管理されることを示します。会話の複数の反復の間で固有の会話 ID を管理することにより、任意の接続から複数の異なる反復が到着できるようになります。このモデルを、クライアント管理の会話状態プログラミング・モデル と呼びます。

デフォルトの場合、会話状態は、後方互換性を確保するために引き続き IMS Connect によって管理されるようになっています。このモデルを、*IMS Connect* 管理の会話状態プログラミング・モデル と呼びます。

アプリケーションでビジネス・プロセス・コレオグラフィーを可能にする場合、またはアプリケーションを共用可能永続ソケット接続で使用できるようにする場合は、クライアント・アプリケーションで会話状態を管理する必要があります。アプリケーションで、IMSInteractionSpec クラスの useConvID プロパティーを true に設定することにより、会話の複数の反復の間で受け渡しする会話 ID をユーザーに代わって IMS によって割り当てる必要があることを、IMS Connect に対して示します。このクライアント管理の会話状態プログラミング・モデルでは、1 つの会話の複数の異なる反復が、拒否されることなく、任意の接続から到着できます。

推奨事項: このクライアント管理の会話状態プログラミング・モデルは、会話状態の管理を向上するために、すべての新規アプリケーション開発に使用してください。

関連タスク:

1006 ページの『クライアント管理の会話状態プログラミング・モデルの使用』

1007 ページの『IMS Connect 管理の会話状態プログラミング・モデルの使用』

孤立した IMS 会話

会話は、明示的に終了されなかった場合、孤立した会話としてシステムに存在し続け、関連する IMS ストレージが引き続きその会話に割り振られます。

IMS 会話は、通常、以下の 2 つの方法のいずれかで明示的に終了されます。

- IMS アプリケーションが、応答をクライアントに返す前に SPA に空白を挿入する。
- クライアント・アプリケーション・プログラムが SYNC_END_CONVERSATION 要求をサブミットする。

例えば、会話が正しく終了する前にブラウザーが閉じられた場合、IMS 会話は明示的に終了されず、システム内に存在し続けます。 IMS 会話が孤立するようになると、その会話をプログラムで続行や終了する方法はありません。孤立した会話の発生を回避するために実行できる 1 つの対策として、タイムアウト (例えば、EJB セッション・タイムアウト) を使用して、適切な時間内に完了しない会話を強制的に終了するという方法があります。この場合、EJB セッション・タイムアウト・クリーンアップ・コードで SYNC_END_CONVERSATION 要求をサブミットします。

クライアント・アプリケーションが終了して、会話が孤立するようになった場合、孤立した IMS 会話を終了できる唯一の方法は IMS の再始動です。システム内に孤立した IMS 会話がないか確認するには、IMS_REQUEST_TYPE_IMS_COMMAND 対話を通じて IMS /DISPLAY CONV コマンドを発行します。 OTMA によってサポートされている IMS コマンドのリストについては、「*IMS* バージョン *14* コマンド 第 *2* 巻」の『LU 6.2 装置および OTMA からサポートされるコマンド』または「*IMS* バージョン *14* コミュニケーションおよびコネクション」の『OTMA を使用する IMS コマンド (IMS Commands using OTMA)』を参照してください。

ビジネス・プロセス・コレオグラフィー・アプリケーション

IMS TM Resource Adapter を使用することで、IMS 会話型トランザクションを複合ビジネス・アプリケーションで実行できます。

サービス指向アーキテクチャー (SOA) でビジネスの柔軟性を確保するための主要な実装環境が、ビジネス・プロセス・コレオグラフィー です。ビジネス・プロセス・コレオグラフィーでは、ビジネス・アプリケーションが柔軟で適応性のある要素で構成されるので、ビジネスの要求に合わせてアプリケーションを迅速に変更できます。新しいビジネス・プロセスやニーズに合わせて簡単に変更できない 1 つの巨大なアプリケーションを持つ代わりに、ビジネス・プロセス・モデルに基づくビジネス・アプリケーションを作成します。 IMS TM Resource Adapterを使用すると、IBM WebSphere Process Server によって提供される複合ビジネス・アプリケーションで IMS 会話型トランザクションを再使用できます。

ビジネス・プロセス・コレオグラフィーを通じて IMS 会話型アプリケーションを呼び出すには、以下のようにします。

- クライアント管理の会話状態プログラミング・モデルを使用する必要があります。

  ビジネス・プロセス・コレオグラフィー・アプリケーション用のサービス・コンポーネント・アーキテクチャー (SCA) で、IMSInteractionSpec クラスの useConvID プロパティーを `true` に設定することにより、IMS Connect に対し、会話を追跡するためにこの会話には固有の会話 ID が必要であることを示す必要があります。

この設定がトリガーとなって、IMS は固有の会話型トークンを割り当て、それを最初の反復の出力メッセージに入れて IMS Connect に返します。これで、この会話型トークンをビジネス・プロセス・コレオグラフィー・アプリケーションと IMS Connect の間でやり取りできるようなり、また、OTMA に渡すことができます。

- SCA エンタープライズ情報システム (EIS) バインディングを使用する場合に WebSphere Process Server によって提供される、動的相互作用仕様 (InteractionSpec) サポートを使用する必要があります。 SCA コンポーネントはこのサポートを使用して、会話の後続のすべての反復内で、または会話を終了する場合は SYNC_END_CONVERSATION 反復内で、会話トークンを次のコンポーネントに伝搬します。
- IMSInteractionSpec オブジェクトでコミット・モード 1 (CM1) と同期レベル NONE または CONFIRM を指定します。
- 非シスプレックス環境では、特定の IMS 会話のすべての反復は、同じ IMS Connect および同じ IMS によって処理される必要があります。この固有の会話 ID は、複数の IMS システムの間で共用されません。
- 特定の IMS 会話のすべての反復は、同じ会話型 ID、ポート番号、IMS Connect、およびデータ・ストアを使用する必要があります。

IMS Connect および OTMA によって提供される会話型サポートおよび関連の制約事項について詳しくは、「*IMS* バージョン *14* コミュニケーションおよびコネクション」のトピック『IMS Connect 会話型サポート』を参照してください。

WebSphere Process Server で提供される動的相互作用仕様サポートについて詳しくは、WebSphere Process Server の資料にある JCA EIS バインディング関連のトピックを参照してください。

関連情報:

WebSphere Process Server の資料 (すべてのバージョン)

IMS Connect 会話型サポート (IMS バージョン 15)

IMS 会話型トランザクションを Java クライアントで使用可能にする

クライアント管理の会話状態プログラミング・モデルまたは IMS Connect 管理の会話状態プログラミング・モデルを使用して、IMS 会話型トランザクションを Java クライアントで使用可能にすることができます。

関連概念:

1003 ページの『会話型プログラム』

クライアント管理の会話状態プログラミング・モデルの使用

クライアント・アプリケーションで会話の反復をすべて管理するには、IMSInteractionSpec クラスの useConvID プロパティーを true に設定する必要があります。

クライアント管理の会話状態プログラミング・モデルを使用して、IMS 会話型トランザクションを Java クライアントで使用可能にするには以下のようにします。

1. IMSConnectionFactory オブジェクトから、接続ハンドルを取得します。このプログラミング・モデルでは、後続の反復で同じ接続ハンドルまたは異なる接続ハンドルを使用できます。これらの接続ハンドルは、IMSConnectionFactory インスタンスからのものでなければなりません。
2. IMSInteractionSpec の useConvID プロパティーを `true` に設定します。このプロパティーは、会話のすべての反復で TRUE になっている必要があります。会話中に useConvID フラグが変更されると、エラーが発生します。
3. IMSInteractionSpec commitMode プロパティーを 1 に設定します。
4. アプリケーションの設計に基づいて、syncLevel プロパティーに適切な値を設定します。
5. 最初の反復で、IMSInteractionSpec convID プロパティーの値を空ストリングに設定します。
6. 最初の反復以外のすべての反復で、convID プロパティーの値を、直前の反復で返される convID プロパティーの値に設定します。
7. クライアント・アプリケーションが会話の終了を制御する場合は、最後の反復で、IMSInteractionSpec interactionVerb プロパティーを SYNC_END_CONVERSATION に設定します。

   推奨事項: クライアント管理の会話状態プログラミング・モデルでは、アプリケーション全体の参照ではなく、クライアント・アプリケーションまたはサーブレットでの要求ごとにローカル参照を使用してください。この方法は、会話型データの保全性に違反する可能性のあるストレスまたは負荷が原因で、クライアント・アプリケーションで発生する可能性のある競合状態を回避することです。クライアント・アプリケーションがブラウザー・ベースの場合は、会話型要求が誤って再サブミットされないように、ナビゲーションを慎重に設計する必要があります。このシナリオは、ユーザーがブラウザーで「戻る」ボタンを押した場合に発生する可能性があります。このボタンを押すと、会話型要求は再度再サブミットされるため、さらに会話が追加されます。

関連概念:

1004 ページの『クライアント管理と IMS Connect 管理の会話状態プログラミング・モデル』

IMS Connect 管理の会話状態プログラミング・モデルの使用

IMS Connect が会話内のすべての反復を管理するよう指定するには、IMSInteractionSpec クラスの useConvID プロパティーを false に設定する必要があります。

IMS Connect 管理の会話状態プログラミング・モデルを使用して、IMS 会話型トランザクションを Java クライアントで使用可能にするには以下のようにします。

1. IMSConnectionFactory から接続ハンドルを取得します。この接続ハンドルは、会話の後続のすべての反復で使用する必要があります。Java アプリケーションが Web アプリケーションの場合、接続ハンドルがリトリーブされ、同じブラウザーの後続の反復で使用されるようにするために、接続ハンドルを HTTP セッション・オブジェクトに保管しなければならない可能性があります。

2. 最初の反復で IMSInteractionSpec useConvID プロパティーを `false` に設定します (デフォルト値は false)。このプロパティーの値を、会話の後続の反復で変更しないでください。会話中に useConvID プロパティーが変更されると、エラーが発生します。
3. IMSInteractionSpec commitMode プロパティーを 1 に設定します。syncLevel プロパティーに適切な値を設定します。
4. クライアント・アプリケーションが会話の終了を制御する場合は、最後の反復で、IMSInteractionSpec interactionVerb プロパティーを SYNC_END_CONVERSATION に設定します。
5. 会話の終わりで、接続ハンドルを閉じます。

関連概念:

1004 ページの『クライアント管理と IMS Connect 管理の会話状態プログラミング・モデル』

グローバル・トランザクションの処理

IMS TM リソース・アダプターはグローバル・トランザクション管理と 2 フェーズ・コミット処理をサポートしているので、アプリケーションは Java EE に準拠したアプリケーション・サーバーで稼働し、IMS トランザクションにアクセスできます。

2 フェーズ・コミットを使用するグローバル・トランザクション・サポート

ビジネス・リソースの保全性を保護および維持するために、IMS TM リソース・アダプターは、グローバル・トランザクション管理および 2 フェーズ・コミット処理をサポートしています。

このサポートを使用すると、複数の変更を 1 つのトランザクションまたは単一の作業単位にまとめる Java EE アプリケーションを作成して、トランザクション内のすべての変更をすべて一括して完了したりロールバックしたりできます。このサポートにより、アプリケーションを Java EE 準拠アプリケーション・サーバー (例えば、WebSphere Application Server) 内で実行して、IMS トランザクションおよびデータに対するアクセスが調整して行われるようになります。グローバル・トランザクション管理を行うと、IMS 内のデータの整合性を確保できます。

グローバル・トランザクション・サポートの例

保護リソースに変更を行う場合、その変更は確実に間違いなく行われなければなりません。例えば、銀行の顧客が普通預金口座から当座預金口座に預金を移す場合があります。預金を普通預金口座から引き出した場合、その金額は同時に当座預金口座に追加されなければなりません。このトランザクションが一部しか完了しない (つまり、預金が普通預金口座から引き出されても、当座預金口座には追加されない) 事態が起こってはなりません。

別の例として、サンフランシスコからパリまでの航空券を購入する必要がありますが、直行便がないとします。サンフランシスコからシカゴ行きの航空券とシカゴからパリ行きの別の航空券を首尾良く予約できない限り、パリへの渡航に対するコミ

ットは行いません。つまり、パリに行くという決定を見直す (ロールバックする) ことになります。旅程の半分の座席だけが確定しても意味がないからです。

上記の例ではいずれも、1 つのトランザクション全体を完了するために複数の小さなトランザクションが必要です。これらの小さなトランザクションのいずれかで問題が発生した場合、トランザクション全体 (例えば、預金の転送やパリへの渡航) をコミットすることはありません。代わりに、トランザクションのすべてのステップをロールバックして、小さなトランザクションがどれもコミットされないようにするでしょう。預金の転送やパリへの渡航を問題なく行うには、小さなトランザクションを一括して管理および調整してトランザクション全体を完了する必要があります。

整合トランザクション処理を確実に行うために、Java EE プラットフォーム (Java EE アプリケーション・サーバー、Java EE アプリケーション・コンポーネント、および Java コネクター・アーキテクチャー・リソース・アダプターで構成されます) では分散トランザクション処理環境が提供されます。この環境では、トランザクションは透過的に管理され、リソースの更新とリカバリーが複数のプラットフォーム全体で調整して行われます。

関連概念:

884 ページの『WebSphere Application Server プラットフォーム構成および通信プロトコルに関する考慮事項』

関連資料:

1012 ページの『2 フェーズ・コミット環境に関する推奨』

グローバル・トランザクションと 2 フェーズ・コミット・サポート処理

Java EE 準拠のアプリケーション・サーバーは、Java トランザクション・マネージャーを使用して、アプリケーション・コンポーネントとリソース・マネージャー間の通信および調整を行います。

例えば、Java EE に準拠したアプリケーション・サーバー (例えば、WebSphere Application Server) は、リソース・アダプター (例えば、IMS TM リソース・アダプター) を介して、アプリケーション・コンポーネント (例えば、Java サーブレットや Enterprise JavaBeans コンポーネント) およびリソース・マネージャー (例えば、IMS や DB2) と通信してトランザクションを調整します。トランザクション・マネージャーがトランザクションを調整すると、そのトランザクションはグローバル・トランザクションとみなされます。トランザクション・マネージャーが複数のリソース・マネージャーを使用してトランザクションを調整すると、外部コーディネーターは 2 フェーズ・コミット・プロトコルを使用します。

普通預金口座から当座預金口座に預金を移そうとしているとします。普通預金口座の情報が当座預金口座の情報と別のリソース・マネージャーにある場合 (例えば、普通預金口座の情報が IMS にあり、当座預金口座の情報が DB2 にある場合)、アプリケーション・サーバー (WebSphere Application Server) 内のトランザクション・マネージャーが、2 フェーズ・コミット処理を使用して IMS と DB2 の間で変更を透過的に調整できるように、アプリケーションを支援します。具体的には、トランザクション・マネージャーは IMS TM リソース・アダプターと協働して IMS 内の変更を調整します。

IMS TM リソース・アダプターは、Java EE プラットフォーム内の Java トランザクション・マネージャー、z/OS のリソース・リカバリー・サービス (RRS)、および IMS Connect と協働して、IMS や他の保護リソースに対して一貫した変更を行うように設計されています。

IMS での 2 フェーズ・コミット処理に参加するために、IMS TM リソース・アダプターは IMS OTMA 同期レベル同期点プロトコルを使用します。リモート・アプリケーションから変更が要求されたときにグローバル・トランザクションと 2 フェーズ・コミット処理に参加するために、IMS は z/OS 上の RRS を使用します。

RRS は、外部コーディネーターまたは同期点マネージャーとして機能して、リソースの更新とリカバリーを調整します。 IMS TM リソース・アダプターと IMS Connect はアプリケーション・サーバーで実行されている Java トランザクション・マネージャー、および z/OS 上の RRS と対話して、Java EE プラットフォームで実行されているグローバル・トランザクションが、ホストで実行されている IMS での整合更新に参加できるようにします。

関連概念:

884 ページの『WebSphere Application Server プラットフォーム構成および通信プロトコルに関する考慮事項』

関連資料:

1012 ページの『2 フェーズ・コミット環境に関する推奨』

グローバル・トランザクション・サポートの要件

グローバル・トランザクションに参加できるように Java EE アプリケーションをセットアップする場合、適切な通信プロトコルを選択して、RSS を IMS で使用できるようにする必要があります。

- IMS でリソース・リカバリー・サービス (RRS) 処理を実行できる必要があります。IMS で RRS 処理を実行できることを確認するには、ご使用の IMS 環境で始動パラメーターの RRS 値が Y に設定されていることを確認してください。この設定値は、IMS を起動したときに生成されるジョブ・ログに表示されます。
- TCP/IP で 2 フェーズ・コミット処理を使用する場合は、IMS Connect コマンド SETRRS ON を発行するか IMS Connect 構成ファイルで RRS=Y を設定して、IMS Connect で RRS 処理を実行できるようにする必要があります。

TCP/IP を使用したグローバル・トランザクション

グローバル・トランザクション・スコープでは、Java EE アプリケーション・コンポーネントは IMS Connect との TCP/IP 接続を確立することによって IMSトランザクションにアクセスできます。

TCP/IP を使用したグローバル・トランザクション

IMS TM リソース・アダプターは、X/Open (XA) プロトコルを使用して Java トランザクション・マネージャーと対話し、グローバル・トランザクションと 2 フェーズ・コミット処理を管理します。 XA プロトコルは、Java トランザクション・マネージャーとリソース・マネージャーが分散トランザクション処理環境でどのように対話するかを記述する 1 組のインターフェースと対話を定義します。 IMS

TM リソース・アダプターは IMS Connect とともに、XA プロトコルを使用し、IMS と z/OS 上のリソース・リカバリー・サービス (RRS) と協働して一貫した変更を行います。

TCP/IP によるグローバル・トランザクションを使用するには、IMS Connect および IMS と同じ z/OS イメージで RRS を実行する必要があります。

IMS と IMS Connect が IMS 14 以降である場合、IMS Connect がグローバル・トランザクションのカスケードをサポートするように構成されていれば、IMS と IMS Connect が同じ z/OS イメージ上になくてもかまいません。ただし、各 z/OS イメージでそれぞれに RRS のインスタンスが実行されている必要があります。

クライアント・アプリケーションでのグローバル・トランザクション・サポート

Java EE プラットフォームでは、プログラマチック・アプローチまたは宣言型トランザクション区分アプローチのいずれかを使用してご使用のアプリケーションのトランザクションを管理できます。

プログラマチック・アプローチはコンポーネント管理 (または Bean 管理) トランザクションであり、宣言型トランザクション区分アプローチはコンテナー管理トランザクションです。

コンポーネント管理 (または Bean 管理) トランザクション

Java EE アプリケーションは、Java トランザクション API (JTA) javax.transaction.UserTransaction インターフェースを使用して、保護リソースへの一連の変更に対してプログラマチックにトランザクション境界を定めます。コンポーネント管理トランザクションは、サーブレットと EJB 環境の両方で使用できます。 EJB コンポーネントの場合は、そのデプロイメント記述子内のトランザクション属性を TX_BEAN_MANAGED として設定します。

通常、トランザクションは UserTransaction.begin() 呼び出しで開始されます。アプリケーション・コンポーネントは、変更をコミットする準備ができると UserTransaction.commit() 呼び出しを起動して変更の調整とコミットを行います。アプリケーション・コンポーネントはトランザクションをロールバックする必要がある場合、UserTransaction.rollback() を呼び出します。これで、すべての変更がバックアウトされます。以下に例を示します。

```
// Get User Transaction
javax.transaction.UserTransaction transaction =
ejbcontext.getUserTransaction();

// Start transaction
transaction.begin();

// Make changes to the protected resources.
// For example, use the Java EE or JCA CCI Interaction interface
// to submit changes to an EIS system(s)
interaction.execute(interactionSpec, input, output);

if (/* decide to commit */) {
// commit the transaction
transaction.commit();
```

```
} else { /* decide to roll back */
// rollback the transaction
transaction.rollback();
}
```

コンテナー管理トランザクション

コンテナー管理トランザクションは、EJB 環境でのみ使用できます。 EJB コンポーネントは、デプロイメント記述子のトランザクション属性 (例えば、TX_REQUIRED) を使用してコンテナー管理トランザクションを宣言によって指定します。コンテナー管理トランザクションは EJB コンテナーによって管理されます。コンテナーは EJB コンポーネントのために適切なメソッド (例えば、begin、commit、rollback) を呼び出します。この宣言型アプローチを使用すると、EJB コンポーネントでのプログラミング呼び出しが簡素化されます。

関連情報:

Java EE アーキテクチャーおよび JTA 仕様

2 フェーズ・コミット環境に関する推奨

2 フェーズ・コミット・アプリケーションを実行する場合は、以下の推奨に従って、リソースの競合のために領域またはメッセージの処理が停止しないようにしてください。

- 1 つの領域を対象に複数の 2 フェーズ・コミット・アプリケーションが競合しないように、可能な限り多くのメッセージ処理プログラム (MPP) 領域を実行してください。これは、2 フェーズ・コミット・アプリケーション内のトランザクションが、2 フェーズ・コミット・トランザクションの期間全体にわたって MPP 領域を使用するからです。
- 2 フェーズ・コミット・トランザクション内で複数の IMS トランザクションが実行されている場合は、2 フェーズ・コミット・アプリケーションが停止しないように、少なくともそれと同じ数の MPP 領域が使用可能になっている必要があります。
- 過大な時間にわたってリソースを待機する可能性があるトランザクションから保護するために、グローバル・トランザクション内で行われる対話ごとに、適切なタイムアウト値を設定します。
- 1 つの 2 フェーズ・コミット・トランザクション内で実行されるデータベース対話の数が過大にならないようにします。複数の IMS トランザクションが 2 フェーズ・コミット・トランザクション内で使用されている場合は、これらが同じデータの更新または変更を試行して競合やロックの状態になる可能性があります。この問題を回避するには、ユーザーが同じ 2 フェーズ・コミット操作内の重複項目にアクセスできないようにアプリケーションを作成します。
- そのデータベースの最も小さな項目と同じ小ささのブロック・サイズを使用するように、内部リソース・ロック・マネージャー (IRLM) またはプログラム分離 (PI) ロック・マネージャーを構成することを検討してください。ブロック・サイズが大きいと、ディスク上で互いに近い場所にあるのみの、同一でもない項目を対象に 2 つのトランザクションが競合する可能性があります。
- グローバル・トランザクション (作業単位) 内で同じ IMS データベースに対して同じ IMS トランザクションを使用して複数の対話が実行される場合は、その

IMS トランザクションとの対話をそれぞれ別個の MPP 領域で実行する必要があります。IMS トランザクションを複数の MPP 領域で実行できることと、すべての対話を新規 MPP 領域で処理するためのスケジューリング要件 (メッセージの数がゼロより大きい) が常に満たされることを示すために、IMS トランザクションには SCHDTYP=PARALLEL と PARLIM=0 の値を指定する必要があります。

- 領域がハングしていて、実行タイムアウト値が設定されていない場合は、異常終了トランザクション・パラメーターを指定した /STOP REGION IMS コマンドを発行して、MPP 領域をハングさせているトランザクションの実行を終了することができます。例えば、/STOP REGION *reg#* ABDUMP *tranname* を指定します。このコマンドは、その特定の対話のトランザクションをロールバックして、MPP 領域を解放します。

その他のトランザクション・サポート

IMS TM リソース・アダプターは、ローカル・トランザクション、1 フェーズ・コミット処理、グローバルでないトランザクション処理、およびグローバル・トランザクション・スコープ内での会話型トランザクション処理もサポートしています。

ローカル・トランザクション

Java EE 接続アーキテクチャーでは、トランザクション・マネージャーではなくリソース・マネージャーがトランザクションをローカルで調整するための javax.resource.cci.LocalTransaction インターフェースが定義されます。ただし、IMS TM リソース・アダプターでは、トランザクション・マネージャーによるトランザクション調整のみがサポートされます。したがって、IMS TM リソース・アダプターでは javax.resource.cci.LocalTransaction インターフェースはサポートされません。IMSConnection.getLocalTransaction() メソッドを呼び出すと、NotSupportedException が返されます。IMS TM リソース・アダプターでトランザクション・サポートを使用するには、JTA トランザクション・インターフェースを使用するか、ご使用のアプリケーションのデプロイメント記述子で適切なトランザクション属性を設定する必要があります。詳しくは、クライアント・アプリケーションでのグローバル・トランザクション・サポートに関するトピックを参照してください。

1 フェーズ・コミット処理

IMS TM リソース・アダプターは、トランザクション・マネージャーによる 1 フェーズ・コミット最適化をサポートします。そのため、トランザクション・スコープ内のすべての変更が同じ IMS リソースに属する場合、トランザクション・マネージャーが、フェーズ 1 準備要求を送信しなくてもフェーズ 2 コミット要求を直接リソース・マネージャーに送信して変更をコミットできるように、1 フェーズ・コミット最適化を行う可能性があります。

グローバルでないトランザクション処理

アプリケーションでグローバル・トランザクション処理が使用されない場合 (例えば、トランザクション属性が TX_NOTSUPPORTED に設定されている場合)、グローバルでないすべてのトランザクション処理で「Sync-On-Return」(OTMA SyncLevel=None) が使用されます。IMS トランザクションがコミットされたときに

は、出力は既にクライアントに返されています。

グローバル・トランザクション・スコープにおける会話型トランザクション処理

IMS は、会話型プログラムを使用して、処理を、クライアントからプログラムへ、プログラムからクライアントへという、一連の関連した対話 (反復とも言います) に分割します。各反復はそれぞれ 1 つのタイプの IMS 会話型トランザクションです。会話型処理は、1 つのトランザクションに複数の部分が含まれる場合に使用されます。1 つの大規模なトランザクションを構成する各部分は、それぞれ別個にコミットまたはロールバックされます。

グローバル・トランザクション・スコープで会話型トランザクションを実行できるのは、以下の場合です。

- それぞれの反復が同一のトランザクション・レベルで実行されている。例えば、最初の反復がグローバル・トランザクション・スコープで処理された場合、その IMS 会話型トランザクション内の後続のすべての反復はグローバル・トランザクション・レベルで処理される必要があります。トランザクション・スコープを指定しないで 2 番目の反復を発行すると、IMS OTMA からエラーが報告されます。
- それぞれの反復が、その IMS 会話内の次の反復の発行前に、コミット呼び出しまたはロールバック呼び出しによって完了しなければならないようになっている。複数の反復を単一のグローバル・トランザクション・スコープにグループ化することはできません。

関連概念:

1009 ページの『グローバル・トランザクションと 2 フェーズ・コミット・サポート処理』

Common Client Interface (CCI)

Common Client Interface (CCI) を使用すると、IMS TM リソース・アダプターを使用して IMS と対話するアプリケーションを作成できます。

Rational または WebSphere 統合開発環境 (IDE) の J2C ウィザードによって生成されたアプリケーション・コードを使用して、IMS TM リソース・アダプターを介して IMS トランザクションにアクセスできます。このアプローチを使用する場合、コーディングは必要ありません。あるいは、IDE を使用せずにアプリケーション・ソース・コードを作成することもできます。

自分でコードを作成する場合は、CCI プログラミング・インターフェースを使用する必要があります。CCI API を使用すると、Java EE クライアント (例えば、エンタープライズ Bean、JavaServer Pages (JSP) ページ、サーブレット) からバックエンド・エンタープライズ情報システム (EIS) (例えば、IMS) へのアクセスが可能になります。

CCI プログラミング・インターフェース・モデルに従うアプリケーションは、使用されている EIS に依存しない共通構造を備えています。 Java EE コネクター・アーキテクチャー (JCA) 仕様は、そのアプリケーションに必要な 2 つのオブジェクトを定義します。

- EIS への接続を表す Connection オブジェクト
- それらの Connection オブジェクトを作成する ConnectionFactory オブジェクト

これらのオブジェクトは、アプリケーション・サーバーがリソース・アダプターのセキュリティー、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ために使用します。 IMS TM リソース・アダプター CCI プログラミング・インターフェースを使用するアプリケーションは、IMSConnectionFactory オブジェクトを取得することによって開始します。IMSConnectionFactory オブジェクトは以下の 2 つの方法で取得できます。

- 管理対象: アプリケーション・サーバーを使用する場合、IMSConnectionFactory オブジェクトは通常、WebSphere Application Server 管理コンソールなどの管理インターフェースを通じてリソース・アダプターから作成されます。このタイプの環境は管理対象環境 と呼ばれます。アプリケーション・サーバーを使用してその環境内の接続のサービス品質を管理するためです。例えば、WebSphere Application Server 管理コンソールを使用して、IMSConnectionFactory オブジェクトを作成し、そのカスタム・プロパティーを構成します。ターゲット IMS システムのホスト名やポート番号などのカスタム・プロパティーは、IMSConnectionFactory オブジェクトで構成されます。IMSConnectionFactory オブジェクトが作成されると、JNDI を通じてすべてのエンタープライズ・アプリケーションがそのオブジェクトを使用できるようになります。
- 非管理対象: 接続の管理にアプリケーション・サーバーを使用しない場合 (例えば、アプリケーションをスタンドアロン Java アプリケーションとして実行している場合)、そのタイプの環境は非管理対象環境 と呼ばれます。このタイプの構成では、以下の状況になります。
  - IMSManagedConnectionFactory オブジェクトを手動で作成し、そのカスタム・プロパティーを設定する必要があります。その後に、IMSManagedConnectionFactory オブジェクトから IMSConnectionFactory オブジェクトを作成できます。
  - IMS TM リソース・アダプター は、JCA 1.5 接続管理 API の DefaultConnectionManager クラスを使用して接続を行います。このクラスでは接続プーリングは行われないため、接続プール・プロパティーはすべて無視されます。 IMS TM リソース・アダプターは、トランザクション要求のたびに、IMS Connect とのソケット接続を開いて閉じます。ソケット接続を開いたり閉じたりするとリソースのオーバーヘッドが生じ、管理対象環境と比較してパフォーマンスが低下します。 PoolManager クラスを実装すると、DefaultConnectionManager クラスで使用する独自の接続プールを作成できます。詳しくは、JCA 1.5 接続管理 API のセクションを参照してください。

    CM0 トランザクションはリカバリー可能であるため、IMS Connect は CM0 を使用するクライアントごとに別個の TPIPE を作成します。アプリケーション・サーバーが提供する接続プール管理機能がない場合、作成される TPIPE の数が増えすぎて、システムが過負荷になります。

IMS 接続

作成された IMSConnectionFactory オブジェクトから IMSConnection オブジェクトを作成できます。IMSConnection オブジェクトのプロパティーは、

getConnection メソッドにパラメーターとして渡される IMSConnectionSpec オブジェクトで指定できます。そうでない場合は、IMSConnectionFactory で定義されたデフォルト値を使用します。IMSConnection が取得されると、その IMSConnection インスタンスから IMSInteraction インスタンスを作成できます。IMSInteraction インスタンスは、その接続で実行されることになる対話を表します。接続と同様に、対話も IMSInteractionSpec クラスからカスタム・プロパティーを取得できます。

入出力

対話を実行するために、アプリケーションは IMSInteraction オブジェクトの execute() メソッドを呼び出して、データを保持する入出力オブジェクトをそれに渡します。 IMS への入力メッセージの各フィールドの値を含む、入力バイト配列を作成する必要があります。同様に、IMS によって返される応答メッセージを保持する出力バイト配列も作成する必要があります。出力メッセージの各フィールドの値は、出力バイト配列から抽出されます。

入力バイト配列と出力バイト配列はユーザー自身で作成できます。あるいは、Rational または WebSphere 開発環境の J2C オプションを使用して CCI アプリケーションの入出力メッセージ用の Java データ・バインディングを作成することもできます。

IMS システムからデータをリトリーブするアプリケーションの要件は、以下のとおりです。

- IMSConnection オブジェクトを作成するには IMSConnectionFactory オブジェクトを使用します。
- IMSInteraction オブジェクトを作成するには IMSConnection オブジェクトを使用します。
- バックエンド IMS システムでトランザクションを実行するには IMSInteraction オブジェクトを使用します。
- IMSInteraction オブジェクトと IMSConnection オブジェクトは閉じてください。

関連情報:

IMS TM Resource Adapter バージョン 15 Java API 仕様

CCI アプリケーションのサンプル・コード

CCI アプリケーションの以下のサンプル・コードでは、接続後に IMSInteraction オブジェクトを取得して、IMS トランザクションを実行する方法を示しています。

管理対象環境では、CCI API を使用して JNDI 名前空間から IMSConnectionFactory インスタンスを検索し、それを使用して IMSConnection インスタンスを取得します。

```
ConnectionFactory cf = null;
if (isManaged) {
   //Use JNDI lookup to get ConnectionFactory instance
   //Assume the connection factory has a JNDI name of MyIMS
   Context ic = new InitialContext();
   cf = (ConnectionFactory) ic.lookup("MyIMS");
```

JNDI が構成されていない場合、CCI アプリケーションは手動で IMSManagedConnectionFactory オブジェクトを構成し、それを使用して接続ファクトリーを取得することができます。

```
IMSManagedConnectionFactory mcf = new IMSManagedConnectionFactory();
mcf.setDataStoreName("MyDSName");
mcf.setHostName("myHostNm");
mcf.setPortNumber(new Integer(1234));
...
//Create connection factory from ManagedConnectionFactory
cf = (IMSConnectionFactory) mcf.createConnectionFactory();
```

以下の例は、EIS でコマンドを実行するための CCI インターフェースの使用法を示しています。

- IMSConnectionFactory オブジェクトを使用して、IMSConnection オブジェクトを作成します。
- IMSConnection オブジェクトを使用して、IMSInteraction オブジェクトを作成します。
- IMSInteraction オブジェクトを使用して、バックエンド IMSシステムでトランザクションを実行します。
- IMSInteraction オブジェクトおよび IMSConnection オブジェクトを閉じます。

```
public void execute() {
   try {
      ConnectionFactory cf = null;
      if (isManaged) {
         //Use JNDI lookup to get ConnectionFactory instance - assumes
         //connection factory has JNDI name of MyIMS
         Context ic = new InitialContext();
         cf = (ConnectionFactory) ic.lookup("MyIMS");
      } else {
         //Create and set values for ManagedConnectionFactory
         IMSManagedConnectionFactory mcf = new IMSManagedConnectionFactory();
         mcf.setDataStoreName("MyDSName");
         mcf.setHostName("myHostNm");
         mcf.setPortNumber(new Integer(1234));
         //Create connection factory from ManagedConnectionFactory
         cf = (IMSConnectionFactory) mcf.createConnectionFactory();
      }
      // Create an IMSConnection object
      Connection connection = cf.getConnection();

      //Create an IMSInteraction from the connection to
      //interact with IMS to run the IVTNO transaction (Phonebook)
      IMSInteraction interaction = (IMSInteraction) connection.createInteraction();
      IMSInteractionSpec ixnSpec = new IMSInteractionSpec();
      ixnSpec.setInteractionVerb(IMSInteractionSpec.SYNC_SEND_RECEIVE);

      //Create new input record
      input = new PhoneBookInputRecordField("cp037");
      input.setIn__ll((short)59);
      input.setIn__zz((short) 0);
      input.setIn__trcd("IVTNO");
      input.setTranCodeLength(10);
      input.setIn__command("DISPLAY");
      input.setIn__name1("LAST3");
      input.setIn__name2("");
      input.setAllFieldsGiven(false);
      PhoneBookOutputRecordField

      //Create new output record
      output = new PhoneBookOutputRecordField("cp037");
```

```
      //Execute interaction by calling the execute() method
      interaction.execute(ixnSpec, input, output);

      //Display output
      System.out.println ("Output is: ");
      System.out.println("\nMessage: "
         + output.getOut__mesg()
         + "\nName:"
         + output.getOut__name1()
         + " "
         + output.getOut__name2()
         + "\nExtension: "
         + output.getOut__extn()
         + "\nZipcode: "
         + output.getOut__zip());

   } catch (Exception e) {
      e.printStackTrace();
      } finally {
      //Close both the interaction and the connection
      interaction.close();
      connection.close();
   }
}
```

サンプルおよびチュートリアル

IMS トランザクションにアクセスする Java アプリケーションまたは Web サービスを開発するためのサンプルおよびチュートリアルは、いくつかの WebSphere 統合開発環境および Rational 統合開発環境で参照できます。

- さまざまなタイプの IMS トランザクション用に J2C アプリケーションを作成するためのチュートリアルおよびサンプルは、各種の WebSphere 開発環境および Rational 開発環境のオンライン・ヘルプで入手可能です。
- WebSphere Integration Developer ユーザーは、WebSphere Integration Developer インフォメーション・センターおよび資料で、IMS トランザクションにアクセスするためのインポートの作成例を参照できます。
- WebSphere Transformation Extender ユーザーは、さまざまなタイプの IMS トランザクション用のマップを作成するためのチュートリアルを参照できます。
- エンドツーエンドのコールアウト IVP サンプルが、IMS および IMS TM リソース・アダプターと一緒に提供されます。
  - IMS では、IMS コールアウト・アプリケーションおよび必要な IMS OTMA 宛先記述子が含まれている IMS インストール検査プログラム (IVP) にジョブおよびタスクが提供されています。OTMA 宛先記述子は、コールアウト要求がキューに入れられる TPIPE を定義します。
  - IMS TM リソース・アダプターでは、IMS からのコールアウト要求がないか listen するサンプル・アプリケーションが用意されています。

関連情報:

Rational Application Developer V9 の Knowledge Center にあるチュートリアル

Rational Application Developer V9 の Knowledge Center にあるサンプル

第 49 章 スタンドアロン WebSphere Application Server でのアプリケーションの実行

スタンドアロン WebSphere Application Server を使用して、アプリケーションをテストできます。最初に、Java EE アプリケーションをエンタープライズ・アプリケーション・アーカイブ (EAR) ファイルとしてエクスポートしてから、その EAR ファイルをサーバーにインストールする必要があります。

前提条件**:** IMS TM リソース・アダプターをエンタープライズ・アプリケーションで圧縮しない場合、以下のことを確認する必要があります。

- アプリケーションが使用する IMS TM リソース・アダプターは、スタンドアロン WebSphere Application Server にデプロイされていること。
- IMSトランザクションを実行する IMS Connect および IMSへの接続を作成するように接続ファクトリーが定義されていること。

前提条件が満たされたら、WebSphere Application Server への EAR ファイルのインストールに進むことができます。

関連タスク:

893 ページの『WebSphere Application Server へのリソース・アダプターのインストール』

895 ページの『WebSphere Application Server での接続ファクトリーの作成』

WebSphere サーバーへの EAR ファイルのインストール

アプリケーションを EAR ファイルとしてエクスポートし、そのファイルを WebSphere サーバーにデプロイした後、そのアプリケーションを実行できます。

前提条件**:** IMS TM リソース・アダプター RAR をインストールし、IMS TM リソース・アダプター用の接続ファクトリーを作成済みである必要があります。

アプリケーション EAR ファイルをインストールするための手順は、IMS TM リソース・アダプター IVP EAR ファイルをデプロイする方法と似ています。アプリケーション EAR ファイルをインストールしてアプリケーションを実行するには、以下のトピックで説明している手順に従ってください。

- 900 ページの『WebSphere Application Server への IVP EAR ファイルのデプロイ』
- 901 ページの『WebSphere Liberty サーバーへの IVP EAR ファイルのデプロイ』

第 50 章 問題の診断

エラーは、メッセージ要求または応答のトランザクションに関係するさまざまな原因で発生します。コンポーネント固有の情報をログに記録し、トレースするようにアプリケーション・サーバーを構成できます。また、IMS TM リソース・アダプターも、Java 例外からメッセージを生成します。

IVP 障害の診断

IMS TM リソース・アダプター IVP の実行時にエラーが発生する場合は、まず、WebSphere Application Server ログの情報を確認します。

WebSphere Application Server ログは、*WebSphere_install_directory*¥AppServer¥logs の下にあります。trace.log ファイルでは例外がリストされ、障害の診断に役立つスタック・トレースが提供されます。

IVP 障害の原因として、次のような問題が考えられます。

- WebSphere Application Server が正しく開始されていない。
- クラスター環境でコンテキストに RAR 参照が検出されないことについて以下の ResourceException エラーが示された場合は、ノード・レベルでの RAR のインストールに加え、適切なクラスター有効範囲またはサーバー有効範囲を持つ RAR ファイルのコピーを作成してあるか確認します。

  ```
  Error 500: javax.resource.ResourceException:
  Context: myCell/clusters/myCluster1, name: myIMSTMRARef:
  First component in name myIMSTMRARef not found.
  ```

- IVP のエンタープライズ・アプリケーション IMSICOIVPServiceEAR が開始されていない。エンタープライズ・アプリケーションの状況を判別するには、以下のことを実行します。
  1. WebSphere Application Server 管理コンソールで、コンソールの左ペインのナビゲーション・ツリーで「**Applications**」を展開します。
  2. 「**Enterprise Applications**」リンクをクリックします。インストール済みアプリケーションの 1 つとして IMSICOIVPServiceEAR EAR ファイルが表示されます。緑の矢印は、そのアプリケーションが開始済みであることを示します。目的のアプリケーションが開始されていない場合は、 900 ページの『Java EE アプリケーション・サーバーでの IVP EAR ファイルのデプロイ』のトピックに記載されているアプリケーションの開始手順に従います。
- 接続ファクトリーの構成時に、無効なデータが指定された。以下に例を示します。
  - ホスト名のつづりが間違っているか、またはホスト名の修飾が不十分である (TCP/IP 通信の場合)。
  - ターゲット IMS Connect に間違ったポート番号が指定されている (TCP/IP 通信の場合)。
  - IMS Connect 名のつづりに誤りがある。

  - ターゲット IMS のデータ・ストア名が無効であるか、またはデータ・ストア名のつづりに誤りがある。データ・ストア名は、大文字で指定する必要があります。
- IMS が実行していない。
- IMS Connect が実行していない。
- IMS Connect ポートがアクティブでない。IMS Connect コマンド VIEWHWS を使用して、該当のポートがアクティブであるかどうかを判別します。IMS Connect ポートを活動化するには、IMS Connect コマンド OPENPORT を使用します。
- ターゲット IMS データ・ストアがアクティブでない。IMS Connect コマンド VIEWHWS を使用して、該当のデータ・ストアがアクティブであるかどうかを判別します。IMS データ・ストアを活動化するには、IMS Connect コマンド OPENDS を使用します。
- TCP/IP に障害がある。ping コマンドを発行してから IVP を実行して、接続が機能していることを確認します。
- ホスト上で実行中の IMS Connect のレベルが正しくない。必要な IMS Connect のレベルについては、 899 ページの『IVP を実行するための前提条件』を参照してください。
- IVP からメッセージ DFS058I `hh:mm:ss START COMMAND COMPLETED` が返されることを予期していたが、代わりに、メッセージ DFS1292E `SECURITY VIOLATION` が返された場合は、以下の状態を確認します。
  - IVP 接続ファクトリーのカスタム・プロパティーに提供したユーザー名とパスワードは、IVP が IMS に対して発行する /STA OTMA コマンドを実行する権限を与えられていない。
  - IMS Connect で Security が有効になっていない (VIEWHWS コマンド出力で RACF=Y)。
  - IMS OTMA で Security が有効になっていない (/DIS OTMA コマンド出力で SECURITY=FULL)。
- 以下の例外を受け取った場合は、IVP が使用する接続ファクトリーが CM0Dedicated プロパティーの値を`false` に設定するよう構成する必要があります。

```
javax.resource.ResourceException: ICO0087E:
com.ibm.connector2.ims.ico.IMSTCPIPManagedConnection@28f39301.call(Connection,
InteractionSpec, Record, Record) error. Protocol violation. Commit Mode 1 is not
allowed for interactions on a dedicated persistent socket
```

Java アプリケーションから IMS にアクセスする場合の問題の診断

Java アプリケーションから IMS にアクセスできない場合は、IMS TM リソース・アダプターが適切にセットアップおよび構成されていること、IMS Connect および IMS がアクティブであること、さらに、ping コマンドを使用してホスト・システムにアクセスできることを確認します。

- IMS TM リソース・アダプターを使用するためのセットアップおよび構成が正しいことを確認します。サポートされる構成については、サポートされるバージョン、構成、およびプラットフォームに関するトピックを参照してください。

- 未解決の IMS Connect 応答 HWSC0000I *IMS CONNECT READY* *ims_connect_name* がターゲット・システムのシステム・コンソールに表示されていることを確認することにより、IMS Connect がアクティブであることを検証します。IMS Connect がアクティブなときにメッセージ HWSC0000I がシステム・コンソールに表示されるようにするために、IMS Connect HWSCFG*xx* 構成メンバーの HWS ステートメントに **WTORCMD=Y** が指定されていること、または **WTORCMD** パラメーターがその HWS ステートメントから省略されていることを確認します。
- IMS Connect の未処理応答で IMS Connect コマンド VIEWHWS を入力して、ポートおよびデータ・ストアがアクティブであることを確認します。
- 未解決の IMS 応答 DFS996I *IMS READY* がターゲット・システムのシステム・コンソールに表示されていることを確認することにより、IMS がアクティブであることを検証します。
- 未解決の IMS 応答に IMS コマンド /DISPLAY OTMA を入力することにより、IMS および IMS Connect の両方のメンバーのシステム間カップリング・ファシリティー (XCF) サービス状況が「ACTIVE」 (アクティブ) であることを検証します。表示出力は、以下の出力のようになります。

```
DFS000I     GROUP/MEMBER      XCF-STATUS   USER-STATUS
SECURITY    IMS1

DFS000I     XCFGRPNM
    IMS1

DFS000I     -IMSNAME          ACTIVE       SERVER           FULL
    IMS1

DFS000I     -ICONNAME         ACTIVE       ACCEPT TRAFFIC
    IMS1

DFS000I     *02033/143629*   IMS1
```

- Javaアプリケーションと IMSConnect との間の通信に TCP/IP を使用している場合は、ping コマンドを使用して、ターゲット・ホスト・システムに正常にアクセスできることを検証します。ホスト・システムを ping できず、IP アドレスではなくホスト名を使用している場合、ホスト名の修飾が十分であることを確認してください。

IMS サービスが IMS トランザクションからの予期された出力を提供していない場合は、IMS アプリケーション・プログラムが返す出力メッセージが、IMS サービスが使用する出力 COBOL 定義に一致することを確認してください。Java EE アプリケーションの場合、traceLevel プロパティーを 3 に設定することにより、IMS アプリケーション・プログラムによって返されたメッセージが含まれている IMS OTMA メッセージを表示できます。IMS TM リソース・アダプター トレースをオンにする方法の手順については、IMS TM リソース・アダプターを使用したロギングとトレースに関するトピックを参照してください。

関連タスク:

1026 ページの『IMS TM リソース・アダプター情報のロギングとトレース』

コールアウト要求での問題の診断

コールアウト要求エラーは、多くの場合、実行タイムアウトの設定、コールアウト・メッセージの誤りまたは破損、あるいはネットワーク障害に関連しています。

- 実行タイムアウトが発生します (この時間間隔内ではコールアウト・メッセージはリトリーブされません)。

  説明: 次のような原因が考えられます。

  – IMS アプリケーションはコールアウト要求を出さず、TPIPE は要求を保持しない。
  – Java アプリケーションの代替クライアント ID フィールドに指定された TPIPE 名が、コールアウト要求に対して IMS アプリケーションが使用する TPIPE 名と一致しない。
  – Java アプリケーションの代替クライアント ID フィールドに指定された TPIPE 名が、OTMA 宛先記述子に指定された TPIPE 名と一致しない。
  – メッセージ処理プログラム (MPP) が開始されていない。

  ユーザー処置: タイムアウト間隔 (executionTimeout プロパティー)を増やすか、または TPIPE 名が一致していることを確認してください。MPP が開始されていることを確認します。
- IMS Connect は、コールアウト要求の処理に失敗し、メッセージはデッド・レター・キューに入れられます。

  説明: この問題は、誤ったコールアウト・メッセージが原因である可能性があります。

  ユーザー処置: IMS アプリケーションのコールアウト・メッセージを修正します。
- IMS Connect は、IMS TM リソース・アダプターにコールアウト要求を送信できません。

  説明: この問題は、ネットワーク障害が原因である可能性があります。

  ユーザー処置: ネットワーク障害が発生した場合、Java 例外を受け取ることがあります。この例外をキャッチするように Java アプリケーションを変更します。
- IMS TM リソース・アダプターまたは Java アプリケーションがコールアウト要求の処理に失敗しました。

  説明: 次のような原因が考えられます。

  – コールアウト要求が破損している。
  – メッセージは、コールアウト要求ではなく、通常の非同期出力メッセージである。

  ユーザー処置: コールアウト要求が IMS アプリケーションから正しく発行されているか確認します。Java アプリケーションに指定された TPIPE 名が正しいものであり、通常の非同期出力メッセージに対する別のキューの名前でないことを確認してください。

出力メッセージを含む Java 例外

IMS TM リソース・アダプターがメッセージを IMS Connect を介して IMS に渡す場合、および IMS がこのトランザクションを処理して出力を返そうとする場合に、プロセスで何らかのエラーが生じると、Java クライアントは例外を受け取ります。

スローされる例外のタイプは、出力メッセージのリトリーブが可能かどうかを示します。例えば、トランザクションが停止していることを示す IMSDFSMessageException 例外を Java クライアントが受け取った場合は、アプリケーションが実行されていません。したがって、リトリーブできる出力メッセージはありません。ただし、トランザクションが実行されていても、出力メッセージが IMS Connect に返される前に executionTimeout 値の期限に達すると、Java クライアントは EISSystemException 例外を受け取ります。この例外は実行タイムアウトが発生したことを示します。この場合、出力メッセージは後でリトリーブできるように該当する IMS OTMA 非同期出力キューまたは TPIPE に入れられます。

非同期出力対話エラー

通常、非同期出力対話 SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT および SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT を使用して、クライアント ID 用のキューに入れられた出力メッセージをリトリーブできます。それらのメッセージがどのように関連するクライアント ID のキューに入れられたかは関係ありません。失敗したコミット・モード 0 トランザクションによるメッセージ、または代替プログラム連絡ブロック (ALTPCB) への INSERT 呼び出しを発行した IMS アプリケーションからのメッセージが、キューに入れられる可能性があります。

例えば、失敗したコミット・モード 0 トランザクションからの出力メッセージをリトリーブする場合、リトリーブ要求の IMSConnectionSpec クラスに指定されたクライアント ID は、失敗したコミット・モード 0 トランザクションに指定されたクライアント ID と一致する必要があります。その特定のクライアント ID の OTMA 非同期出力キューに入っているメッセージがない場合、実行タイムアウト例外が発生します。このタイムアウト例外は、以下の 2 つのシナリオのいずれかを意味します。

- キューにメッセージがない。
- そのタイムアウト値では、IMS Connect がキューからメッセージをリトリーブする時間が足りない。

どちらの非同期出力対話でも、実行タイムアウトは、IMS Connect が IMS からの応答を待つ時間です。リトリーブ要求に実行タイムアウト値を指定しない場合、デフォルトの実行タイムアウト値が使用されます。デフォルトのタイムアウト値は、IMS Connect 構成メンバーの TIMEOUT の値です。どのタイプの対話でも出力メッセージが返されるようにするために、実行タイムアウト値を試してみることが必要と考えられます。

関連概念:

954 ページの『コミット・モードと同期レベルの組み合わせによってサポートされる対話』

921 ページの『非同期出力プログラミング・モデル』

989 ページの『IMS 保留キューからのメッセージのリトリーブの保護』

IMS TM リソース・アダプター情報のロギングとトレース

IMS TM リソース・アダプターについてメッセージをログに記録し、診断トレース情報を記録するには、WebSphere Application Server でロギングとトレースの設定を構成します。

アプリケーション・ロギングは、アプリケーションの実行中に発生した各種イベントをキャプチャーします。IMS TM リソース・アダプターは、WebSphere Application Server に組み込まれている java.util.logging パッケージで作成される Java ロギング・サポートを提供します。Java メッセージ・ロギングはトレース・ログに記録されるので、トラブルシューティングのサポート機能を記述するために、ロギングとトレースは一緒に使用されます。メッセージ・ログはエンド・ユーザー、アプリケーション開発者、またはシステム管理者のためのものですが、診断トレース情報は、技術サポート担当者が、発生したイベントに関するより厳密で詳細な情報を入手するためのものです。

WebSphere Application Server では、すべてのコンポーネントについてのトレースは、デフォルトでは使用不可になっています。ロギングとトレースの設定を構成した後、トレースは使用可能になります。トレース・ファイルは、ロギングとトレースが使用可能な場合に作成され、Java アプリケーションは WebSphere Application Server で実行されます。

スタンドアロン・アプリケーションでは、java.util.logging.Logger オブジェクトを使用して独自ロガーを作成し、FileHandler を作成し、ログ・ファイルを FileHandler に渡し、ロガーを ManagedConnectionFactory オブジェクトに設定することができます。

ヒント**:** トレースがオンになっているときは、1 つのクライアントのみが実行していることを確認してください。

WebSphere Application Server でのロギングとトレース

WebSphere Application Server で IMS TM リソース・アダプターの情報のロギングとトレースをオンにするには、トレース出力をファイルまたはメモリー・バッファーに設定して、IMS TM リソース・アダプターの情報を含むパッケージにログ・レベルを指定します。

IMS TM リソース・アダプター情報のロギングとトレース用に制御を設定するには、次のようにします。

1. WebSphere Application Server管理コンソールのウェルカム・ページで、「**Troubleshooting**」を展開し、**Logs and Trace**」をクリックして「Logging and Tracing」ページを開きます。
2. 「Logging and Tracing」ページで、トレースするサーバー (例えば server1) をクリックして、「**Diagnostic Trace**」をクリックします。 「Diagnostic Trace Service」ページが開きます。
3. 「**Configuration**」タブで、「Trace Ouput」を「**Memory Buffer**」または「**File**」に設定します。

ヒント:

- 「**Configuration**」タブで行った変更は、サーバーの再始動後も持続します。トレース・レベルを一時的に変更するには、「**Runtime**」タブで変更を行います。
- トレース情報は、レベル「fine」、「finer」、および「finest」のイベントであり、トレース・ログにのみ書き込むことができます。異なるログ・レベルの相違点については、WebSphere Application Server バージョン 8 インフォメーション・センターの『ログ・レベル設定情報』 (the log level setting information in the WebSphere Application Server Version 8 Information Center) を参照してください。

4. 「**File**」を選択してトレース出力を保管する場合は、トレース出力のデフォルトの名前と場所を受け入れることもできますし、変更することもできます。デフォルトの名前と場所を変更するには、「File Name」フィールドにファイルの別の名前と場所を入力します。
5. 「**OK**」をクリックする。
6. 「Logging and Tracing」ページで、「**Change Log Detail Levels**」をクリックします。 「Change Log Detail Levels」ページが表示されます。
7. IMS TM リソース・アダプターのロギングとトレースを使用可能にするには、次のようにします。
   a. 「General Properties」の下で、「**Components**」をクリックします。
   b. テキスト・ボックスに以下のストリングを入力します。

   ```
   *=info: com.ibm.j2ca.RAIMSTM=finest
   ```

   そのストリングを他のストリングと結合して、他のコンポーネントでのトレースを使用可能にすることができます。
8. 「**OK**」をクリックし、「**Configuration**」タブで変更を行っている場合は、そのページの上部にある「**Save**」をクリックして、マスター構成に対する変更を保管します。
9. Java アプリケーションを実行してから、トレース・ファイルを調べます。

WebSphere Liberty でのロギングとトレース

WebSphere Liberty で IMS TM リソース・アダプターの情報のロギングとトレースをオンにするには、IMS TM リソース・アダプターの情報を含むパッケージにログ・レベルを指定します。

IMS TM リソース・アダプターのロギングとトレース情報を出力するには、`server.xml` ファイルに `<logging>` エントリーを追加します。

1. `server.xml` ファイルを開きます。
2. traceSpecification エレメントに com.ibm.j2ca.RAIMSTM パッケージを指定して IMS TM リソース・アダプターの`<logging>` を追加し、トレース・レベルを FINEST に設定します。

   ```
   <logging traceSpecification="*=info:com.ibm.j2ca.RAIMSTM=finest"></logging>
   ```

   WebSphere Liberty サーバーのロギングとトレース構成、ログ・レベル設定およびそれぞれの内容、さらに 1 次ログ・ファイルについて詳しくは、WebSphere Liberty サーバーのバージョン用の資料を参照してください。

関連情報:

Liberty プロファイルのロギングとトレース (V8.5.5)

ファイルに送信された出力を使用するスタンドアロン・ロガーの作成

このコード・サンプルでは、ファイルに送信された出力を使用してスタンドアロン・ロガーを作成する方法を示しています。

このサンプルでは、java.util.logging.Logger オブジェクトを使用し、ロガーを ManagedConnectionFactory オブジェクトに設定し、ログ・レベルを FINEST に設定します。

1. アプリケーションで、以下のように、java.util.logging.Logger オブジェクトおよび FileHandler を作成し、出力ファイルを FileHandler に渡し、ロガーを ManagedConnectionFactory オブジェクトに設定して、ログ・レベルを設定します。

```
com.ibm.connector2.ims.ico.IMSManagedConnectionFactory mcf =
        new com.ibm.connector2.ims.ico.IMSManagedConnectionFactory();
mcf.setHostName("yourHostName");
mcf.setDataStoreName("yourDataStore");
mcf.setPortNumber(new Integer(yourPortNumber));

// Create a file logger
Logger logger = Logger.getLogger("myLogger");
FileHandler fh = new FileHandler("output.txt");
SimpleFormatter formatter = new SimpleFormatter();
fh.setFormatter(formatter);
logger.addHandler(fh);

// Set the file logger to the IMSManagedConnectionFactory object
// Set log level to finest
LogUtils logUtils = new LogUtils(logger, "log", "yourProduct", "yourVersion");
mcf.setLogUtil(logUtils);
mcf.getLogUtil().setLoggingLevel(Level.FINEST);
connFactory = (ConnectionFactory) mcf.createConnectionFactory();
```

2. アプリケーションの他のコードを追加します。
3. アプリケーションを実行します。

アプリケーションが実行されると、情報は output.txt ファイルに記録されます。

トレース・データの分析

WebSphere Application Server の trace.log ファイルのトレース・データには、ロギングとトレースに関して構成された設定値に基づいて、基底クラスおよび追加クラスによるメソッド呼び出しの時刻と順番が格納されます。

トレース出力のフォーマットは、以下の内容で構成されています。

- タイム・スタンプ
- スレッド ID
- ロギング・コンポーネントの省略した短縮名
- イベント・タイプ標識
- メッセージまたはトレース・イベントを発行したクラス

- 追加のテキスト・メッセージ

以下のトレース項目は、WebSphere Application Server が IMS TM リソース・アダプター 用に正しく構成されているときにログに記録される情報を示しています。

- RAIMSTM は、IMS TM リソース・アダプターの短縮名です。
- 重要なイベントが発生するたびに、その入り口と出口の情報がログに記録されます。

```
[7/5/11 14:24:04:968 PDT] 00000012 RAIMSTM       2
         com.ibm.ims.ico.IMSTCPIPAdapter generateClientID() Entering method.
[7/5/11 14:24:04:968 PDT] 00000012 RAIMSTM       3
         com.ibm.ims.ico.IMSTCPIPAdapter generateClientID()
         LocalPort = [3107]
[7/5/11 14:24:04:968 PDT] 00000012 RAIMSTM       3
         com.ibm.ims.ico.IMSTCPIPAdapter generateClientID()
         IP Address = [9030020219]
[7/5/11 14:24:04:968 PDT] 00000013 RAIMSTM       3
         com.ibm.ims.ico.IMSTCPIPAdapter generateClientID()
         Generated ID  = [HWSY6N4P]
[7/5/11 14:24:04:968 PDT] 00000013 RAIMSTM       <
         com.ibm.ims.ico.IMSTCPIPAdapter generateClientID() Exiting method.
[7/5/11 14:24:04:968 PDT] 00000013 RAIMSTM       3
         com.ibm.ims.ico.IMSTCPIPAdapter connect()
         <-- [com.ibm.ims.ico.IMSTCPIPAdapter@1eca167.connect()]
```

IMS ホスト・システムへの接続が成功すると、メッセージが IMS に送信され、クライアント ID が生成されることがトレース出力に記録されます。

```
[7/5/11 14:24:04:968 PDT] 00000013 RAIMSTM       2
         com.ibm.connector2.ims.ico.inbound.IMSInboundUtil .sendIMSMessage()
         Entering method.
[7/5/11 14:24:04:968 PDT] 00000013 RAIMSTM       3
         com.ibm.connector2.ims.ico.inbound.IMSInboundUtil .sendIMSMessage() Mode is:
         MODE_RECEIVE_ASYNCOUTPUT_AUTO
[7/5/11 14:24:04:968 PDT] 00000012 RAIMSTM       3
         com.ibm.ims.ico.IMSTCPIPAdapter generateClientID()
         Generated ID  = [HWSYEW06]
[7/5/11 14:24:04:968 PDT] 00000012 RAIMSTM       <
         com.ibm.ims.ico.IMSTCPIPAdapter generateClientID() Exiting method.
[7/5/11 14:24:04:968 PDT] 00000012 RAIMSTM       3
         com.ibm.ims.ico.IMSTCPIPAdapter connect()
         <-- [com.ibm.ims.ico.IMSTCPIPAdapter@1ee8e00.connect()]
```

以下のトレース項目は、IMS (ホスト dev555.vmec.ibm.com、ポート 9999) への接続が失敗した場合を示しています。ICO0003E メッセージがログに記録されます。接続に失敗した場合、メッセージは送信されません。

```
connect() ->  [com.ibm.ims.ico.IMSTCPIPAdapter@2037bc2.connect():
          HostName=dev555.vmec.ibm.com PortNumber=9999] SocketTimeout = [0]
[7/5/11 14:24:16:734 PDT] 00000021 RAIMSTM       E
         com.ibm.connector2.ims.ico.IMSTCPIPManagedConnection connect() ICO0003E:
         com.ibm.connector2.ims.ico.IMSTCPIPManagedConnection@2037bae.connect() error.
         Failed to connect to host [dev555.vmec.ibm.com], port [9999].
         [java.net.UnknownHostException: dev555.vmec.ibm.com]
         Explanation=The IMS TM resource adapter was unable to connect to the host and
         port combination. java_exception indicates the reason for the failure to
         connect.
         UserAction=Examine the exception to determine the reason for the failure to
         connect to the host.
[7/5/11 14:24:16:734 PDT] 00000021 RAIMSTM       2
         com.ibm.connector2.ims.ico.IMSManagedConnection errorOccurred(Exception)
         Entering method.
[7/5/11 14:24:16:734 PDT] 00000021 ConnectionEve W
   J2CA0206W: A connection error occurred.
         To help determine the problem, enable the Diagnose Connection Usage option
```

```
        on the Connection Factory or Data Source.
[7/5/11 14:24:16:734 PDT] 00000021 ConnectionEve A
   J2CA0056I: The Connection Manager received a fatal connection error from
        the Resource Adapter for resource
```

接続に成功した場合、メッセージの送信時に、トレース・データは、送信されたバッファーを示します。この情報は、IMS Connect のレコーダー・トレースに出力されるものと同じ情報です (出口ルーチンをカスタマイズした場合は除く)。

```
 [7/5/11 14:24:04:984 PDT] 00000012 RAIMSTM       2
        com.ibm.ims.ico.IMSTCPIPAdapter send(byte[]) Entering method.
[7/5/11 14:24:04:984 PDT] 0000000b ApplicationMg A   WSVR0221I: Application started:
        query
[7/5/11 14:24:04:984 PDT] 0000000b CompositionUn A   WSVR0191I: Composition unit
        WebSphere:cuname=query in BLA WebSphere:blaname=query started.
[7/5/11 14:24:04:984 PDT] 00000013 RAIMSTM       3
        com.ibm.ims.ico.IMSTCPIPAdapter send(byte[]) Buffer sent:
 [
  000001f2 001c0100 5cc8e6e2 d1c1e55c |...2. ..*HWSJAV*| : 16
  00000000 c0ff0000 c8e6e2e8 f6d5f4d7 |....{...HWSY6N4P| : 32
  01100000 28004040 40404040 4040a0e0 |......        μ\| : 48
  00000000 00000000 00000000 00010000 |................| : 64
  00480240 01104040 40404040 40400000 |.ç. ..        ..| : 80
  00000000 00000000 00000000 00000000 |................| : 96
  00000000 00000000 00000000 00000000 |................| : 112
  00000000 00000000 00000000 00004040 |..............  | : 128
  40404040 40400000 006ac614 09024040 |      ...¦F...  | : 144
  40404040 40400903 40404040 40404040 |      ..        | : 160
  00000000 00000000 00000000 00000000 |................| : 176
  00000000 00000000 00000000 00000000 |................| : 192
  00000000 00000000 00000000 00000000 |................| : 208
  00000000 00000000 00000000 00000000 |................| : 224
  00000000 00000000 00000000 00000000 |................| : 240
  00000100 0000c9d4 e2f14040 4040c8e6 |......IMS1    HW| : 256
  e2e8f6d5 f4d70000 00000000 00000000 |SY6N4P..........| : 272
  00000000 00000000 00000000 00000000 |................| : 288
  00000000 00004040 40404040 40401040 |......        .| : 304
  00000000 00004040 40404040 40400000 |......        ..| : 320
  00000300 00000000 00000000 0000e3d4 |..............TM| : 336
  d9c1e3d7 f4400000 00000000 00000000 |RATP4 ..........| : 352
  00000000 00000000 00000000 00000000 |................| : 368
  00000000 00000000 00000000 00000000 |................| : 384
  00000000 00000000 00000000 00000000 |................| : 400
  00000000 00000000 00000000 00000000 |................| : 416
  00000000 00000000 00000000 00000000 |................| : 432
  00000000 00000000 00000000 00000000 |................| : 448
  00000000 00000000 00000000 00000000 |................| : 464
  00000000 00000000 00000000 00000000 |................| : 480
  00000000 00000000 00000000 00000000 |................| : 496
  0000                                |..|
 ]
[7/5/11 14:24:04:984 PDT] 00000013 RAIMSTM
       < com.ibm.ims.ico.IMSTCPIPAdapter send(byte[]) Exiting method.
```

トレースの処理方法について詳しくは、WebSphere Application Server インフォメーション・センターのトラブルシューティングおよびサポート情報を参照してください。

関連情報

WebSphere Application Server バージョン 8 インフォメーション・センターのトラブルシューティングおよびサポート情報

IMS TM リソース・アダプターのメッセージおよび例外

IMS TM リソース・アダプターのメッセージは、IMS TM リソース・アダプター自体によって、または IMS TM リソース・アダプターが使用するクラス・ライブラリー (Java クラス・ライブラリーなど) によってスローされる Java 例外で、接頭部 ICO が付きます。

このトピックでは、IMS TM リソース・アダプター J2C アプリケーションによって生成される例外について説明します。

後述のメッセージの説明にイタリック で示されている以下の用語は、実行時に具体的な値に置き換えられます。

hostname
IMS Connect を実行しているシステムの TCP/IP ホスト名。

java_exception
スローされた Java 例外。

length
データの長さ。

libraryFileName
ローカル・オプションのネイティブ・ライブラリーのファイル名。

llvalue
LL の値。

maxlength
データの最大有効長。

methodname
この例外をスローしているメソッドの名前。

モード
IMS TM リソース・アダプターとホスト上の IMS Connect コンポーネントの間で行われる対話のタイプ (interactionSpec オブジェクト内で定義される)。

nativeMethodName
ローカル・オプションのネイティブ・メソッド名。

portnumber
IMS Connect に割り当てられたポート番号。

propertyname
プロパティーの名前。

propertyvalue
プロパティーの値。

reasoncode
IMS Connect から返される理由コード。

OTMA センス・コードの場合、理由コード 0 は、センス・コードに関連付けられた理由コードがないことを示します。

rectype
レコードのタイプ。

returncode
IMS Connect によって返される、10 進数形式の戻りコード。

sensecode
IMS OTMA から返される、10 進数形式のセンス・コード。

socketexception
ソケット例外。

source_exception
エラーが最初に内部メソッドで発生したときにスローされた例外。

source_methodname
エラーが最初に発生した内部メソッド。

状態　IMS TM リソース・アダプターの内部状態。

IMS TM リソース・アダプター J2C アプリケーションによって生成される例外

以下の例外メッセージは、エラー状態が検出されたときに、Java EE コネクター・アーキテクチャーのクラス・ライブラリーでビルドされたアプリケーションが生成するものです。

ICO0001E **javax.resource.spi.EISSystemException: ICO0001E: *methodname* error. IMS Connect returned an error: RETCODE=[*returncode*], REASONCODE=[*reasoncode*]. *reasoncode_string*.**

説明: IMS Connect がエラーを返しました。エラー状態の接続は再使用されません。*reasoncode_string* は、*reasoncode* (ある場合) の要旨を示します。

IMS TM Resource Adapter バージョン 12 以降が IMS バージョン 12 以降とともに使用されている場合、RACF 戻りコードと障害の説明が返されます。リソース・アダプターまたは IMS のバージョンがこれより古い場合、すべての RACF 認証エラーに対して単一の戻りコードと理由コードが返されます。

ユーザーの処置: z/OS コンソールを調べて、関連する IMS Connect のエラー・メッセージがないか確認してください。IMS Connect のエラー・メッセージは先頭に接頭部 HWS が付いています。戻りコード (*returncode*) および理由コード (*reasoncode*) の値に関する診断情報、および IMS Connect のエラー・メッセージについては、「*IMS* メッセージおよびコード」の統合された IMS Connect の戻りコードと理由コードのセクションを参照してください。

関連資料:

統合された IMS Connect 戻りコードと理由コード (IMS バージョン 15)

ICO0002E **javax.resource.spi.EISSystemException: ICO0002E:*methodname* error. IMS OTMA returned an error: SENSECODE=[*sensecode*], REASONCODE=[*otmareasoncode*]. [*source_methodname*:*source_exception*]**

説明: IMS OTMA が否定応答 (NAK) エラーを返しました。

ユーザーの処置: IMS TM リソース・アダプターは、*sensecode* と *otmareasoncode* を 10 進数で表示します。理由コード 0 は、センス・コードに関連付けられた理由コードがないことを示します。アプリケーションが 2 フェーズ・コミットで実行されている場合には、NAK エラーとともに以下のセンス・コード値を受信することがあります。

表 121. アプリケーションが 2 フェーズ・コミットで実行されている場合のセンス・コード値

センス・コード	説明
17 (10 進数。16 進数では 11)	ご使用の IMS ではリソース・リカバリー・サービス (RRS) 処理を行うことができません。ご使用の IMS で、RRS が有効な保護会話処理が行われていることを確認してください。詳しくは、グローバル・トランザクションおよび 2 フェーズ・コミットのサポートに関するトピックを参照してください。
46 (10 進数。16 進数では 2E)	RRS および 2 フェーズ・コミット処理は、IMS Connect および IMS TM リソース・アダプターではサポートされません。
51 (10 進数。16 進数では 33)	非同期出力メッセージまたはコールアウト要求が保留キューからリトリーブされるときに許可の失敗が発生した場合、OTMA 理由コードは以下のいずれかの値になります。 • 1 – セキュリティー・セグメントが必要です。 • 2 – ユーザー ID が必要です。 • 3 – RACF グループ名が必要です。 • 4 – UTOKEN が必要です。 • 5 – TPIPE 名が必要です。 • 6 – RACF でシステム・エラーが発生しました。 • 7 – USERID 用の RACF プロファイルが RACF データベースにありません。 • 8 – 許可ユーザー ID が必要です。 • 9 – 有効な TPIPE 名が必要です。

NAK エラーのセンス・コード (*sensecode*) および OTMA 理由コード (*otmareasoncode*) の値に関する診断情報については、「*IMS* メッセージおよびコード 」を参照してください。

関連概念:

1009 ページの『グローバル・トランザクションと 2 フェーズ・コミット・サポート処理』

関連資料:

OTMA センス・コードと戻りコード (IMS バージョン 15)

ICO0003E **javax.resource.spi.CommException: ICO0003E:***methodname* **error. Failed to connect to host [***hostname***], port [***portnumber***]. [***java_exception***]**

説明: IMS TM リソース・アダプターがホストとポートの組み合わせに接続できませんでした。
java_exception は接続に失敗した理由を示しています。

ユーザーの処置: *java_exception* を調べて、ホストへの接続に失敗した理由を判別してください。
java_exception の値には以下のようなものがあります。

表 122. *ICO0003E* の例外

例外	説明
java.net .UnknownHost Exception: *hostname*	アプリケーションで使用されている接続ファクトリーの構成時に指定したホスト名が無効であるか、アプリケーションが無効なホスト名を指定しました。ホスト名のスペルを確認してください。ホスト名または IP アドレスに完全修飾パスの使用が必要であると考えられます。

表 122. ICO0003E の例外 (続き)

例外	説明
java.net .ConnectException: Connection refused	以下に、この例外の理由として考えられるものをいくつか示します。 • ポート番号が無効である。*hostname* によって示されている IMS Connect に有効なポート番号を使用していることを確認してください。 • 指定されたポートが停止している。IMS Connect コマンド VIEWHWS を使用してポートの状況を確認してください。ポートが停止している場合、その状況は NOT ACTIVE です。ポートを開始するには、IMS Connect コマンド OPENPORT *dddd* を使用します。*dddd* は指定されたポート番号です。 • 指定されたホスト上で IMS Connect が実行していない。ホスト・システムで IMS Connect を開始してください。 • IMS Connect の取り消しと再始動、ホストでの STOPPORT に続く OPENPORT の発行がいずれも行われずに TCP/IP が再始動された。

表 122. ICO0003E の例外 (続き)

例外	説明
java.net .SocketException: connect (code=10051)	以下に、この例外の理由として考えられるものをいくつか示します。 • 指定されたホスト名を持つシステムが、インターネット・プロトコル・ネットワークで到達不能である。指定されたホスト・システムに ping コマンドを発行して、そのホスト・システムがインターネット・プロトコル・ネットワークからアクセス可能であることを確認してください。 ping コマンドは、IMS TM リソース・アダプターが実行されているシステムで入力します。ホストで TCP/IP を開始します (まだ開始されていない場合)。 • TCP/IP が再始動されたが、アプリケーションによって使用されているポートの状況が NOT ACTIVE だった。この状況を修正するには、以下のいずれかのアクションを実行します。 – IMS Connect コマンド OPENPORT *dddd* (*dddd* はポート番号) を使用してポートを活動化します。 – IMS Connect を再始動します。

ICO0005E **javax.resource.spi.CommException: ICO0005E:*methodname* error. A communication error occurred during sending or receiving the IMS message. clientID=[*clientid*][*java_exception*]**

説明: IMS TM リソース・アダプターが、ターゲット IMS Connect との間の送受信の対話を正常に完了できませんでした。メッセージに示されている *clientid* は、通信例外が発生した接続のクライアント ID です。*java_exception* は対話の完了に失敗した理由を示しています。

ユーザーの処置: クライアント ID を利用して、関係するさまざまなコンポーネントからのトレース・データを分析してください。 *java_exception* を調べて、失敗の

理由を判別します。次の表には、*java_exception* の値をいくつか示しています。

表 *123. ICO0005E* の *Java* 例外

Java 例外	説明
java.io .EOFException	以下に、この例外の理由として考えられるものをいくつか示します。 • IMS Connect が IMS から応答を受信する前に、IMS Connect 構成メンバーに指定されているタイムアウト値を超えた。通常、タイムアウト値の超過は、クライアント要求を処理する IMS トランザクションを実行するための領域が IMS 内で使用可能でないときに発生します。その場合は、該当する領域が開始済みであり、要求の処理に使用できることを確認してください。トランザクションに関連付けられている IMS アプリケーション・プログラムが停止した場合にも、タイムアウト値の超過が発生することがあります。その場合は、IMS コマンド /START PROGRAM を使用して IMS アプリケーション・プログラムを開始してください。 • Java クライアントが、前もってアクティブになっているクライアント (例えば、プールからの接続) を使用しようとしたが、そのクライアントに対して IMS Connect コマンド STOPCLNT が発行されている。
java.net .SocketException: Connection reset by peer: socket write error	以下に、この例外の理由として考えられるものをいくつか示します。 • Java クライアントが接続を使用しようとしているが、その接続の基礎にあるソケットが IMS Connect に接続されなくなっている。IMS Connect がリサイクルされていても、アプリケーション・サーバーがリサイクルされていない場合は、ソケット接続が失われる可能性があります。 IMS Connect の再始動後も、前に正常に IMS Connect に接続されていた接続は接続プール内に残っています。クライアントがこうした接続のいずれかを再使用しようとすると、例外 `java.net.SocketException` がスローされ、その接続オブジェクトは接続プールから削除されます。 この動作を変更するには、WebSphere Application Server で、Java アプリケーションによって使用されている接続ファクトリーのパージ・ポリシーをプール全体に設定します。 • ホスト上の TCP/IP がダウンしている。

ICO0006E **javax.resource.ResourceException: ICO0006E:*methodname* error. The value for *DataStoreName* is null or an empty string.**

説明: *methodname* に示されているメソッドが、空の *DataStoreName* パラメーターを使用して呼び出されました。このエラー・メッセージは、空の *DataStoreName* パラメーターが指定された接続ファクトリーが開始されるとトレース・ログに表示されます。このメッセージの後に、次の Java EE Connector の警告が出されます。

```
J2CA0007W: An exception occurred while invoking
method setDataStoreName on
com.ibm.connector2.ims.ico
.IMSManagedConnectionFactory used by resource
Connection_Factory_JNDI_name.
```

この後、処理が続行されますが、ヌルの名前を持つデータ・ストアが見つからないことを示す応答を IMS

Connect が送信した後に、他のエラー・メッセージが出されることになります。他のメッセージをトリガーする、基礎になっているメッセージは次のとおりです。

```
javax.resource.spi.EISSystemException: ICO0001E:
com.ibm.connector2.ims.ico
.IMSTCPIPManagedConnection@.processOutputOTMAMsg
(byte[], InteractionSpec, Record) error.
IMS Connect returned error: RETCODE=[4],
REASONCODE=[NFNDDST ].
Datastore not found.
```

このエラーが発生すると、IMS Connect が実行されているホスト・システムの z/OS コンソールに対応する HWSS0742W 警告メッセージが表示されます。この HWSS0742W メッセージには、検出が試みられたデータ・ストア名を示すフィールド (この場合はすべてブランク) が含まれます。

```
DESTID=        ;
```

ユーザーの処置: *DataStoreName* パラメーターに有効な名前を指定してください。管理対象環境では、WebSphere Application Server によって使用される接続ファクトリーを構成するときに *DataStoreName* が指定されます。非管理対象環境では、*DataStoreName* は Java アプリケーションで指定されます。

ICO0007E **javax.resource.NotSupportedException: ICO0007E:*methodname* error. The [*propertyname*] property value [*propertyvalue*] is not supported.**

説明: プロパティー *propertyname* に指定された値 *propertyvalue* はサポートされていません。

ユーザーの処置: 示されたプロパティーにサポートされる値を指定してください。例えば、J2C アーキテクチャーで定義されている InteractionSpec クラスの interactionVerb プロパティーの一部の値は、IMS TM リソース・アダプターの IMSInteractionSpec ではサポートされません。また、専用永続ソケット接続では、reRoute プロパティーの `true` 値はサポートされません。

ICO0008E **javax.resource.ResourceException: ICO0008E:*methodname* error. The value [*propertyvalue*] of the [*propertyname*] property exceeds the maximum allowable length of [*maxpropertylength*].**

説明: プロパティー *propertyname* に指定された値 *propertyvalue* の長さが *maxpropertylength* (プロパティー *propertyname* の値の許容最大長) を超えています。

ユーザーの処置: 示されているプロパティーに、*maxpropertylength* を超えない値を指定してください。

ICO0009E **javax.resource.ResourceException: ICO0009E:*methodname* error. The [*propertyname*] property value [*propertyvalue*] is not valid.**

説明: プロパティー *propertyname* に指定された値 *propertyvalue* は無効です。

ユーザーの処置: 示されているプロパティーに有効な値を指定してください。例えば、IMS TM リソース・アダプターの InteractionSpec クラスの interactionVerb プロパティーで有効な値は、IMSInteractionSpec クラスの Javadoc の資料にリストされています。

ICO0010E **javax.resource.spi.IllegalStateException: ICO0010E:*methodname* error. The method was invoked on an invalid IMSConnection instance.**

説明: *methodname* に示されているメソッドが、無効な IMSConnection インスタンスで呼び出されました。*methodname* が `lazyEnlist` である場合、現行トランザクション内で、参加できなかった接続の参加が試みられました。

ユーザーの処置: 示されているメソッドが、既に閉じられている IMSConnection インスタンスで発行された可能性があります。

- *methodname* が `lazyEnlist` でない場合は、IMSConnection インスタンスを使用または閉じようとする前に、そのインスタンスがまだ閉じられていないことを確認してください。
- *methodname* が `lazyEnlist` である場合は、管理対象環境内でアプリケーションが非管理対象接続を使用していないことを確認してください。トランザクションの参加の遅延最適化の対象として適格なのは、管理対象接続のみであるためです。詳しくは、WebSphere Application Server のインフォメーション・センターを参照してください。

関連情報:

トランザクションの参加の遅延最適化
(WebSphere Application Server V8 のインフォメーション・センター)

ICO0011E **javax.resource.spi.IllegalStateException: ICO0011E:*methodname* error. The method was invoked on an invalid IMSInteraction instance.**

説明: *methodname* に示されているメソッドが、無効な

IMSInteraction インスタンスで呼び出されました。

ユーザーの処置: 示されているメソッドが、既に閉じられている IMSInteraction インスタンスで発行された可能性があります。 IMSInteraction インスタンスを使用または閉じようとする前に、そのインスタンスがまだ閉じられていないことを確認してください。

ICO0012E **javax.resource.ResourceException: ICO0012E:*methodname* error. The value provided for HostName is null or an empty string.**

説明: *methodname* に示されているメソッドが、ヌルまたは空の **HostName** パラメーターを使用して呼び出されました。

ユーザーの処置: **HostName** パラメーターに有効な値を指定してください。管理対象環境では、WebSphere Application Server によって使用される接続ファクトリーの構成時にこのプロパティー値が指定されます。非管理対象環境では、Java アプリケーションでこのプロパティー値が指定されます。

ICO0013E **javax.resource.ResourceException: ICO0013E:*methodname* error. The ConnectionManager is null.**

説明: アプリケーション・サーバーが、ヌルの ConnectionManager オブジェクトを使用して IMSManagedConnectionFactory クラスの createConnectionFactory メソッドを呼び出しました。

ユーザーの処置: この形式の createConnectionFactory メソッドは、通常、クライアント・プログラムによって呼び出されるのではなく、管理環境で使用されます。ご使用のアプリケーション・サーバーのサービス担当員にお問い合わせください。

ICO0014E **javax.resource.ResourceException: ICO0014E:*methodname* error. The input record contains no data.**

説明: *methodname* に示されているメソッドが、データを含んでいない入力レコードを使用して呼び出されました。

ユーザーの処置: 指定した入力レコードが空でないことを確認してください。

ICO0015E **ResourceAdapterInternalException ICO0015E: *methodname* error. An unexpected error occurred while the OTMA message was being processed. [*java_exception*]**

説明: OTMA メッセージの処理中に予期しない内部エラーが発生しました。ローカル Java クライアントがコード・ページ 1047 と 037 をいずれも検出できませんでした。

ユーザーの処置: ローカル Java クライアントにコード・ページ 1047 と 037 がインストール済みであること、およびそれらが破損していないことを確認してください。

ICO0016E **javax.resource.ResourceException: ICO0016E:*methodname* error. The message was encoded using an unsupported code page. [*java_exception*].**

説明: 指定されたコード・ページを使用してメッセージをエンコードできないか、コード・ページが見つかりませんでした。

ユーザーの処置: ローカル Java クライアントにコード・ページ 1047 と 037 がインストール済みであること、およびそれらが破損していないことを確認してください。

ICO0017E **ResourceAdapterInternalException ICO0017E:*methodname* error. Invalid value provided for TraceLevel.**

説明: 無効なトレース・レベルが指定されました。

ユーザーの処置: 有効なトレース・レベルを指定してください。この接続ファクトリーにはデフォルトのトレース・レベルが使用されるため、オプションで、この例外を無視することもできます。この場合、接続ファクトリーは引き続き使用可能ですが、トレース・レベルはデフォルトのトレース・レベルになります。

ICO0018E **javax.resource.ResourceException: ICO0018E:*methodname* error. The value provided for PortNumber is null.**

説明: *methodname* で示されているメソッドが、ヌルの PortNumber を使用して呼び出されました。

ユーザーの処置: 有効な PortNumber パラメーターを指定してください。管理対象環境では、WebSphere Application Server によって使用される接続ファクトリーの構成時にこのプロパティー値が指定されます。非管理対象環境では、Java アプリケーションでこのプロパティー値が指定されます。

ICO0020E **javax.resource.ResourceException: ICO0020E:*methodname* error. Alternate client ID is not supported for interaction [*interactionverb*].**

説明: 代替クライアント ID を使用して指定された *interactionverb* の値が無効です。

ユーザーの処置: 有効な *interactionverb* 値を指定してください。代替クライアント ID を指定できるのは、SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT、SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT、および SYNC_RECEIVE_CALLOUT の場合のみです。

ICO0024E **javax.resource.ResourceException: ICO0024E:*methodname* error. Invalid segment length (LL) of [*llvalue*] in the input object. [*java_exception*]**

説明: IMS アプリケーション・プログラム用の Java プログラムによって渡された入力メッセージに含まれるセグメント長の値 (*llvalue*) が、負であるか、0 であるか、またはメッセージ・セグメント内のデータのバイト数より大きい値です。

ユーザーの処置: 入力メッセージのセグメント長に正しい値を指定してください。

ICO0025E **javax.resource .IllegalArgumentException: ICO0025E:*methodname* error. Invalid segment length (LL) of [*llvalue*] in the OTMA message.**

説明: IMS アプリケーション・プログラムによって提供される出力メッセージに、負、0、またはメッセージ・セグメント内のデータのバイト数より大きなセグメント長の値 (*llvalue*) が含まれています。IMS アプリケーション・プログラムによって提供される出力メッセージは、OTMA メッセージに含まれます。

ユーザーの処置: IMS アプリケーション・プログラムが、出力メッセージのセグメントに対して有効な長さを指定していることを確認してください。

ICO0026E **javax.resource.ResourceException: ICO0026E:*methodname* error. An error was encountered while processing the IMS message. [*source_methodname*:*source_exception*]**

説明: IMS トランザクション・メッセージの処理中にエラーが発生しました。 *source_exception* は、エラーの原因に関連する追加情報を提供しています。

ユーザーの処置: エラーの原因に関する追加情報については、*source_exception* を調べてください。*source_exception* の値に基づいて推奨される処置については、次の表で説明しています。

表 *124. ICO0026E* のソース例外

ソース例外	説明
java.io .IOException	入力レコードまたは出力レコードの準備中にエラーが発生しました。IMS トランザクションの入出力として使用するために IMS TM リソース・アダプターに提供するオブジェクトが、J2C アーキテクチャーに準拠して正しく定義されていることを確認してください。例えば、インターフェース javax.resource.cci.Record と javax.resource.cci.Streamable を実装する必要があります。
com.ibm.ims.ico .IMSConnResource Exception	IMS トランザクション出力メッセージを含む OTMA メッセージに、無効な長さフィールドが含まれていました (例えば、LLLL が 0 以下だった)。IMS アプリケーション・プログラムが有効な出力メッセージを返していることを確認した後にもこのエラーが引き続き発生する場合は、IBM サービス担当員に連絡してください。
java.lang .IllegalArgument Exception	IMS Connect から返された出力メッセージが無効です。 IMS TM リソース・アダプターと IMS Connect のリリース・レベルに互換性があることを確認してください。例えば、新しいバージョンのリソース・アダプターを使用して、TCP/IP 経由で 2 フェーズ・コミット・トランザクションを実行するトランザクション必須 EJB アプリケーションを作成した場合に、実行時に IMS TM Resource Adapter のサービス休止のバージョンを使用すると、この例外が発生します。この問題を解決するには、実行時環境で同じバージョンの IMS TM Resource Adapter に更新します。

ICO0027E **javax.resource.ResourceException: ICO0027E:*methodname* error. The OTMA header of the IMS output message did not contain a *segment_name* segment.**

説明: IMS トランザクション出力メッセージの OTMA ヘッダーが正しく作成されませんでした。 *segment_name* によって示されているセグメントが含まれていませんでした。

ユーザーの処置: メッセージ・ヘッダーの作成方法についての問題を修正してください。このエラーが発生する可能性がある状態の例は、IMS TM リソース・アダプター以外のクライアントによってキューに入れられたメッセージをリトリーブする SYNC_RETRIEVE_ASYNCHOUTPUT_WAIT または SYNC_RETRIEVE_ASYNCHOUTPUT_NOWAIT の対話 (例えば、3270 端末の対話) を実行する場合です。IMS TM リソース・アダプター以外のクライアントによってキューに入れられるメッセージには通常、IMS TM リソース・アダプターが必要とする OTMA ヘッダーが含まれていません。この問題を解決するには、IMS TM リソース・アダプターのクライアント、および IMS TM リソース・アダプター以外のクライアントが、それぞれの非同期出力に対して別個のキューを使用するようにしてください。

ICO0028E **javax.resource.ResourceException: ICO0028E:*methodname* error. The Prefix flag in the OTMA header Message-Control Information segment of the IMS output message is not valid.**

説明: IMS トランザクション出力メッセージの OTMA ヘッダーが正しく作成されませんでした。 OTMA ヘッダー内の接頭部フラグが、メッセージの OTMA ヘッダーに含まれるセグメントを指定していましたが、メッセージの合計長 (ヘッダーとメッセージ・データの両方を含む必要がある) が、指定されたヘッダーの長さの合計を下回っています。合計メッセージ長は、ヘッダーの合計長と等しいかそれ以上であることが必要ですが (メッセージがデータを含むかどうかによって異なる)、ヘッダーの合計長を下回ることはありません。

ユーザーの処置: メッセージ・ヘッダーの作成方法についての問題を修正してください。通常、この状態になる原因は、ユーザー・メッセージがメッセージ・ヘッダーを誤って変更したか、IMS Connect または IMS TM リソース・アダプターのいずれかに内部エラーが発生したことです。ユーザー出口が変更されずに出荷時の状態で使用されている場合、またはヘッダーが正しく作成されない理由が判別できない場合は、IBM ソフトウェア・サポートに連絡してください。

ICO0030E **javax.resource.spi.ApplicationServerInternalException: ICO0030E:*methodname* error. [*source_methodname*:*source_exception*]**

説明: 対話中に実行時エラーまたは例外が *methodname* 内で検出されました。*source_methodname:source_exception* は、*methodname* 内で検出されたエラーまたは例外が最初に発生した場所を示し、エラーの原因に関する追加情報を提供する場合もあります。

ユーザーの処置: エラーの原因に関する追加情報については、*source_exception* を調べてください。実行する処置は、*source_methodname*:*source_exception* の値によって決まります。*source_methodname:source_exception* の値に基づいて推奨される処置については、次の表で説明しています。

表 *125. ICO0030E* のソース例外

Java 例外	説明
java.lang .OutOfMemory Error	このエラーは、メモリー不足のために Java Virtual Machine がオブジェクトを割り振ることができず、ガーベッジ・コレクターによって追加のメモリーを取得できない場合にスローされます。WebSphere Application Server によって使用される仮想マシンが使用可能なメモリーの量を増やします。
java.io .InterruptedIO Exception	転送を実行しているスレッドが終了したために、入力または出力の転送が終了したことを示す InterruptedIOException がスローされました。スレッドが終了した理由を調べてください。

ICO0031E **javax.resource.spi.IllegalStateException: ICO0031E:*methodname* error. Protocol violation. The interaction verb [*interactionverb*] is not allowed for the current state [*state*]. [*java_exception*]**

説明: アプリケーションが対話を実行しようとしましたが、プロトコル違反が発生しました。[*interactionverb*] は、対話に使用された IMSInteractionSpec オブジェクトの interactionVerb プロパティーの値です。[*state*] は、IMS TM リソース・アダプターと IMS Connect

の間の対話に使用されているプロトコルの現在の状態です。

ユーザーの処置: IMSInteractionSpec クラスの interactionVerb プロパティーに適切な値を使用していることを確認してください。

ICO0034E javax.resource.NotSupportedException: ICO0034E:*methodname* error. Auto-commit is not supported.

説明: 自動コミットは、IMS TM リソース・アダプターではサポートされていません。

ユーザーの処置: お使いの Java アプリケーションが、IMS TM リソース・アダプターで提供されるサポートのレベルに適合するクラスとメソッドを使用していることを確認してください。

ICO0035E javax.resource.NotSupportedException: ICO0035E:*methodname* error. Local transaction is not supported.

説明: ローカル・トランザクションは、IMS TM リソース・アダプターではサポートされません。

ユーザーの処置: お使いの Java アプリケーションが、IMS TM リソース・アダプターで提供されるサポートのレベルに適合するクラスとメソッドを使用していることを確認してください。

ICO0037E javax.resource.NotSupportedException: ICO0037E:*methodname* error. ResultSet is not supported.

説明: ResultSets は、IMS TM リソース・アダプターではサポートされません。

ユーザーの処置: お使いの Java アプリケーションが、IMS TM リソース・アダプターの提供するサポートのレベルに適合するクラスとメソッドを使用していることを確認してください。

ICO0039E javax.resource.spi.IllegalStateException: ICO0039E:*methodname* error. Not in CONNECT state.

説明: IMS TM リソース・アダプターと IMS Connect の間の対話のシーケンスが無効です。IMS TM リソース・アダプターと IMS Connect の間の対話に使用されるプロトコルの現在の状態が、対話のこの時点で必要な CONNECT ではありません。

ユーザーの処置: IMS TM リソース・アダプターまたは IMS Connect でエラーが発生している可能性があります。IBM ソフトウェア・サポートに連絡してください。

ICO0040E javax.resource.NotSupportedException: ICO0040E:*methodname* error. IMSConnector does not support this version of execute method.

説明: IMS TM リソース・アダプターは、2 つの入力パラメーターを取得してタイプ javax.resource.cci.Record のオブジェクトを返す形式の execute メソッドをサポートしません。

ユーザーの処置: クラス IMSInteraction のサポートされる形式の execute メソッドを使用してください。サポートされる形式の execute メソッドのシグニチャーは、以下のとおりです。

```
boolean execute(InteractionSpec, Record input,
        Record output)
```

ICO0041E javax.resource.ResourceException: ICO0041E:*methodname* error. An invalid interactionSpec [*interactionSpec*] was specified.

説明: クラス com.ibm.connector2.ims.ico.IMSInteraction の execute メソッドに、無効な InteractionSpec オブジェクトが渡されました。

ユーザーの処置: クラス com.ibm.connector2.ims.ico.IMSInteraction の execute メソッドに渡す InteractionSpec オブジェクトが、タイプ com.ibm.connector2.ims.ico.IMSInteractionSpec のものであることを確認します。

ICO0042E javax.resource.ResourceException: ICO0042E: *methodname* error. The input is not of type Streamable.

説明: com.ibm.connector2.ims.ico.IMSInteraction の execute メソッドの入力パラメーターに指定された入力がヌルであったか、インターフェース javax.resource.cci.Streamable を実装していませんでした。この例外が発生する可能性が最も高いのは、アプリケーションが Java EE コネクター・アーキテクチャーの Common Client Interface (CCI) を使用するように作成されている場合です。この例外は、入力メッセージの作成に Rational または WebSphere 統合開発環境が使用されている場合には発生しません。

対話のタイプによっては、execute メソッドはヌルの入力オブジェクトを許可します。例えば、対話の interactionVerb の値が SYNC_END_CONVERSATION、

SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT、および SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT の場合は、ヌルの入力オブジェクトを使用できます。

ユーザーの処置: execute メソッドの入力パラメーターに有効な javax.resource.cci.Record オブジェクトを指定したことを確認してください。例えば、このオブジェクトがインターフェース javax.resource.cci.Record と javax.resource.cci.Streamable を実装していることを確認します。

ICO0043E **javax.resource.ResourceException: ICO0043E:** *methodname* **error. The output is not of type Streamable.**

説明: com.ibm.connector2.ims.ico.IMSInteraction の execute メソッドに指定された出力オブジェクトがヌルであったか、インターフェース javax.resource.cci.Streamable を実装していませんでした。この例外が発生する可能性が最も高いのは、アプリケーションが Java EE コネクター・アーキテクチャーの Common Client Interface (CCI) を使用するように作成されている場合です。この例外は、出力メッセージの作成に Rational または WebSphere 統合開発環境が使用されている場合には発生しません。

ユーザーの処置: execute メソッドに有効な出力オブジェクトを指定したことを確認してください。

ICO0044E **javax.resource.NotSupportedException: ICO0044E:***methodname* **error. RecordFactory is not supported by IMS TM Resource Adapter.**

説明: RecordFactory は、IMS TM リソース・アダプターではサポートされていません。

ユーザーの処置: お使いの Java アプリケーションが、IMS TM リソース・アダプターの提供するサポートのレベルに適合するクラスとメソッドを使用していることを確認してください。

ICO0045E **javax.resource.NotSupportedException: ICO0045E:***methodname* **error. Invalid type of ConnectionRequestInfo.**

説明: IMS TM リソース・アダプターのメソッドに、無効な ConnectionRequestInfo オブジェクトが渡されました。

ユーザーの処置: IMS TM リソース・アダプターでエラーが発生している可能性があります。IBM ソフトウェア・サポートに連絡してください。

ICO0049E **javax.resource.NotSupportedException: ICO0049E:***methodname* **error. The security credentials passed to getConnection do not match existing security credentials.**

説明: 要求のセキュリティー資格情報が、要求の処理に使用されている IMSManagedConnection インスタンスのセキュリティー資格情報と一致しません。

ユーザーの処置: IBM ソフトウェア・サポートに連絡してください。

ICO0050E **ICO0050E:** *methodname* **error. Invalid RACF user id is specified in SSLKeyStoreName or SSLTrustStoreName when specifying a RACF keystore or truststore.**

説明: RACF 鍵ストアまたはトラストストアの SSLKeyStoreName プロパティーまたは SSLTrustStoreName プロパティーに指定されたユーザー ID が無効です。

ユーザーの処置: 有効なユーザー ID を指定し、RACF ユーザー ID の長さが 8 文字未満であることを確認してください。

ICO0053E **javax.resource.ResourceException: ICO0053E:** *methodname* **error. The client ID is not value. The prefix HWS is reserved by the IMS TM resource adapter.**

説明: clientID プロパティーに指定された値は無効です。接頭部 HWS は、IMS TM リソース・アダプターによって予約されています。

ユーザーの処置: clientID プロパティーに有効な値を指定します。有効な値は、次の規則に従っているものです。

- ヌル・ストリングではない。
- 先頭がブランク・フィールドになっていない。
- IMS TM リソース・アダプターの予約接頭部 HWS で始まっていない。
- 長さが 8 文字である。
- 有効な文字 A から Z、0 から 9、@、#、および $ が使用されている。

ICO0054E **javax.resource.ResourceException: ICO0054E:*methodname* error. Invalid ConnectionSpec.**

説明: IMS TM リソース・アダプターは、この接続に対して提供された connectionSpec をタイプ IMSConnectionSpec にキャストできませんでした。Common Client Interface は、サポートされるすべてのコネクターについて connectionSpec オブジェクトを受け入れますが、IMS TM リソース・アダプターは、IMSConnectionSpec または IMSConnectionSpec からの派生物のみを connectionSpec として処理します。

ユーザーの処置: アプリケーションが使用する connectionSpec が、IMSConnectionSpec または IMSConnectionSpec から継承されたものであることを確認してください。

ICO0055E **javax.resource.ResourceException: ICO0055E:*methodname* error. Failed to cast the connection object to IMSConnection.**

説明: IMS TM リソース・アダプターは、この接続に対して ConnectionManager によって割り振られた接続オブジェクトを、タイプ IMSConnection にキャストできませんでした。IMS TM リソース・アダプターは、IMSConnection または IMSConnection からの派生物のみを接続オブジェクトとして処理します。このエラーは、ConnectionManager インスタンスの問題によって生じている可能性があります。

ユーザーの処置: IBM ソフトウェア・サポートに連絡してください。

ICO0056E **javax.resource.ResourceException: ICO0056E:*methodname* error. IMSConnectName is only valid for Local Option connections which can only be used in z/OS or OS/390.**

説明: ローカル・オプションの接続に使用される管理接続ファクトリーには、IMSManagedConnectionFactory インスタンスの IMSConnectName プロパティーの設定が必要です。また、IMS TM リソース・アダプターを使用するアプリケーションが z/OS 上で実行されている場合は、IMS Connect との通信にローカル・オプションのみを使用できます。この例外は、IMSConnectName プロパティーに値を指定した一方で、アプリケーションが z/OS 上で実行されていないことを示しています。

ユーザーの処置: IMS TM リソース・アダプターを使用するアプリケーションが、z/OS で実行されていることを確認してください。アプリケーションを実行する Web アプリケーション・サーバーは、IMS Connect と同じ z/OS イメージ内で実行されている必要があります。例えば、ワークステーション・プラットフォームでアプリケーションを実行する場合、またはアプリケーションを実行する Web サーバーが z/OS 上にあり、ただし IMS Connect とは異なる z/OS イメージ内にある場合は、アプリケーションによって使用される接続ファクトリーが TCP/IP 通信を使用するようにセットアップされていることを確認します。

ICO0057E **javax.resource.spi.IllegalStateException: ICO0057E:*methodname* error. Invoked with invalid connection handle.**

説明: アプリケーションが正しくない状態です。この対話に使用される接続ハンドル (IMSConnection インスタンス) が無効です。以前に使用された接続の接続ハンドル、またアプリケーションが複数の接続を開いている場合は正しくない接続の接続ハンドルを、アプリケーションが使用しようとしている可能性があります。

ユーザーの処置: アプリケーションが、その接続の有効な IMSConnection インスタンスを使用していることを確認してください。

ICO0058E **javax.resource.ResourceException: ICO0058E:*methodname* error. Interactions SYNC_SEND_RECEIVE, SYNC_SEND, SYNC_RECEIVE_ASYNCOUTPUT, SYNC_RECEIVE_ASYNCOUTPUT _SINGLE_NOWAIT and SYNC_RECEIVE_ASYNCOUTPUT _SINGLE_WAIT interactions with Commit Mode 0 are not supported with Local Option.**

説明:

ユーザーの処置: アプリケーションに対してコミット・モード 1 が選択されていることを確認してください。コミット・モード 0 でアプリケーションを実行する場合は、TCP/IP 通信を使用するようにアプリケーションを修正してください。

ICO0059E **javax.resource.ResourceException: ICO0059E: *methodname* error. SYNC_END_CONVERSATION interaction with Commit Mode 0 is not supported.**

説明: コミット・モード 0 の対話 SYNC_END_CONVERSATION はサポートされていません。

ユーザーの処置: IMS TM リソース・アダプターは、コミット・モード 1 の SYNC_END_CONVERSATION、コミット・モード 0 の SYNC_SEND_RECEIVE、およびコミット・モード 0 の SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT の対話の組み合わせをサポートします。

ICO0060E **java.lang.UnsatisfiedLinkError: ICO0060E:***methodname* **error. Error loading Local Option native library: libname=***libraryfilename***. [***source_exception***].**

説明: *libraryfilename* という名前のローカル・オプションのネイティブ・ライブラリーが、LIBPATH 環境変数にリストされたどのディレクトリーにも見つかりません。

ユーザーの処置: LIBPATH 環境変数に含まれるディレクトリーのいずれかに、ローカル・オプションのネイティブ・ライブラリーが存在することを確認してください。IMS TM リソース・アダプターが WebSphere Application Server for z/OS 上で実行されている場合は、ローカル・オプションのネイティブ・ライブラリー・ファイルが格納されているディレクトリーのフルネームが、Java EE サーバーの LIBPATH 環境変数に定義されていることを確認してください。詳しくは、WebSphere Application Server インフォメーション・センターで、基本オペレーション・システムを準備する方法に関するトピックを参照してください。

関連情報:

基本 z/OS オペレーティング・システムの準備 (WebSphere Application Server V8 のインフォメーション・センター)

ICO0061E **javax.resource.ResourceException: ICO0061E:***methodname* **error. Local Option runs only in z/OS and OS/390.**

説明: IMS TM リソース・アダプターを使用しているアプリケーションが z/OS 上で実行されている場合にのみ、ローカル・オプションを使用して IMS Connect と通信できます。

ユーザーの処置: IMS TM リソース・アダプターを使用するアプリケーションが、z/OS で実行されていることを確認してください。アプリケーションを実行する Web アプリケーション・サーバーは、IMS Connect と同じ z/OS イメージ内で実行されている必要があります。ワークステーション・プラットフォームでアプリケーションを実行する場合、またはアプリケーションを実行する Web サーバーが z/OS 上にあり、ただし IMS Connect とは異なる z/OS イメージ内にある場合は、アプリケーションによって使用される接続ファクトリーが TCP/IP 通信を使用するようにセットアップされていることを確認します。

ICO0062E **javax.resource.ResourceException: ICO0062E:***methodname* **error. Error loading Local Option native method: libfilename=***libraryFileName***, methodname=***nativeMethodName***. [***source_exception***].**

説明: *nativemethodname* という名前のローカル・オプションのネイティブ・メソッドが、*libraryfilename* という名前のローカル・オプションのネイティブ・ライブラリー・ファイルに見つかりません。

ユーザーの処置: 正しいレベルの IMS TM リソース・アダプターとローカル・オプションのネイティブ・ライブラリーが、システムにインストールされていることを確認してください。ローカル・オプションのネイティブ・ライブラリーは、WebSphere Application Server for z/OS にインストールした IMS TM リソース・アダプターに付属するバージョンのものを必ず使用してください。

ICO0063E **javax.resource.spi .ResourceAdapterInternalException: ICO0063E:***methodname* **error. Exception thrown in native method. [***source_exception***].**

説明: ローカル・オプションのネイティブ・メソッドで内部エラーが発生しました。

ユーザーの処置: IBM ソフトウェア・サポートに連絡してください。

ICO0064E **javax.resource.spi.SecurityException: ICO0064E:***methodname* **error. Invalid security credential.**

説明: WebSphere Application Server は、IMS TM リソース・アダプターによってサポートされるセキュリティー資格情報を提供しませんでした。

ユーザーの処置: 正しいレベルの WebSphere Application Server for z/OS がインストールされていることを確認してください。IMS TM リソース・アダプターによってサポートされるセキュリティー資格情報

(TCP/IP 接続の場合は PasswordCredential、ローカル・オプション接続の場合は UToken GenericCredential) を提供するように WebSphere Application Server for z/OS を構成します。

ICO0065E **javax.resource.spi.SecurityException: ICO0065E:***methodname* **error. Error obtaining credential data from the security credential. [***source_exception***].**

説明: アプリケーション・サーバーによって提供されたセキュリティー資格情報から資格情報データを取得する際に、セキュリティー関連エラーが発生しました。

ユーザーの処置: 呼び出し側プログラムに関連したユーザーがセキュリティー資格情報からデータを抽出することを許可されるように、アプリケーション・サーバーのセキュリティーを正しくセットアップしていることを確認してください。

ICO0066E **javax.resource.ResourceException: ICO0066E:***methodname* **error. Error loading WebSphere Application Server Transaction Manager. [***source_exception***].**

説明: トランザクション要求を処理するために WebSphere Application Server のトランザクション・マネージャーにアクセスする際に、エラーが発生しました。

ユーザーの処置: 正しいレベルの WebSphere Application Server for z/OS がインストールされていることを確認してください。

ICO0068E **javax.resource.ResourceException: ICO0068E:***methodname* **error. Error obtaining the transaction object. [***java_exception***]**

説明: WebSphere Application Server for z/OS トランザクション・マネージャーを使用してトランザクションが開始されたかどうか判別する際に、エラーが発生しました。

ユーザーの処置: 正しいレベルの WebSphere Application Server for z/OS がインストールされていることを確認してください。

ICO0069E **javax.resource.spi .ResourceAllocationException ICO0069E:***methodname* **error. Error obtaining RRS transaction context token. IMSConnResourceException: RRS retcode=[***rrs_routinecode***].**

説明: グローバル・トランザクションを処理するためのリカバリー・リソース・サービス (RRS) トランザクション・コンテキスト・トークンを取得する際に、予期しない内部エラーが発生しました。

ユーザーの処置: 関連する RRS エラー・メッセージがないか、RRS ジョブ・ログを調べてください。 RRS 戻りコード (*rrs_routinecode*) に関する診断情報については、「*z/OS V1R9 MVS* プログラミング: リソース・リカバリー」、またはご使用のリリースの z/OS の同じマニュアルを参照してください。

関連情報:

「z/OS MVS プログラミング: 高水準言語向け呼び出し可能サービス」のリソース・リカバリー・サービス (RRS) 内の RRS 戻りコード

ICO0070E **javax.resource.spi.EISSystemException ICO0070E:***methodname* **error. IMS Connect reported an RRS error: IMS Connect Return Code=[***returncode***], RRS Routine name=[***rrs_routine***], RRS Return code=[***rrs_routinecode***].**

説明: リカバリー・リソース・サービス (RRS) の障害が原因で、IMS Connect がエラーを返しました。

ユーザーの処置: また、関連する IMS Connect および RRS のエラー・メッセージがないか、z/OS コンソールを調べてください。戻りコード (*returncode*) 値、および IMS Connect エラー・メッセージに関する診断情報については、「*IMS* メッセージおよびコード」を参照してください。RRS 戻りコード (*rrs_routinecode*) に関する診断情報については、「*z/OS V1R9 MVS* プログラミング: リソース・リカバリー」、またはご使用のリリースの z/OS の同じマニュアルに記載されている、RRS ルーチン名 (*rrs_routine*) の項を参照してください。

関連情報:

「z/OS MVS プログラミング: 高水準言語向け呼び出し可能サービス」のリソース・リカバリー・サービス (RRS) 内の RRS 戻りコード

ICO0071E **javax.transaction.xa.xAException ICO0071E:*methodname* error. A communication error occurred when processing the XA *commandtype* operation. [*java_exception*]**

説明: グローバル・トランザクションの処理中に、TCP/IP またはソケットの障害が発生した場合、または IMS Connect が停止した場合は、通信障害が発生する可能性があります。エラーが発生した接続は再使用されません。

ユーザーの処置:

- Java 例外を調べて、ホストとの接続に失敗した理由を判別してください。
- また、関連する IMS Connect または TCP/IP のエラー・メッセージがないか、z/OS コンソールを調べてください。
- TCP/IP 経由でシステムに到達できること、および IMS Connect が停止していないことを確認します。

エラー・メッセージのコマンド・タイプ (*commandtype*) は、グローバル・トランザクション中にこの通信障害が発生した段階 (準備、コミット、ロールバック、リカバリー、または解放) を指します。

ICO0072E **javax.transaction.xa.xAException: ICO0072E:*methodname* error. The associated UR for the Xid is not found.**

説明: トランザクション処理中に、特定のグローバル・トランザクション ID (XID) に結合されたリカバリー単位 (UR) が、手操作による介入または IMS Connect またはリカバリー・リソース・サービス (RRS) のエラーによって除去されました。

ユーザーの処置: WebSphere Application Server ログ内のトランザクション情報と Xid の取得方法については、WebSphere Application Server インフォメーション・センターを参照してください。XID とそれらに関連した UR をリストするための IMS コマンドについては、「*IMS Version 14 Commands, Volume 3*」を参照してください。その XID の UR がリストされていることを確認してください。グローバル・トランザクションが、トランザクションを手動で完了する必要がある状態のままになっていないことを確認してください。

関連資料:

「IMS バージョン 15 コマンド」の IMS Connect コマンド

関連情報:

WebSphere Application Server V8 の Knowledge Center

ICO0073E **javax.transaction.xa.xAException: ICO0073E:*methodname* error. RRS is not available.**

説明: リカバリー・リソース・サービス (RRS) が停止しているか、または RRS と IMS Connect の間の通信が終了しました。

ユーザーの処置: また、関連する IMS Connect および RRS のエラー・メッセージがないか、z/OS コンソールを調べてください。 RRS が z/OS システム上で停止していないことを確認してください。RRS が使用可能になっていることを確認するために使用できる IMS Connect のコマンドについては、「*IMS バージョン 14* コミュニケーションおよびコネクション」を参照してください。

関連資料:

「IMS バージョン 15 コマンド」の IMS Connect コマンド

関連情報:

「z/OS MVS プログラミング: 高水準言語向け呼び出し可能サービス」のリソース・リカバリー・サービス (RRS) 内の RRS 戻りコード

WebSphere Application Server V8 の Knowledge Center

ICO0074E **javax.transaction.xa.xAException: ICO0074E: The RRS *rrs_routine* call returns with a return code [*rrs_routinecode*].**

説明: グローバル・トランザクションの処理中に、リカバリー・リソース・サービス (RRS) エラー・メッセージが IMS Connect から渡されました。

ユーザーの処置: z/OS コンソールを調べて、関連する IMS Connect と RRS のエラー・メッセージがないか確認します。RRS 戻りコード (*rrs_routinecode*) の診断情報については、「*z/OS V1R9 MVS* プログラミング: リソース・リカバリー」、またはz/OS リリースのバージョン用の同資料内で、RRS ルーチン名 (*rrs_routine*) を見つけてください。

関連情報:

「z/OS MVS プログラミング: 高水準言語向け呼び出し可能サービス」のリソース・リカバリー・サービス (RRS) 内の RRS 戻りコード

ICO0075E **javax.transaction.xa.xAException: ICO0075E:*methodname* error. The transaction branch might have been heuristically completed. [*rrs_exception*]**

説明: リカバリー・リソース・サービス (RRS) エラーが IMS Connect から渡されました。このエラーは、トランザクションの処理が、コミット・フェーズ中にトランザクションの一部はコミットされ、一部はエラーが発生している状態のままになっている可能性があることを示しています。*rrs_exception* は、この問題に関連した RRS ルーチンと戻りコードを示す ICO0074E エラー・メッセージです。

ユーザーの処置: z/OS コンソールを調べて、関連する IMS Connect と RRS のエラー・メッセージがないか確認します。RRS 戻りコード (*rrs_routinecode*) の診断情報については、「*z/OS V1R9 MVS プログラミング:* リソース・リカバリー」、またはz/OS リリースのバージョン用の同資料内で、RRS ルーチン名 (*rrs_routine*) を見つけてください。

WebSphere Application Server ログ内のトランザクション情報とグローバル・トランザクション ID (XID) を取得する手順については、使用している WebSphere Application Server のバージョン用のインフォメーション・センターを参照してください。XID および関連したリカバリー単位 (UR) をリストする IMS Connect のコマンドについては、「*IMS バージョン 14 コミュニケーションおよびコネクション*」を参照してください。関係している XID と UR、および IMS にコミットされる必要があった結果を判別します。IMS 内の値を確認して、ヒューリスティック状態 (トランザクションの完了を判別できない状態) が発生したことを判別します。コミットされるはずだった結果と一致するように IMS 内のデータを修正する処置、または関連したその他のデータベースを修正してそのトランザクションを実行する前の状態に戻すための処置を決定します。

関連資料:

「IMS バージョン 15 コマンド」の IMS Connect コマンド

関連情報:

「z/OS MVS プログラミング: 高水準言語向け呼び出し可能サービス」のリソース・リカバリー・サービス (RRS) 内の RRS 戻りコード

WebSphere Application Server V8 の Knowledge Center

ICO0076E **javax.resource.ResourceException: ICO0076E:*methodname* error. An internal error occurred. [*rrs_exception*]**

説明: IMS Connect からリカバリー・リソース・サービス (RRS) エラー・メッセージに関する情報を抽出する際に、内部エラーが発生しました。*rrs_exception* は、エラーに関連した RRS ルーチンと戻りコードを示す ICO0074E エラー・メッセージです。

ユーザーの処置: z/OS コンソールを調べて、関連する IMS Connect と RRS のエラー・メッセージがないか確認します。RRS 戻りコード (*rrs_routinecode*) の診断情報については、「*z/OS V1R9 MVS プログラミング:* リソース・リカバリー」、またはz/OS リリースのバージョン用の同資料内で、RRS ルーチン名 (*rrs_routine*) を見つけてください。

関連情報:

「z/OS MVS プログラミング: 高水準言語向け呼び出し可能サービス」のリソース・リカバリー・サービス (RRS) 内の RRS 戻りコード

ICO0077E **javax.resource.ResourceException: ICO0077E:*methodname* error. The transaction has already rolled back. [*rrs_exception*]**

説明: リカバリー・リソース・サービス (RRS) エラーが IMS Connect から渡されました。このエラーは、同じリカバリー単位 (UR) でトランザクションのロールバックの試行が 2 回行われたことを示しています。RRS は、2 回目のロールバックが行われないようにして、このようなアクションが試行されたことを示すエラーをスローします。*rrs_exception* は、エラーに関連した RRS ルーチンと戻りコードを示す ICO0074E エラー・メッセージです。

ユーザーの処置: トランザクションはロールバックされているので、アクションは不要です。発生した RRS 障害についての詳細は、ICO0074E の資料を参照してください。予防措置として、トランザクションの実行前にデータの脱落または変更が生じていなかったことを確認してください。

関連情報:

1045 ページの『ICO0074E』

ICO0078E **javax.resource.ResourceException: ICO0078E: *methodname* error. A valid user-specified clientID is required for interactions on a dedicated persistent connection.**

説明: コミット・モードが 0 の場合は、clientID プロパティーに有効なユーザー指定の値が必要で、対話は専用の永続ソケット接続を使用します。この要件は、SYNC_SEND_RECEIVE、 SYNC_SEND、SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT、および SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT の対話に適用されます。

ユーザーの処置: clientID プロパティーに有効な値を指定します。有効な値は、次の規則に従う必要があります。

- ヌル・ストリングではない。
- 先頭がブランク・フィールドになっていない。
- IMS TM リソース・アダプターの予約接頭部 HWS で始まっていない。
- 長さが 8 文字である。
- 有効な文字 A から Z、0 から 9、および @、#、$ が使用されている。

ICO0079E **com.ibm.connector2.ims.ico .IMSDFSMessageException: ICO0079E:***methodname* **error. IMS returned a DFS message:***DFS_message*

説明: IMS は、IMS トランザクションの出力の代わりに DFS メッセージ (*DFS_message)* を返しました。対話が imsRequestType プロパティーの値として IMS_REQUEST_TYPE_IMS_TRANSACTION を使用している場合に、この例外がスローされます。

例えば、停止した IMS トランザクションを Java アプリケーションが実行しようとすると、この例外がスローされ、*DFS_message* の値は次のようになります。

```
DFS064 hh:mm:ss DESTINATION CAN NOT BE FOUND OR
       CREATED, DEST=
DFS065 hh:mm:ss TRAN/LTERM STOPPED
```

ユーザーの処置: 「*IMS* メッセージおよびコード」資料で *DFS_message* に対応する説明と対応を見つけ、IMS 内で問題に対処してください。

関連資料:

DFS メッセージ (IMS バージョン 15)

ICO0080E **javax.resource.spi.EISSystemException: ICO0080E:***methodname* **error. Execution timeout has occurred for this interaction. The executionTimeout was [***executionTimeout_value***] milliseconds. The IMS Connect TIMEOUT was used.**

説明: IMS Connect が IMS にメッセージを送信し、応答を受け取るためにかかった時間 (*executionTimeout_value)*) が、IMS Connect の TIMEOUT 値を超えていました。IMS Connect の TIMEOUT 値は、以下のとおりです。

- SYNC_SEND_RECEIVE 対話の場合は、IMS Connect の構成メンバー内で指定されます。
- SYNC_RECEIVE_ASYNCOUTPUT _SINGLE_NOWAIT、および SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT の対話の場合は 2 秒です。

この対話の executionTimeout プロパティーが指定されていないか、ゼロに設定されていたため、IMS Connect の TIMEOUT 値が使用されました。

ユーザーの処置: アプリケーションが有効な executionTimeout 値を設定していることを確認してください。executionTimeout 値を設定するには、WebSphere または Rational 統合開発環境を使用するか、setExecutionTimeout メソッドを使用できます。

ICO0081E **javax.resource.spi.EISSystemException: ICO0081E:***methodname* **error. Execution timeout has occurred for this interaction. The executionTimeout value specified was [***executionTimeout_value***] milliseconds. The value used by IMS Connect was [***rounded_executionTimeout_value***] milliseconds.**

説明: IMS Connect が IMS にメッセージを送信し、応答を受け取るまでにかかった時間 (*executionTimeout_value*) が、該当する実行タイムアウト間隔に丸められた executionTimeout 値 (*rounded_executionTimeout_value*) を超えていました。有効な実行タイムアウト値が設定された後、この値は IMS Connect が使用できる値に変換されます。

ユーザーの処置: 丸められた実行タイムアウト値が予期したものと異なる場合は、値が以下の変換規則に従っていることを確認します。

表 126. 実行タイムアウト値の変換規則

ユーザー指定値の範囲	変換規則
1 - 250	ユーザー指定の値が 10 で割り切れない場合は、10 の増分で次に大きな数に変換されます。
251 - 1000	ユーザー指定の値が 50 で割り切れない場合は、50 の増分で次に大きな数に変換されます。
1001 - 60000	ユーザー指定値は、1000 の増分でその値に最も近い数に変換されます。1000 の増分のちょうど中間にある値は、1000 の増分で次に大きい数に変換されます。
60001 - 3 600000	ユーザー指定値は、60000 の増分でその値に最も近い数に変換されます。 60000 の増分のちょうど中間にある値は、60000 の増分で次に大きい数に変換されます。

ICO0082E **javax.resource.NotSupportedException: ICO0082E:***methodname* **error. Execution timeout has occurred for this interaction. The executionTimeout value of [{***executionTimeout_value***}] milliseconds is not supported. The valid range is [{***executionTimeout_waitforever_flag***}, 0 to {***maximum_executionTimeout_value***}] milliseconds. The IMS Connect TIMEOUT was used.**

説明: executionTimeout プロパティーに対して指定された例外タイムアウト値が、最小または最大のタイムアウト値の要件を満たしていませんでした。

ユーザーの処置: アプリケーションが executionTimeout プロパティーに有効な値を設定していることを確認してください。実行タイムアウト値はミリ秒で表され、1 から 3600000 まで (両端を含む) の範囲の 10 進整数でなければなりません。時間制限なしで対話を実行する場合は、この値を -1 に設定することもできます。

関連資料:

1066 ページの『実行タイムアウト (executionTimeout)』

ICO0083E **javax.resource.ResourceException: ICO0083E:***methodname* **error. SYNC_SEND_RECEIVE, SYNC_SEND, SYNC_RECEIVE_ASYNCOUTPUT, SYNC_RECEIVE_ASYNCOUTPUT _SINGLE_NOWAIT and SYNC_RECEIVE_ASYNCOUTPUT _SINGLE_WAIT interactions with Commit Mode 0 are not valid within the scope of a global transaction.**

説明: グローバル・トランザクションの有効範囲内では、コミット・モード 0 の SYNC_SEND_RECEIVE、SYNC_SEND、 SYNC_RECEIVE_ASYNCOUTPUT (非推奨)、SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT (SYNC_RECEIVE_ASYNCOUTPUT を置き換える)、および SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT の対話は無効です。グローバル・トランザクションは、同期レベル SYNC_LEVEL_SYNCPOINT を指定したコミット・モード 1 の対話を行う必要があります。

ユーザーの処置:

- コミット・モード 0 を使用する場合は、アプリケーションが非トランザクション・アプリケーションとして構成されていることを確認してください。
- グローバル・トランザクションの有効範囲内で対話を実行する場合は、commitMode プロパティー値は 1 でなければなりません。

ICO0084E **javax.resource.ResourceException: ICO0084E:***methodname* **error. An unexpected internal IMS TM Resource Adapter error occurred. [***source_method***] [***source_exception***]**

説明: *methodname* 内で [*source_method*] 呼び出しを実行する際に、PrivelegedActionException が発生しました。Java EE セキュリティーが有効になっていて、呼び出し側プログラム *methodname*、または現行呼び出しスタック内のいずれかのプログラムに関連付けられたユーザーに、[*source_method*] を実行する許可が与えられていない場合に、この例外が発生します。

ユーザーの処置: 呼び出し側プログラム (および例外の発生時に現行呼び出しスタック内にあるプログラム) に関連したユーザーに [*source_method*] を実行する許可が与えられるように、アプリケーション・サーバーのセキュリティーを正しくセットアップしたことを確認してください。あるいは、アプリケーション・サーバーにおけ

る Java EE セキュリティー検査をオフにします。

ICO0085E **javax.resource.ResourceException: ICO0085E:** *methodname* **error. Protocol violation. A user-specified clientID is not allowed for interactions on a shareable persistent socket.**

説明: clientID プロパティーに指定された値は許可されていません。接続ファクトリーは共用可能永続ソケット用に構成されているため、この接続ファクトリー内ではユーザー指定のクライアント ID は許可されません。

ユーザーの処置: 共用可能永続ソケット接続ファクトリーの場合には、IMS TM リソース・アダプターが生成済みのクライアント ID を提供します。ユーザー指定のクライアント ID を使用することはできません。共用可能永続ソケットを使用しているかどうかを判別するには、対話で使用される接続ファクトリーの CM0Dedicated プロパティーの値が `false` であるかどうか確認します。

関連資料:

1059 ページの『クライアント ID (clientID)』

ICO0086E **javax.resource.ResourceException: ICO0086E:***methodname* **error. Invalid value was specified for CommitMode property.**

説明: commitMode プロパティーのフィールドに指定した値が無効です。

ユーザーの処置: アプリケーションが commitMode プロパティーに有効な値を設定していることを確認してください。サポートされている値は以下のとおりです。

- 1 (SEND_THEN_COMMIT) は、IMS がトランザクションを処理して応答を返した後で、データをコミットすることを示します。
- 0 (COMMIT_THEN_SEND) は、IMS がトランザクションを処理してデータをコミットした後で、応答を送信することを示します。

関連資料:

1065 ページの『コミット・モード (commitMode)』

ICO0087E **javax.resource.ResourceException: ICO0087E:** *methodname* **error. Protocol violation. commit mode 1 is not allowed for interactions on a dedicated persistent socket.**

説明: commitMode プロパティーに指定された 1 という値は無効です。接続ファクトリーは専用永続ソケット用に構成されているため、コミット・モード 1 は許可されません。

ユーザーの処置: 専用永続ソケット接続ファクトリーの場合は、コミット・モード 0 の対話のみが有効です。専用永続ソケットを使用しているかどうかを判別するには、対話で使用される接続ファクトリーの CM0Dedicated プロパティーの値が `true` であるかどうか確認します。

関連資料:

1060 ページの『CM0 専用 (CM0Dedicated)』

ICO0088E **javax.resource.ResourceException: ICO0088E:** *methodname* **error. Protocol violation. SYNC_RECEIVE_ASYNCOUTPUT _SINGLE_NOWAIT interactions are not allowed on a shareable persistent socket.**

説明: interactionVerb プロパティーに指定された値 SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT が無効です。接続ファクトリーは共用可能永続ソケット用に構成されているため、SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT は許可されません。

ユーザーの処置: 対話によって使用されている接続ファクトリーの CM0Dedicated プロパティーの値が `false` であるかどうか確認することにより、共用可能永続接続を使用しているかどうか判別します。共用可能永続接続の場合、対話は SYNC_SEND_RECEIVE、SYNC_SEND、および SYNC_END_CONVERSATION に設定してください。

ICO0089I **javax.resource.ResourceException: ICO0089I:** *methodname***. Non-persistent socket closed for Commit Mode 0 IMS transaction.**

説明: 非永続ソケット (トランザクション・ソケット) を使用してコミット・モード 0 を実行すると、IMS TM リソース・アダプターは接続プールから管理接続オブジェクトを強制的に除去します。

ユーザーの処置: アクションは不要です。

ICO0091E **javax.resource.ResourceException: ICO0091E:** *methodname* **error.SSL client context could not be created. [{1}]**

説明: 以下の理由のいずれかのため、SSL コンテキストを作成できませんでした。

- 鍵ストアの整合性をチェックするアルゴリズムが見つからなかった。
- 鍵ストア内の証明書をロードできなかった。
- 鍵を復旧できなかった (例えば、指定したパスワードが間違っている)。

ユーザーの処置: 以下のことを確認します。

- 証明書を作成するアルゴリズムが、IBM Java Secure Socket Extension プロバイダー (IBMJSSE) によってサポートされている。
- 鍵ストアとトラストストアのパスワードが正しい。

ICO0096I **javax.resource.ResourceException: ICO0096I:** *methodname* **Warning. Invalid value provided for SSL parameter.**

説明: *methodname* に示されたメソッドが、ヌルまたは空の SSLKeystoreName、 SSLKeystorePassword、SSLTruststoreName、または SSLTruststorePassword のいずれかのパラメーターを使用して呼び出されました。この情報メッセージにより、プログラムの実行が終了することはありません。

ユーザーの処置: SSLKeystoreName、SSLKeystorePassword、 SSLTruststoreName、および SSLTruststorePassword の各パラメーターに有効な値を指定します。都合に応じて、鍵ストアまたはトラストストアのどちらかに秘密鍵と証明書を保管できます。正しく実行するには、有効な値のセット (SSLKeystoreName と SSLKeystorePassword、または SSLTruststoreName と SSLTruststorePassword のどちらか) が 1 つだけ必要です。

ICO0097E **javax.resource.ResourceException: ICO0097E:***methodname* **error. The given value is invalid for 'SSLEncryptionType'. The value must be 'STRONG' for strong encryption, 'WEAK' for weak (export) encryption or 'ENULL' for null (no) encryption.**

説明: SSLEncryptionType パラメーターに STRONG、WEAK、または ENULL 以外の値が指定されていました。

ユーザーの処置: SSLEncryptionType パラメーターに STRONG、WEAK、または ENULL を指定します。値に大/小文字の区別はありません。

関連資料:

1061 ページの『SSL 暗号化タイプ (SSLEncryptionType)』

ICO0111E **javax.resource.ResourceException: ICO0111E:***methodname* **error. SSLEnabled must be set to FALSE when using Local Option.**

説明: プロパティー SSLEnabled が `true` に設定されており、プロパティー IMSConnectName が非ヌル値に設定されています。これは、ローカル・オプション接続が使用されることを示します。ただし、SSL はローカル・オプション接続ではサポートされません。

ユーザーの処置: SSLEnabled プロパティーを `false` に設定します。

ICO0112E **ICO0112E:***methodname* **error. Connection is closed due to transaction timeout.**

説明: WebSphere Application Server トランザクション・タイムアウトのために、接続が閉じられました。

ユーザーの処置: WebSphere Application Server トランザクション・タイムアウト値を増やして、トランザクションを完了させるためにより多くの時間を割り当てます。

ICO0113E **javax.resource.spi.CommException: ICO0113E:** *methodname* **error. Socket timeout has occurred for this interaction. The socket timeout value specified was [***socket_timeout_value***] milliseconds. [***source_exception***:***exception_reason***]**

説明: IMS TM リソース・アダプターが IMS Connect から応答を受け取るための実行タイムアウト値が、ソケット・タイムアウトに指定された値を上回っています。

ユーザーの処置: socketTimeout プロパティーの値が、IMS TM リソース・アダプターが IMS Connect から応答を受信するために十分な時間であることを確認してください。時間が不十分な場合には、値を大きくしてください。指定されている socketTimeout プロパティーの値が十分な場合は、ネットワークの問題が原因で遅延が生じている可能性があります。ネットワーク管理者に連絡してください。

関連資料:

1073 ページの『ソケット・タイムアウト (socketTimeout)』

ICO0114E **javax.resource.ResourceException: ICO0114E:** *methodname* **error. The socket timeout value of [***socket_timeout_value***] milliseconds is not valid. [***source_exception***:***exception_reason***]**

説明: socketTimeout プロパティーに指定された値 *socket_timeout_value* が無効です。

ユーザーの処置: 提供された例外の理由を確認してください。socketTimeout プロパティーが正の数値に設定されていることを確認します。

関連資料:

1073 ページの『ソケット・タイムアウト (socketTimeout)』

ICO0115E **javax.resource.spi.CommException: ICO0115E:** *methodname* **error. A TCP Error occurred.**

説明: 基礎となるネットワーク・プロトコルに問題が発生しました。

ユーザーの処置: ネットワーク管理者に連絡してください。

ICO0116E **ICO0116E:***methodname* **error. A Common Client Interface error occurred.**

説明: 基礎となるプロトコルでエラーが発生しました。

ユーザーの処置: IBM ソフトウェア・サポートに連絡してください。

ICO0117E **javax.resource.ResourceException: ICO0117E:** *methodname* **error. Protocol violation: Commit Mode 1 is not allowed for SYNC_SEND, SYNC_RECEIVE_ASYNCOUTPUT, SYNC_RECEIVE_ASYNCOUTPUT _SINGLE_NOWAIT and SYNC_RECEIVE_ASYNCOUTPUT _SINGLE_WAIT interactions.**

説明: IMS TM リソース・アダプターは、SYNC_SEND 対話に対してコミット・モード 0 のみをサポートします。

ユーザーの処置: コミット・モードまたは対話 verb のどちらかを修正します。SYNC_SEND、SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT、および SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT の対話には、コミット・モード 0 が必要です。コミット・モード 1 は、SYNC_SEND_RECEIVE および SYNC_END_CONVERSATION の対話に対して有効です。

関連概念:

954 ページの『コミット・モードと同期レベルの組み合わせによってサポートされる対話』

ICO0118E **javax.resource.ResourceException: ICO0118E:** *methodname* **error. Protocol violation. IMS request type 2 (IMS_REQUEST_TYPE_IMS _COMMAND) is not allowed for SYNC_SEND, SYNC_END_CONVERSATION, SYNC_RECEIVE_ASYNCOUTPUT, SYNC_RECEIVE_ASYNCOUTPUT _SINGLE_NOWAIT and SYNC_RECEIVE_ASYNCOUTPUT _SINGLE_WAIT interactions.**

説明: imsRequestType プロパティーに対して指定された値 2 (IMS_REQUEST_TYPE_IMS_COMMAND) が無効です。

ユーザーの処置: ImsRequestType プロパティーの値 2 (IMS_REQUEST_TYPE_IMS_COMMAND) は、SYNC_SEND_RECEIVE の対話に対してのみ有効です。SYNC_SEND、 SYNC_END_CONVERSATION、SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT、および SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT の対話の場合は、値 1 (IMS_REQUEST_TYPE_IMS_TRANSACTION) を指定する必要があります。

ICO0119E **javax.resource.ResourceException: ICO0119E:** *methodname* **error. A supported SSL provider was not found. [***caught_exception***]**

説明: IMS Connect との Secure Sockets Layer (SSL) TCP/IP 接続を初期化する際に、IMS TM リソース・アダプターはサポートされる 2 つのプロバイダー (com.ibm.jsse.JSSEProvider または sun.security.provider.Sun) のいずれかを使用する必要があります。このエラーは、これらのプロバイダーのどちらも使用可能ではないことを示しています。

ユーザーの処置: プロバイダー com.ibm.jsse.JSSEProvider は、デフォルトで IBM JVM に追加されています。 Sun JVM では、プロバイダー sun.security.provider.Sun がデフォルトで追加されま

す。IMS TM リソース・アダプターを実行している環境が、WebSphere Application Server 内で実行している場合はサポートされる IBM JVM であること、その他のアプリケーション・サーバーの場合は Sun JVM であることを確認してください。

ICO0121E **javax.resource.ResourceException: ICO0121E:** *methodname* **error. Invalid reRoute name value. Prefix HWS is reserved for use by IMS TM Resource Adapter.**

説明: reRouteName プロパティーの値が無効です。接頭部 HWS は、IMS TM リソース・アダプターのみが使用するために予約されています。

ユーザーの処置: reRouteName プロパティーに有効な値を指定します。詳しくは、reRouteName プロパティーに関する情報を参照してください。

関連資料:

1072 ページの『転送名 (reRouteName)』

ICO0122E **javax.resource.ResourceException: ICO0122E:** *methodname* **error. Invalid reRoute value. When purgeAsyncOutput value is true, reRoute value cannot be true.**

説明: reRoute プロパティーの値が無効です。これは、purgeAsyncOutput プロパティーの値が `true` に設定されているか、purgeAsyncOutput プロパティーにデフォルト値 (`true`) が使用されているためです。

ユーザーの処置: reRoute プロパティーを `true` に設定する場合は、purgeAsyncOutput プロパティーを `false` に設定します。

関連資料:

1071 ページの『非同期出力のパージ (purgeAsyncOutput)』

1072 ページの『転送 (reRoute)』

ICO0123E **javax.resource.NotSupportedException: ICO0123E :** *methodname* **error . A Sync Level value of** *synclevel* **is not supported for commit-then-send (Commit mode 0) interactions.**

説明: syncLevel プロパティーに指定された値は、「コミット後送信」(コミット・モード 0) の対話ではサポートされていません。「コミット後送信」(コミット・モード 0) の対話の場合は、値 1 (確認) のみが syncLevel プロパティーの値としてサポートされています。

ユーザーの処置: syncLevel プロパティーに値 1 (確認) を指定するか、または「コミット後送信」(コミット・モード 0) の対話のデフォルトである同期レベル値 1 を受け入れます。

関連資料:

1074 ページの『同期レベル (syncLevel)』

ICO0124E **javax.resource.ResourceException: ICO0124E :** *methodname* **error . SYNC_SEND_RECEIVE interactions with Sync Level Confirm are not supported with Local Option.**

説明: 「コミット後に送信」(コミット・モード 0) の対話は、ローカル・オプションではサポートされていません。ローカル・オプションでは、確認 (1) の同期レベルが指定された「送信後にコミット」(コミット・モード 1) の対話はサポートされません。

ユーザーの処置: この機能にはローカル・オプションを使用しないでください。

ICO0125E **javax.resource.EISSystemException: ICO0125E :** *methodname* **error. An internal error occurred. The status of the IMS transaction associated with this SYNC_SEND_RECEIVE interaction with Commit Mode 1 and Sync Level 1 cannot be determined.**

説明: 「送信後コミット」(コミット・モード 1) 確認 (1) 同期レベルのプロトコルの一部として、IMS TM リソース・アダプターが IMS トランザクションから出力メッセージを受信し、IMS Connect に肯定応答を送信しましたが、トランザクションがコミットされたという IMS Connect からの正しい通知の代わりに、エラー通知を受信しました。IMS TM リソース・アダプターが受信した出力メッセージは、アプリケーション・コンポーネントに返されませんでした。IMS トランザクションはコミットされた可能性も、コミットされなかった可能性もあります。

ユーザーの処置: ターゲット IMS システム上のタイプ X'01' および X'03' IMS ログ・レコードを参照して、未完了トランザクションがコミットされたか、または打ち切られたかを判別します。必要に応じて、IMS システムと外部アプリケーションの状態の調整を行います。

ICO0126E **javax.resource.ResourceException: ICO0126E : *methodname* error . IMS Connect reported a Commit Mode 1 Sync Level Confirm error: IMS Connect Return Code=[*returncode*], Reason Code=[*reasoncode*]. *reasoncode_string***

説明: IMS Connect がエラーを返しました。エラーが発生した接続は再使用されません。*reasoncode_string* は、理由コードの簡略説明を示します (提供されている場合)。

ユーザーの処置: 関連する IMS Connect のエラー・メッセージについては、z/OS コンソールを確認してください。IMS Connect エラー・メッセージは HWS という文字で始まります。

ICO0127E **com.ibm.ims.ico .IMSIllegalStateException: ICO0127E : *methodname* error. Protocol violation. The Mode [*mode*] is not allowed for the current state [*state*]. [*java_exception*]**

説明: IMS TM リソース・アダプターが正しくない状態にあります。

ユーザーの処置: IMS TM リソース・アダプターまたは IMS Connect でエラーが発生している可能性があります。IBM サービス担当員にお問い合わせください。

ICO0128E **javax.resource.NotSupportedException ICO0128E : *methodname* error. The Sync Level property value of [*synclevel*] given is invalid. Sync Level NONE (0) and Sync Level CONFIRM (1) are the only values supported by the setSyncLevel(int) method.**

説明: 0 または 1 以外の値が setSyncLevel(int) メソッドへの入力として指定されました。

ユーザーの処置: setSyncLevel(int) メソッドへの同期レベル・プロパティー値の入力として、0 または 1 のいずれかを指定します。

関連資料:

1074 ページの『同期レベル (syncLevel)』

ICO0129E **javax.resource.ResourceException: ICO0129E:*methodname* error. Specifying the Alternate ClientID is not allowed on a dedicated persistent connection. The Alternate ClientID value is supported on shareable persistent socket connections ONLY.**

説明: 代替クライアント ID を使用した非同期出力のリトリーブは、共用可能永続ソケット接続の場合のみサポートされます。

ユーザーの処置: 代替クライアント ID を使用して非同期出力をリトリーブするには、共用可能永続ソケット接続を使用します。

関連資料:

1064 ページの『代替クライアント ID (altClientID)』

ICO0130E **javax.resource.ResourceException: ICO0130E:*methodname* error. The ignorePURGCall property is not allowed on a dedicated persistent connection.**

説明: SYNC_SEND_RECEIVE の対話の専用永続接続では、ignorePURGCall プロパティーはサポートされません。専用永続接続に対してこのプロパティーを設定しようとすると、例外がスローされます。

ユーザーの処置: この対話に対しては共用可能永続ソケット接続を使用するか (推奨される解決策)、アプリケーションで専用永続接続を使用する必要がある場合は ignorePURGCall プロパティーを設定しないでください。

関連資料:

1067 ページの『PURG 呼び出しの無視 (ignorePURGCall)』

ICO0131E **javax.resource.ResourceException: ICO0130E:*methodname* error. The property ignorePURGCall is not supported for interaction verb [SYNC_END_CONVERSATION or SYNC_RECEIVE_ASYNCOUTPUT_*]. The property ignorePURGCall can only be specified for interaction SYNC_SEND and SYNC_SEND_RECEIVE.**

説明: ignorePURGCall プロパティーは、SYNC_END_CONVERSATION、SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT、

および
SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT の対話ではサポートされません。

ユーザーの処置: 共用可能永続ソケット接続では、SYNC_SEND または SYNC_SEND_RECEIVE の対話を使用します。SYNC_END_CONVERSATION、SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT、または
SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT の対話を実行する意図がある場合は、ignorePURGCall プロパティーに値 `true` を指定しないでください。

関連資料:

1067 ページの『PURG 呼び出しの無視 (ignorePURGCall)』

ICO0132E **IMS Connect returned an error during a conversational transaction: RETCODE=[{1}], REASONCODE=[{2}]. {3}**

説明: IMS Connect は、会話型トランザクション中にエラー戻りコードと理由コードを返しました。

ユーザーの処置: 「*IMS* メッセージおよびコード」資料を参照し、特定の戻りコードと理由コードに対して推奨されている適切な処置を行います。

関連資料:

「IMS バージョン 15 コマンド」の IMS Connect コマンド

ICO0133E **The interactionVerb must be either SYNC_SEND_RECEIVE or SYNC_END_CONVERSATION when the value of the IMSInteractionSpec property useConvID is true (client-managed conversation state programming model).**

説明: IMSInteractionSpec オブジェクトの interactionVerb プロパティーが、SYNC_SEND_RECEIVE または SYNC_END_CONVERSATION に設定されていません。

ユーザーの処置: IMSInteractionSpec オブジェクトの対話 Verb 設定に、SYNC_SEND_RECEIVE または SYNC_END_CONVERSATION のどちらかを使用します。

関連資料:

1068 ページの『対話 Verb (interactionVerb)』

ICO0134E **When the value of IMSInteractionSpec property useConvID is true (client-managed conversation state programming model), conversational transactions are supported on shareable persistent socket connections ONLY.**

説明: 会話 ID を使用して会話の反復を追跡する (IMSInteractionSpec useConvID プロパティーを true に設定することにより) 会話型トランザクションは、共用可能永続ソケット接続でのみサポートされます。

ユーザーの処置: クライアントが会話状態を管理するプログラミング・モデルでは、共用可能永続ソケット接続を使用します。

ICO0135E **When the value of IMSInteractionSpec property useConvID is true (client-managed conversation state programming model), conversational transactions are supported with Commit Mode 1 ONLY.**

説明: 会話 ID を使用して会話の反復を追跡する (IMSInteractionSpec useConvID プロパティーを true に設定することにより) 会話型トランザクションは、コミット・モード 1 でのみサポートされます。

ユーザーの処置: クライアントが会話状態を管理するプログラミング・モデルでは、コミット・モード 1 を使用します。

ICO0136I **The duplicate [queue] name *name* was removed. The list of [queue] names was reset to *new_list*.**

説明: キュー名またはデータ・ストア名の重複は許可されません。

ユーザーの処置: このメッセージは通知目的であり、ユーザー処置は必要ありません。

ICO0137I **The maximum length of the property *property_name* was exceeded. *old_property_value* has been truncated to *old_property_value*.**

説明: このプロパティーの最大長は 8 文字です。

ユーザーの処置: このメッセージは通知目的であり、処置は必要ありません。

ICO0138E The value for *property_name* is invalid. The prefix HWS is reserved for use by IMS TM Resource Adapter.

説明: *property_name* 値の中で、予約済みの接頭部 HWS が使用されました。

ユーザーの処置: 予約済みの接頭部を使用しない値を *property_name* プロパティーに指定します。

ICO0139E The message endpoint threw an exception. *endpoint_exception*

説明: メッセージ駆動型 Bean (MDB) によって、Java 例外 (*endpoint_exception*) が IMS TM リソース・アダプターに渡されました。

ユーザーの処置: MDB によって IMS TM リソース・アダプターに渡された例外を解決してください。

ICO0140I Reconnecting in *number_of_seconds* seconds...

ユーザーの処置: このメッセージは通知目的であり、ユーザー処置は必要ありません。

ICO0141E The endpoint is already active.

説明: 既に存在しているエンドポイントの追加が試行されました。

ユーザーの処置: このエンドポイントが削除されるまで、追加を試行しないでください。

ICO0142E The endpoint is not found.

説明: 存在していないエンドポイントへのアクセスが試行されました。

ユーザーの処置: このエンドポイントの削除を試行しないでください。

ICO0143E An unexpected error occurred when the message was dispatched from IMS to the message endpoint. *java_exception*

ユーザーの処置: IBM ソフトウェア・サポートに連絡し、スローされた Java 例外を伝えてください。

ICO0144E The IMS TM resource adapter encountered an error while sending the message to IMS.

説明: メッセージが IMS へ送信されたときに、予期しない内部エラーが発生しました。

ユーザーの処置: IBM ソフトウェア・サポートに連絡してください。

ICO0145E The IMS TM resource adapter encountered an error while sending the response message for a synchronous callout request to IMS. *java_exception*

説明: このプロパティーの最大長は 8 文字です。

ユーザーの処置: Java 例外を調べて、メッセージの送信に失敗した原因を判別してください。IMS で /DISPLAY TMEMBER TPIPE *queue_name* SYNC コマンドを発行して、キューが無効状態になっていないことを確認してください。

ICO0146E An error occurred when the error message for a synchronous callout request was back to IMS. *java_exception*

説明: このメッセージは、同期コールアウト要求時に IMS に戻されるメッセージ内で生成されたエンコード・エラー、またはメッセージ・タイプ・エラーの結果です。

ユーザーの処置: Java 例外またはエラー・メッセージの資料を調べて、元のメッセージがエラーを生成した理由を判別してください。

ICO0147E No message listeners were started. There are no queues on which to listen.

ユーザーの処置: IMSActivationSpec クラスの queueNames プロパティーに、少なくとも 1 つの有効なキュー名を指定してください。

ICO0148E The ActivationSpec *activationSpec_name* is not supported.

説明: サポートされない ActivationSpec がメソッドに渡されました。

ユーザーの処置: サポートされる ActivationSpec (IMSActivationSpec) を使用してください。

ICO0150E	**The IMS TM Resource Adapter is unable to listen for callout messages from IMS on queue** *queue_name* **because it is unable to connect to host** *host_name*, **port** *port_number*.

説明: TM リソース・アダプターが、指定されたアドレスのホストから応答を受信しませんでした。

ユーザーの処置: ActivationSpec で有効なホスト名とポート番号を指定していることを確認してください。

その他の例外およびエラー・メッセージ

IMS Connect、または WebSphere Application Server が、IMS TM リソース・アダプター、または IMS TM リソース・アダプターによって使用されているクラス・ライブラリーからスローされたエラー・メッセージと例外をキャッチすると、エラーが返される場合があります。

J2CA0056I

IMS TM リソース・アダプターが例外をスローすると、Java アプリケーション以外のコンポーネント (例えば、WebSphere Application Server) がその例外をキャッチする可能性があります。

例えば、WebSphere Application Server が Java アプリケーションから例外をキャッチすると、IMS TM リソース・アダプター例外からのメッセージを含む、独自のメッセージを発行する場合があります。例えば、実行タイムアウトが発生すると、WebSphere Application Server コンソールに以下のメッセージが表示されます。

```
J2CA0056I: The Connection Manager received a fatal connection error
from the
Resource Adaptor for resource myConnFactry.  The exception which
was received is
ICO0080E:
com.ibm.connector2.ims.ico.IMSTCPIPManagedConnection@e59583c.
processOutputOTMAMsg(byte[],IMSInteractionSpec, int) error.
Execution timeout has occurred for this interaction.
The executionTimeout was [0] milliseconds.  The IMS Connect TIMEOUT
was used.
```

J2CA0056I は、WebSphere Application Server からの通知メッセージです。実行がタイムアウトになったときに IMS Connect がソケットをクローズし、WebSphere Application Server Connection Manager が接続プールからソケットの接続オブジェクトを除去したため、致命的な接続エラーが発生しています。

別の例としては、トランザクション (非永続) ソケットがコミット・モード 0 の対話に使用される場合があります。この場合は、WebSphere Application Server コンソールに以下のメッセージが表示されます。

```
J2CA0056I: The Connection Manager received a fatal connection error
from the
Resource Adaptor for resource  myConnFactry.  The exception which
was received is ICO0089I:
com.ibm.connector2.ims.ico.IMSTCPIPManagedConnection@6db5d83a.call(Connection,
InteractionSpec, Record, Record). Non-persistent socket closed for
Commit Mode 0 IMS transaction.
```

関連情報:

WebSphere Application Server バージョン 8 の資料

WLTC0017E

WLTC0017E は、WebSphere Transaction Monitor によって生成され、ローカル・トランザクション内包 (LTC) に参加しているリソースがコミットされる代わりにロールバックされたことを示します。

LTC は、未指定のトランザクション・コンテキストでアプリケーション・サーバーの動作を定義するために使用されます。例えば、トランザクション属性 NotSupported を持つ、コンテナーによって管理されるエンタープライズ Java Bean (EJB) 内の単一メソッドがすべてのトランザクションの範囲外で呼び出された場合、WebSphere Application Server はそのメソッドの実行時に使用されるリソースを処理するためのローカル・トランザクションを作成します。

メッセージ WLTC0017E は、setRollbackOnly() メソッドが LTC に対して呼び出されたために、LTC に参加していたリソースがコミットされる代わりにロールバックされたことを示します。このメッセージは、ユーザーによる処置を必要とせず、通知のみを目的としています。

WLTC0017E: `Resources rolled back due to setRollbackOnly() being called.`

WebSphere Application Server メッセージの接頭部は、メッセージを発行したコンポーネントを示します。これらのメッセージの資料は、WebSphere Application Server インフォメーション・センターにあります。

関連情報:

WebSphere Application Server バージョン 8 の資料

HWSP1445E

メッセージ HWSP1445E は、接続に Secure Sockets Layer (SSL) を使用するように指定する一方で、Java アプリケーションに非 SSL ポートを指定すると、IMS Connect で発生します。

Java アプリケーションで使用する接続ファクトリーを構成する際には、SSL を使用するかどうかを SSLEnabled プロパティーによって指定します。SSL を使用する場合は (SSLEnabled=TRUE)、指定するポート番号が IMS Connect 内で SSL ポートとして構成されている必要があります。Java アプリケーションに非 SSL ポートを指定すると、アプリケーションの実行時に予期しない結果が生じます。

- IMS TM リソース・アダプターは、以下のように通信エラーを示す例外をスローします。

```
javax.resource.spi.CommException:
ICO0003E:
com.ibm.connector2.ims.ico.IMSTCPIPManagedConnection@56503fc6.connect()
error.
Failed to connect to host [CSDMEC13], port [9999].
[java.net.SocketException:
Connection reset by peer: socket closed]
```

- 以下の IMS Connect メッセージが z/OS コンソールに表示されます。

```
HWSP1445E UNKNOWN EXIT NAME SPECIFIED IN MESSAGE PREFIX; MSGID=
   /9  * !hR, M=SDRC
```

SSL 接続を確立する最初のステップでは、SSL ハンドシェーク・プロトコルを実行します。このプロトコルでは、クライアント (IMS TM リソース・アダプター) がサーバー (IMS Connect) に SSL「Hello」メッセージを送信します。このシナリオでは、IMS Connect は非 SSL ポートで着信メッセージを待機しています。IMS Connect は、ハンドシェーク・メッセージを受け取ると、接頭部に有効な出口名を持つ OTMA メッセージとしてメッセージを解釈し、メッセージ HWSP1445E を発行します。

HWSSSL00E

メッセージ HWSSSL00E は、接続に Secure Sockets Layer (SSL) を使用しないように指定する一方で、Java アプリケーションに指定したポート番号が SSL ポートである場合に、IMS Connect で発生します。

このエラー・シナリオでは、以下のエラーが発生します。

- IMS TM リソース・アダプターは、以下のように通信エラーを示す例外をスローします。

```
javax.resource.spi.CommException: ICO0005E:
com.ibm.connector2.ims.ico.IMSTCPIPManagedConnection@5bcdcdd4.receive()
error. A communication error occurred while sending or receiving
the IMS message.
[java.net.SocketException: Connection reset by peer: socket closed]
```

- 以下の IMS Connect メッセージが z/OS コンソールに表示されます。

```
HWSSSL00E Unable to initialize the SSL socket:Error while reading
or writing data
```

IMS Connect が SSL ハンドシェーク・プロトコルの一部である初期のクライアントの「Hello」メッセージを受信しないため、IMS Connect が SSL ソケットを初期化しようとして失敗しました。

第 51 章 参照情報

IMS TM リソース・アダプターには、IMS との対話および接続を管理するだけでなく、IMS から Java クライアント・アプリケーションへのインバウンド通信も管理するために、以下のクラスが備わっています。

IMS 接続ファクトリーのプロパティー

IMS TM リソース・アダプター接続ファクトリーのプロパティーでは、ターゲットのエンタープライズ情報システムの特性が記述されます。

これらの接続プロパティーについて、以下のリストで説明しています。

関連情報:

IMS TM Resource Adapter バージョン 15 Java API 仕様

クライアント ID (clientID)

このプロパティーの値を指定する必要があるのは、アプリケーション・コンポーネントがこの特定のクライアントの識別に IMS Connect への専用永続ソケット接続を使用する場合です。

(CM0Dedicated プロパティーが `true` に設定されている場合) ソケット接続を専用で使用しているクライアントを識別するために、クライアント ID の値を指定する必要があります。専用永続ソケット接続は必ず、以下の対話 Verb でコミット・モード 0 の対話の場合にのみ使用してください。

- SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT
- SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT (非推奨になった SYNC_RECEIVE_ASYNCOUTPUT に代わるものです)
- SYNC_SEND_RECEIVE

(CM0Dedicated プロパティーが `false` に設定されている場合)、クライアント ID はコミット・モード 0 およびコミット・モード 1 の対話によって使用できる共用可能永続ソケット接続で生成されます。

共用可能永続ソケットでのクライアント ID は、IMSConnectionMetaData の getClientID メソッドを使用して取得できます。IMSConnectionMetaData クラスには IMSConnection クラスからアクセスできます。このメソッドを IMS バージョン 12 で使用する場合は、APAR PM75222 (V12.1.2) が必要です。

有効なユーザー指定のクライアント ID は、以下の要件を満たしている必要があります。

- 1 文字から 8 文字までの英数字 (A から Z、0 から 9) または特殊文字 (@、#、$) のストリングでなければなりません。
- 先頭に文字ストリング HWS が付いていてはなりません。
- IMS Connect のポート番号であってはなりません。

- 小文字を指定した場合、その文字は大文字に変更されます。

クライアント ID は、共用可能、専用のどちらの場合も、そのソケットに専用です。他のプロパティーとは異なり、元のクライアント ID はソケットが切断されるまで変更されません。

CM0 専用 (CM0Dedicated)

このオプションのプロパティーは、TCP/IP 接続のみに適用されます。デフォルト値は `false` です。これは、接続ファクトリーが、コミット・モード 0 およびコミット・モード 1 の対話で使用できる共用可能永続ソケット接続を生成することを示します。

値が `false` の場合、IMS TM リソース・アダプターが、ソケット接続を識別するクライアント ID を生成します。

値が `true` の場合、接続ファクトリーが専用永続ソケット接続を生成することを示します。この接続には、ソケット接続を識別するユーザー指定のクライアント ID が必要です。それぞれの専用永続ソケット接続は、1 つの特定のクライアント ID 用に予約されます。専用永続ソケット接続ではコミット・モード 0 の対話のみが可能です。

データ・ストア名 (dataStoreName)

この必須値は、ターゲット IMS データ・ストアの名前です。

IMS データ・ストア名は、ターゲット IMS Connect 構成メンバー内の Datastore ステートメントの ID パラメーターです。また、この名前は IMS Connect と IMS OTMA の間の内部 XCF 通信時に IMS の XCF メンバー名としても使用されます。dataStoreName プロパティーの値では大/小文字が区別されます。

グループ名 (groupName)

このオプションの値は、デフォルトのユーザー名が使用された場合に、この接続ファクトリーが作成するすべての接続に使用する IMS のグループ名です。

groupName プロパティーは、コンポーネント管理の EIS サインオン環境内でのみ指定できます。

ホスト名 (hostName)

ターゲットの IMS Connect が実行されているシステムの IP アドレスまたはホスト名を指定する必要があります。この値は、ローカル・オプション接続では無視されます。

パスワード (password)

このオプションの値は、デフォルトのユーザー名が使用された場合に、この接続ファクトリーが作成する接続に使用されるパスワードです。

パスワード・フレーズ (passwordPhrase)

RACF などのセキュリティー・アクセス機能が、userName プロパティーに指定されたユーザーの認証に使用するパスワード・フレーズです。

V12.2、V13.2、およびそれ以降 このプロパティーは、IMS TM Resource Adapter バージョン 12.2.0、バージョン 13.2.0、およびそれ以降に適用されます。

パスワード・フレーズは 9 文字から 100 文字にする必要があります。

ポート番号 (portNumber)

ターゲットの IMS Connect が TCP/IP 接続に使用するポート番号を指定する必要があります。このプロパティーは、ローカル・オプション接続では無視されます。

単一の TCP/IP ポートで複数のソケットを開くことができ、IMS Connect は、IMS TM リソース・アダプターや他のクライアントとの通信に複数のポートを使用するように構成できます。

SSL 使用可能 (SSLEnabled)

このオプションのプロパティーは、TCP/IP 接続のみに適用されます。デフォルト値は `false` で、これは、portNumber プロパティーで指定されているポートへの接続に SSL ソケットが使用されることを示します。

値 `true` は、この接続ファクトリーにより、その接続プロパティーで指定されたホスト名とポート番号を使用して IMS Connect への SSL ソケット接続が作成されることを示します。このポート番号は、IMS Connect の構成で SSL ポートとして構成する必要があります。

SSL 暗号化タイプ (SSLEncryptionType)

このオプションのプロパティーは、TCP/IP 接続のみに適用され、SSL 暗号化タイプを指定します。SSLEnabled プロパティーは、`true` に設定する必要があります。

有効な暗号化タイプは STRONG、WEAK および ENULL です。 STRONG と WEAK は暗号の強度を反映し、強度は鍵の長さに関連します。エクスポートに使用できる暗号はすべて WEAK カテゴリーに含まれ、それ以外は STRONG カテゴリーに含まれます。

デフォルトでは、暗号化タイプは WEAK に設定されます。ENULL が指定された場合、IMS TM リソース・アダプターは名前にストリング「NULL」が含まれる暗号仕様を使用します。 Null 暗号化を使用すると、SSL ハンドシェーク・プロセス中の認証が可能になります。必要な認証を含む、ソケットのハンドシェーク・プロセスが完了すると、すべてのメッセージがそのソケットを介して流れるようになります。

SSLEncryptionType プロパティーでは大/小文字は区別されません。

SSL 鍵ストア名 (SSLKeyStoreName)

このオプションのプロパティーは TCP/IP 接続のみに適用され、SSLEnabled プロパティーが `true` に設定されている場合にのみ適用されます。この値には、鍵ストアの名前 (絶対ファイル・パスを含む) が含まれています。

秘密鍵およびそれぞれの関連する公開鍵証明書は、鍵ストアと呼ばれる、パスワードで保護されたデータベースに保管されます。利便性を図るために、信頼できる証明書も鍵ストアに保管できます。 SSLKeyStoreName プロパティーは、空にすることも、鍵ストア・ファイルを指すように設定することもできます。SSLKeyStoreName または SSKeyStorePassword のいずれかのプロパティーが空の場合、サーバー・ログに通知メッセージが生成されます。

z/OS 以外のプラットフォームの場合は、JKS 鍵ストア・ファイルの完全修飾パス名を指定します。 JKS 鍵ストア・ファイルの完全修飾パス名は、例えば `c:¥keystore¥MyKeystore.ks` です。

z/OS の場合、SSLKeyStoreName プロパティーを使用して、JKS 鍵ストアまたは RACF 鍵リングのいずれかを指定できます。JKS ストアの場合は、JKS 鍵ストア・ファイルの絶対パスを使用してその名前を指定します。 RACF 鍵リングの場合は、その RACF 鍵リングへのアクセスに必要な情報を提供するストリングを指定します。RACF 鍵リングは、例えば *keystore_type;keyring_name;racfid* です。

- *keystore_type* は、以下のいずれかの値でなければなりません。
  - **JCERACFKS** (SSL にソフトウェア暗号化が使用される場合)
  - **JCE4758RACFKS** (ハードウェア暗号化が使用される場合)
- *keyring_name* は、鍵ストアとして使用している RACF 鍵リングの名前です。
- *racfid* は、指定された鍵リングへのアクセス権限がある RACF ID です。

以下の例は、2 つの RACF 鍵リングの仕様を示しています。

- `JCERACFKS;myKeyring;kruser01`
- `JCE4758RACFKS;myKeyring;kruser01`

z/OS では、SSLKeyStoreName が RACF 鍵リングのフォーマットと一致する場合、IMS TM リソース・アダプターは指定された RACF 鍵リングをその鍵ストアとして使用します。指定された鍵ストア・タイプが **JCERACFKS** と **JCE4758RACFKS** 以外のものである場合、IMS TM リソース・アダプターは JKS 鍵ストア・ファイルの名前として指定された SSLKeyStoreName の解釈を試みます。

JKS ファイルには、KS 以外のファイル拡張子が付くことがあります。

関連資料:

1063 ページの『SSL トラストストア名 (SSLTrustStoreName)』

SSL 鍵ストアのパスワード (SSLKeyStorePassword)

このオプションのプロパティーは、TCP/IP 接続のみに適用され、SSL 鍵ストアのパスワードを指定します。SSLEnabled プロパティーは、`true` に設定する必要があります。

SSL トラストストア名 (SSLTrustStoreName)

このオプションのプロパティーは TCP/IP 接続のみに適用され、SSLEnabled プロパティーが `true` に設定されている場合にのみ適用されます。この値には、トラストストアの名前 (絶対ファイル・パスを含む) が含まれています。

z/OS では、SSL トラストストア名は、トラストストアの JKS 名または RACF 鍵リングです。 SSLKeyStoreName プロパティーと SSLTrustStoreName プロパティーの値には、同じフォーマットが使用されます。このフォーマットについては、SSLKeyStoreName プロパティーの説明を参照してください。

その他のプラットフォームの場合は、JKS トラストストア・ファイルの完全修飾パス名を指定します。

トラストストア・ファイルは、公開鍵または証明書を含む鍵データベース・ファイル (鍵ストア) です。利便性を図るために、秘密鍵もトラストストアに保管できます。 SSLKeyStoreName プロパティーは、空にすることも、トラストストア・ファイルを指すように設定することもできます。 SSLTrustStoreName または SSLTrustStorePassword のいずれかのプロパティーが空の場合、サーバー・ログに通知メッセージが生成されます。

JKS ファイルには、KS 以外のファイル拡張子が付くことがあります。

関連資料:

1062 ページの『SSL 鍵ストア名 (SSLKeyStoreName)』

SSL トラストストアのパスワード (SSLTrustStorePassword)

このオプションのプロパティーは、TCP/IP 接続のみに適用され、SSL トラストストアのパスワードを指定します。SSLEnabled プロパティーは、`true` に設定する必要があります。

ユーザー名 (userName)

このオプションの値は、アプリケーション・コンポーネントまたはコンテナーによってユーザー名が指定されていない場合に、この接続ファクトリーが作成する接続に使用するデフォルトのセキュリティー許可機能 (SAF) ユーザー名です。

IMS 相互作用仕様プロパティー

IMSInteractionSpec オブジェクトのプロパティーは、IMS との対話を記述します。

IMSInteractionSpec プロパティーの値は、`set` メソッドを使用して、アプリケーションに直接指定できます。それらの値は、アプリケーション用にコードを生成する IDE 内の J2C ウィザードに提供することもできます。

以下のリストでは、IMSInteractionSpec オブジェクトのプロパティーについて説明しています。

関連情報:

IMS TM Resource Adapter バージョン 15 Java API 仕様

代替クライアント ID (altClientID)

この入力専用プロパティーでは、代替クライアント ID の名前を指定します。この名前は、TPIPE として使用されます。この TPIPE から、共用可能永続ソケット接続で非同期出力がリトリーブされます。

このプロパティーは以下の基準でサポートされます。

- 共用可能永続ソケットを使用する TCP/IP 接続
- 非同期出力メッセージをリトリーブするための有効な対話 Verb (RESUME TPIPE 要求) は、以下のとおりです。
  - SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT
  - SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT
  - SYNC_RECEIVE_CALLOUT
- コミット・モード 0 の対話は、ローカル・オプション接続では使用できません。
- このプロパティーは、専用ソケット接続では使用できません。
- 転送名と代替クライアント ID は相互に排他的であるため、同時に指定できません。

非同期出力が使用可能 (asyncOutputAvailable)

Java アプリケーションはこの出力専用プロパティーを使用して、その Java アプリケーションが使用する接続と関連する TPIPE に関して、キューに入れられた出力があるかどうかを判別します。

専用永続ソケット接続の場合、TPIPE の名前は IMSConnectionSpec オブジェクトの clientID プロパティー内の値です。

共用可能永続ソケット接続の場合、TPIPE の名前は IMS TM リソース・アダプターによって生成されます。asyncOutputAvailable の値は、キューにメッセージが存在する場合は true です。asyncOutputAvailable プロパティーはアプリケーション・コンポーネントによって入力に設定されません。

Java アプリケーションでこのプロパティーが使用される場合、そのアプリケーションの IMSInteractionSpec オブジェクトの出力プロパティーとして、このプロパティーを公開する必要があります。

コールアウト要求タイプ (calloutRequestType)

このプロパティーは、Java アプリケーションがリトリーブするコールアウト要求メッセージのタイプを指定します。

有効な値は以下のとおりです。

- CALLOUT_REQUEST_ASYNC (デフォルト): 非同期コールアウト要求メッセージのみをリトリーブします。
- CALLOUT_REQUEST_SYNC: 同期コールアウト要求メッセージのみをリトリーブします。
- CALLOUT_REQUEST_BOTH: 同期と非同期の両方のコールアウト要求メッセージをリトリーブします。

このプロパティーは、対話 Verb が SYNC_RECEIVE_CALLOUT に設定されている場合にのみ有効です。

会話終了 (convEnded)

Java アプリケーションは、この出力専用プロパティーを使用して IMS が会話を終了しているかどうかを判別します。アプリケーション・コンポーネントは、入力には convEnded プロパティーを設定しません。

Java アプリケーションでこのプロパティーが使用される場合、そのアプリケーションの IMSInteractionSpec オブジェクトの出力プロパティーとして、このプロパティーを公開する必要があります。

会話 ID (convID)

このプロパティーは、IMS 会話を一意的に識別する 8 バイトのタイム・スタンプ (16 進数ストリング) です。

useConvID プロパティーが `true` に設定されている場合は、以下のようになります。

- IMS TM リソース・アダプターは、この会話 ID を、会話の反復ごとに出力メッセージに含めて Java クライアントに返します。
- 会話 ID は IMS によって割り当てられ、会話の最初の反復の最後に応答メッセージ・ヘッダーの中で設定されます。
- IMS Connect が共用可能永続ソケット接続から到着する会話のすべての反復を処理するために、Java クライアントが後続のすべての反復の入力内でこの会話 ID を渡す役目を果たします。

制約事項**:** このプロパティーは、対話 Verb SYNC_SEND_RECEIVE および SYNC_END_CONVERSATION でのみサポートされます。他の対話 Verb では無視されます。会話 ID の使用に適用される制約事項については、ビジネス・プロセス・コレオグラフィーの会話型トランザクションに関するトピックを参照してください。

関連概念:

1005 ページの『ビジネス・プロセス・コレオグラフィー・アプリケーション』

コミット・モード (commitMode)

このプロパティーは、IMS トランザクションに対して実行されるコミット・モード処理のタイプを示します。

- commitMode プロパティーは、interactionVerb プロパティーが SYNC_SEND_RECEIVE に設定されている場合は 0 または 1 に設定できます。
- interactionVerb プロパティーが SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT、SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT、SYNC_RECEIVE_CALLOUT、または SYNC_SEND に設定されている場合、IMS TM リソース・アダプターはコミット・モード 0 を使用します。
- interactionVerb プロパティーが SYNC_END_CONVERSATION に設定されている場合、コミット・モード 1 が必須です。

コミット・モードが 0 であり、対話に共用可能永続ソケットが使用されている場合 (接続ファクトリーの CM0Dedicated プロパティーが false に設定されている場合) は、clientID プロパティーを指定しないでください。共用可能永続ソケットで対話にコミット・モード 0 が指定されている場合、トランザクションからの出力メッセージをパージまたは転送できます。プログラム間通信からの配信されなかった 2 次出力も、パージまたは転送できます。

対話に専用永続ソケット接続が使用されている場合 (接続ファクトリーの CM0Dedicated プロパティーが true に設定されている場合)、コミット・モードは 0 でなければならず、接続に使用される IMSConnectionSpec オブジェクトのクライアント ID を指定する必要があります。コミット・モード 0 の対話に専用永続ソケットが使用される場合、配信されなかった出力メッセージは常にリカバリー可能であり、パージも転送もできません。

関連資料:

1074 ページの『同期レベル (syncLevel)』

CM0 応答 (CM0Response)

このプロパティーでは、IMS アプリケーションが IOPCB に応答しない場合、または別のトランザクションへのメッセージ通信を行う場合に、CM0 入力トランザクションについて DFS2082 メッセージを発行するかどうかを指定します。

デフォルトでは、IMS アプリケーションが IOPCB に応答しない場合、または別のトランザクションへのメッセージ通信を行う場合、CM0 トランザクションは DFS2082 通知メッセージを受け取りません。 DFS2082 メッセージを要求する場合は、このプロパティーを true に設定します。

実行タイムアウト (executionTimeout)

このプロパティーは、IMS Connect が IMS にメッセージを送信してから応答を受信するまでの最大許容時間を指定します。

- executionTimeout 値はミリ秒で表され、-2、-1、0、または 1 から 3600000 (1 時間) までの 10 進整数でなければなりません。
- このプロパティーに値 0 が設定されている場合、実際のタイムアウト値は IMS Connect によって決定されます。
- このプロパティーに値 -1 が設定されている場合、対話は時間制限なしで実行されます。
- このプロパティーに値 -2 が設定されている場合、コールアウト要求メッセージは、SINGLE オプションおよびタイマー NOWAIT を指定した OTMA RESUME TPIPE 呼び出しを使用してリトリーブされます。OTMA 保留キューにコールアウト要求メッセージが存在しない場合、IMS TM リソース・アダプターはコールアウト要求を待機せず、即時に制御を Java アプリケーションに返します。

関連資料:

997 ページの『有効な実行タイムアウト値』

1073 ページの『ソケット・タイムアウト (socketTimeout)』

PURG 呼び出しの無視 (ignorePURGCall)

このプロパティーは、コミット・モード 0 の対話に対して、IMS アプリケーションでの複数の ISRT 呼び出しおよび PURG 呼び出しを無視するかどうかを制御します。

IMS アプリケーションに複数の ISRT 呼び出しおよび PURG 呼び出しがある場合、以下のようになります。

- クライアント・アプリケーションがコミット・モード 1 の対話を使用する場合、IMS は複数の出力セグメントを持つ 1 つの応答メッセージを送信します。
- クライアント・アプリケーションでコミット・モード 0 を使用する場合は、以下のようになります。
  - ignorePURGCall プロパティーが `false` (デフォルト) に設定されている場合、IMS は、PURG 呼び出しごとに 1 つずつ、複数の出力応答メッセージを送信します。クライアント・アプリケーションは最初の出力メッセージを受け取り、残りの出力メッセージはそのクライアントの非同期保留キューに残ります。このメッセージはその後、SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT または SYNC_RECEIVE_CALLOUT 対話を使用してリトリーブできます。
  - ignorePURGCall プロパティーが `true` に設定されている場合、IMS アプリケーション内の複数の PURG 呼び出しは無視されます。出力は、複数のセグメントを持つ単一のメッセージとして返されます。

ignorePURGCall フラグは、共用可能永続ソケット接続上のコミット・モード 0 の対話での、SYNC_SEND および SYNC_SEND_RECEIVE 対話に有効です。コミット・モード 1 の対話の場合、このフラグの値に関係なく、複数の PURG 呼び出しは常に無視されます。

ignorePURGCall フラグは、SYNC_END_CONVERSATION、SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT、SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT 対話では無効であり、専用永続ソケット接続での SYNC_SEND_RECEIVE 対話の場合も無効です。これらの対話では例外がスローされます。

IMS 要求タイプ (imsRequestType)

このプロパティーは、IMS 要求のタイプを示し、IMS TM リソース・アダプターが要求からの出力を処理する方法を決定します。

次の表では、有効な IMS 要求タイプについて説明します。

表 *127. IMS* 要求タイプ

値	**IMSInteractionSpecProperties** の名前付き定数	説明
1	IMS_REQUEST_TYPE_IMS_TRANSACTION	この要求は IMS トランザクションです。 IMS によって返される通常のトランザクション出力は、アプリケーションの出力メッセージに取り込まれます。 IMS が DFS メッセージを返すと、IMS TM リソース・アダプターはその DFS メッセージを含む IMSDFSMessageException をスローします。 imsRequestType プロパティーのこの値は、メッセージ入力記述子 (MID) 名およびメッセージ出力記述子 (MOD) 名によって MFS メッセージ定義から生成されないアプリケーションに使用されます。
2	IMS_REQUEST_TYPE_IMS_COMMAND	この要求は IMS コマンドです。 IMS によって返されるコマンド出力 (DFS メッセージを含む) は、アプリケーションの出力メッセージに取り込まれます。 IMSDFSMessageException はスローされません。 imsRequestType プロパティーのこの値は、IMS コマンドをサブミットするアプリケーションに使用されます。
3	IMS_REQUEST_TYPE_MFS_TRANSACTION	この要求は、メッセージ入力記述子 (MID) 名およびメッセージ出力記述子 (MOD) 名によって MFS メッセージ定義から生成されるアプリケーションからのものです。 IMS によって返される通常のトランザクション出力、および DFS メッセージは、アプリケーションからの出力メッセージに取り込まれます。 IMSDFSMessageException はスローされません。

対話 Verb (interactionVerb)

Java アプリケーションと IMS の間で行われる対話のモードを指定します。以下の表では、IMS TM リソース・アダプターでサポートされる値について説明しています。

表 *128. Java* アプリケーションと *IMS* の間の対話のモード

値	**IMSInteractionSpecProperties** の名前付き定数	説明
0	SYNC_SEND	IMS TM リソース・アダプターは IMS Connect を介して IMS にクライアント要求を送信しますが、IMS からの応答は予期していません。SYNC_SEND 対話では、クライアントは IMS から応答を同期的に受信する必要はありません。 SYNC_SEND は共用可能永続ソケット接続と専用永続ソケット接続の両方でサポートされ、コミット・モード 0 の対話でのみ使用できます。 interactionVerb プロパティーが SYNC_SEND に設定されると、実行タイムアウトとソケット・タイムアウトの値は無視されます。 **制約事項:** IMS 要求のタイプ 2 (IMS_REQUEST_TYPE_IMS_COMMAND) は SYNC_SEND 対話では使用できず、例外が生成されます。

表 128. *Java* アプリケーションと *IMS* の間の対話のモード (続き)

値	**IMSInteractionSpecProperties** の名前付き定数	説明
1	SYNC_SEND_RECEIVE	IMS 対話を実行すると、要求が IMS に送信され、応答が同期的に受信されます。 一般的な SYNC_SEND_RECEIVE 対話には、入力レコード (IMS トランザクションの入力メッセージ) が IMS に送信され、出力レコード (IMS トランザクションの出力メッセージ) が IMS から返される、という非会話型 IMS トランザクションが含まれます。 SYNC_SEND_RECEIVE 対話は、会話型 IMS トランザクションの反復にも使用されます。会話型トランザクションではコミット・モード 1 を使用する必要があります。非会話型トランザクションは、コミット・モード 1 またはコミット・モード 0 を使用して実行できます。コミット・モード 0 を専用永続ソケットで使用する場合は、IMSConnectionSpec の clientID プロパティーの値を指定する必要があります。コミット・モード 0 を共用可能永続ソケットで使用する場合は、IMSConnectionSpec の clientID プロパティーの値を指定しないでください。
3	SYNC_END_ CONVERSATION	interactionVerb プロパティーが SYNC_END_CONVERSATION に設定された状態でアプリケーションが対話を実行すると、IMS TM リソース・アダプターは、IMS 会話型トランザクションを強制的に終了させるメッセージを送信します。 IMSInteractionSpec クラスの commitMode プロパティーと IMSConnectionSpec クラスの clientID プロパティーは、SYNC_END_CONVERSATION が対話 Verb として指定されている場合は適用されません。
4	SYNC_RECEIVE_ ASYNCOUTPUT	対話 Verb SYNC_RECEIVE_ASYNCOUTPUT は、より具体的な SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT に置き換えられました。SYNC_RECEIVE_ASYNCOUTPUT は後方互換性を確保するためにサポートされています。新しいアプリケーションでは SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT または SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT のいずれかを使用する必要があります。

表 128. *Java* アプリケーションと *IMS* の間の対話のモード (続き)

値	IMSInteractionSpecProperties の名前付き定数	説明
5	SYNC_RECEIVE_ ASYNCOUTPUT_ SINGLE_NOWAIT	対話 Verb SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT は、共用可能永続ソケット接続と専用永続ソケット接続の両方で有効です。これは、非同期出力のリトリーブに使用されます。 共用可能永続ソケット接続上の SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT 対話は元の SYNC_SEND 対話または SYNC_SEND_RECEIVE 対話と同じアプリケーション内で行われる必要があり、同じ共用可能永続接続を使用している必要があります。これは基本的に実行タイムアウトの後に行われます。 このタイプの対話では、Java クライアントは、単一のメッセージのみを受信できます。要求が行われたときに IMS OTMA 非同期保留キューにそのクライアント ID 向けのメッセージがない場合、メッセージのリトリーブはそれ以降試みられません。メッセージは返されず、SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT 対話の executionTimeout プロパティーに指定された時間が経過すると、タイムアウトが発生します。
6	SYNC_RECEIVE_ ASYNCOUTPUT_ SINGLE_WAIT	対話 Verb SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT は、非同期出力のリトリーブに使用されます。これは、共用可能永続ソケット接続と専用永続ソケット接続の両方で有効です。 共用可能永続ソケット接続上の SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT 対話は元の SYNC_SEND 対話または SYNC_SEND_RECEIVE 対話と同じアプリケーション内で行われる必要があり、同じ共用可能永続接続を使用している必要があります。これは基本的に実行タイムアウトの後に行われます。 このタイプの対話では、Java クライアントは、単一のメッセージのみを受信できます。要求が行われたときに IMS OTMA 非同期保留キューにそのクライアント ID 向けのメッセージがない場合、IMS Connect は OTMA がメッセージを返すのを待機します。IMS Connect は、SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT 対話の executionTimeout プロパティーに指定された時間だけ待機した後、例外を返します。
7	SYNC_RECEIVE_ CALLOUT	対話 Verb SYNC_RECEIVE_CALLOUT は、非同期または同期のコールアウト・メッセージのリトリーブに使用されます。 calloutRequestType プロパティーを使用することで、同期コールアウト・メッセージのみ、非同期コールアウト・メッセージのみ、同期と非同期の両方のコールアウト・メッセージの、いずれをリトリーブするかを指定します。 このタイプの対話では、要求が行われたときに指定のクライアント ID 向けのメッセージが IMS OTMA 非同期保留キューに含まれていない場合、IMS Connect は OTMA がメッセージを返すのを待機します。 IMS TM リソース・アダプターが IMS Connect と OTMA からの応答を待機する時間は、executionTimeout プロパティーの値によって異なります。

制約事項: Java EE 接続アーキテクチャー (JCA) の値 SYNC_RECEIVE (2) はサポートされません。

LTERM 名 (ltermName)

IMS アプリケーション・プログラムの入出力 PCB の LTERM フィールドの値をオーバーライドするために使用される LTERM 名です。

このプロパティーの値は、クライアント・アプリケーションが LTERM オーバーライド名を指定する必要がある場合に設定できます。この名前が IMS アプリケーション・プログラムの入出力 PCB に入れられる目的は、IMS アプリケーションがこのオーバーライド値に基づいてロジック決定を行うようにすることです。

マップ名 (mapName)

mapName フィールドには、メッセージ形式サービス (MFS) 制御ブロックの名前が含まれています。MFS は、トランザクション入出力メッセージのオンライン・フォーマットを実行する IMS のコンポーネントです。

IMS Connect は IMS OTMA を使用して IMS にアクセスするので、MFS オンライン・フォーマットは迂回されます。 mapName フィールドは、IMS TM リソース・アダプターによって、入出力メッセージが MFS メッセージ定義に基づいてフォーマットされる Java アプリケーションからの要求の処理に使用されます。mapName フィールドは Java アプリケーションが直接使用するフィールドではありません。参照目的で、mapName フィールドが IMS TM リソース・アダプターによってどのように使用されるかに関する要旨を以下に示します。

入力メッセージの場合、mapName プロパティーの値は MFS メッセージ出力記述子 (MOD) の名前です。MOD 名は、入出力 PCB 内で、デフォルトの MODNAME として IMS アプリケーション・プログラムに渡されます。このデフォルト値は、ご使用の IMS アプリケーション・プログラムで上書きできます。IMS アプリケーション・プログラムでは、入出力 PCB のこのデフォルトの MODNAME を使用することも、その値をオーバーライドして別の任意の MODNAME を使用することもできます。

一般に、出力メッセージの場合、mapName プロパティーの値は MFS メッセージ出力記述子 (MOD) の名前です。IMS アプリケーションがどのようにプログラムされているかに応じて、この MOD 名はさまざまな方法で設定できます。 IMS アプリケーション・プログラムは、トランザクション出力メッセージを入出力 PCB に挿入するときに別の MODNAME を指定できます。IMS アプリケーション・プログラムが出力メッセージで使用するための別の MOD 名を指定しない場合、入力メッセージの mapName プロパティーの値が、出力メッセージの mapName プロパティーの値として使用されます。入力メッセージに mapName 値が指定されておらず、IMS アプリケーションが、入出力 PCB 内の MODNAME フィールドに出力メッセージで使用する値を設定しない場合、IMS はデフォルトの IMS MOD 名を出力メッセージの mapName フィールドに挿入します。

非同期出力のパージ (purgeAsyncOutput)

この入力プロパティーは、IMS Connect が配信されなかった出力をパージするかどうかを決定します。

このプロパティーは、IMS 対話 Verb SYNC_SEND_RECEIVE を使用する、共用可能永続ソケット接続上の対話でのみ有効です。専用永続ソケット接続上の対話では無効です。これはコミット・モード 0 の対話のみに適用され、コミット・モード 1 の対話には適用されません。ただし、コミット・モード 1 の対話がプログラム間通信を実行する場合、作成されたプログラムはコミット・モード 0 で実行されることになるため、このプロパティーが適用されます。

purgeAsyncOutput プロパティーが共用可能永続ソケット接続上の SYNC_SEND_RECEIVE 対話に指定されていない場合、デフォルトは true であるため、以下の出力メッセージがパージされます。

- 1 次 IMS アプリケーション・プログラムによって入出力 PCB に挿入された、配信されなかった出力メッセージ
- プログラム間通信によって起動された 2 次 IMS アプリケーション・プログラムによって入出力 PCB に挿入された出力メッセージ

転送 (reRoute)

この入力プロパティーは、配信されない出力を reRouteName フィールドで指定された宛先に転送するかどうかを決定します。

reRoute プロパティーは、IMS 対話 Verb SYNC_SEND_RECEIVE を使用する、共用可能永続ソケット接続上の対話で有効です。専用永続ソケット接続上の対話では無効です。これはコミット・モード 0 の対話に適用され、コミット・モード 1 の対話には適用されません。

ただし、コミット・モード 1 の対話がプログラム間通信を実行する場合、作成されたプログラムはコミット・モード 0 で実行されることになるため、このプロパティーが適用されます。このプロパティーは、配信されなかった出力を reRouteName フィールドで指定された宛先に転送するかどうかを決定します。reRoute が true の場合、非同期出力は、生成されたクライアント ID の TPIPE でキューに入れられません。非同期出力は代わりに、reRouteName フィールドに指定された宛先でキューに入れられます。 reRoute のデフォルト値は false です。

reRoute と purgeAsyncOutput が両方とも true に設定されている場合、例外がスローされます。

転送名 (reRouteName)

このプロパティーでは、非同期出力をキューに入れる出力先の名前を指定します。

reRoute プロパティーが true に設定されている場合、指定された宛先をこのプロパティーが示します。 reRoute プロパティーが false に設定されている場合、reRouteName プロパティーは無視されます。reRoute プロパティーが `true` に設定されていて、reRouteName プロパティーに値が指定されていない場合、reRouteName プロパティーの値は以下のようになります。

- IMS Connect 構成ファイルに指定されている値
- IMS Connect 構成ファイルに値が指定されていない場合、値 HWS$DEF が使用されます。

有効な値は必ず以下の規則に従っています。

- 1 文字から 8 文字のストリングでなければなりません。有効な文字は、A から Z、0 から 9、@、#、および $ です。
- IMS TM リソース・アダプターの予約接頭部 HWS で始まっていてはなりません。
- IMS Connect のポート番号であってはなりません。

重要: 小文字を指定した場合、その文字は大文字に変更されます。

reRouteName プロパティーは、共用可能永続ソケット接続上の SYNC_SEND_RECEIVE 対話でのみ有効です。専用永続ソケット接続上の対話では無効です。

RESUME TPIPE ネットワーク・セキュリティー資格情報 (resumeTpipeNSC)

このプロパティーは、Java アプリケーションが IMS からの非同期コールアウト要求にネットワーク・セキュリティー資格情報が含まれるかどうかを示すために使用します。

resumeTpipeNSC プロパティーの値を 1 に設定すると、IMS TM リソース・アダプターは、IMS に送信される OTMA メッセージの接頭部に、ネットワーク・セキュリティー資格情報が非同期コールアウト・メッセージに含まれることを示すフラグ・バイトを設定します。

ソケット・タイムアウト (socketTimeout)

IMS TM リソース・アダプターが IMS Connect からの応答を待機する最大時間です。この時間が経過すると、ソケットが切断され、クライアント・アプリケーションに例外が返されます。

socketTimeout 値は、ミリ秒単位で表されます。この値はゼロより大きい値でなければなりません。対話にソケット・タイムアウトが指定されていない場合、または対話に 0 ミリ秒のソケット・タイムアウト値が指定されている場合、IMS TM リソース・アダプターは IMS Connect が応答するのを無期限に待機します。

関連資料:

1066 ページの『実行タイムアウト (executionTimeout)』

同期コールアウト相関トークン (syncCalloutCorrelationToken)

このプロパティーには、同期コールアウト要求を応答と関連付けるために、IMS Connect によって生成されてコールアウト要求メッセージとともに送信される相関トークンが含まれています。

管理コールアウト・プログラミング・モデル (メッセージ駆動型 Bean を使用するモデル) を使用する場合、このトークンは IMS TM リソース・アダプターによって自動的に管理されます。クライアント管理コールアウト・プログラミング・モデル (非 MDB アプリケーション) では、Java アプリケーションが相関トークンを管理し、関連付けられた getter メソッドおよび setter メソッドを使用して応答ととも

にそのトークンを返す必要があります。したがって、2 つのモデルのうちでは、管理コールアウト・プログラミング・モデルの方がシンプルで、推奨されるアプローチです。

このプロパティーは、対話 Verb が SYNC_RECEIVE_CALLOUT に設定されている場合にのみ有効です。

同期コールアウト状況コード (syncCalloutStatusCode)

Java アプリケーションが IMS アプリケーションからの同期コールアウト・メッセージに対するエラー応答メッセージを送信するとき、このプロパティーにはユーザー指定状況コードが含まれます。

有効なユーザー指定状況コードは 500 から 1000 までです。

同期レベル (syncLevel)

この入力プロパティーは、IMS TM リソース・アダプターと IMS OTMA 間の対話の同期レベルを指定します。

有効な同期レベル値は、0 (NONE) および 1 (CONFIRM) です。syncLevel プロパティーは、interactionVerb プロパティーが SYNC_SEND_RECEIVE、SYNC_SEND、および SYNC_RECEIVE_CALLOUT に設定されている場合のにみ適用されます。syncLevel プロパティーの値は会話型および非会話型の両方のアプリケーションに適用され、commitMode プロパティーと組み合わせて使用されます。

コミット・モード **1**

同期レベル 0 および 1 が有効です。0 はデフォルト値です。interactionVerb プロパティーが SYNC_SEND_RECEIVE に設定され、commitMode プロパティーが 1 に設定されている場合は、syncLevel プロパティーを設定する必要はありません。setSyncLevel(int) メソッドに 0 または 1 以外のいずれかの値が渡されると、例外がスローされます。

コミット・モード **0**

同期レベル 1 のみが有効な値です。このコミット・モードでは同期レベルを設定する必要はありません。 setSyncLevel(int) メソッドに他のいずれかの値が渡されると、例外がスローされます。

プログラム間通信によってトリガーされるプログラムは、発信側プログラムの同期レベルに関係なく、常にコミット・モード 0 として処理されます。このため、これらのプログラムからの 2 次出力は、元の対話の reRoute プロパティーおよび purgeNotDeliverable プロパティーの設定に従って、転送キューに入れられるかパージされる可能性があります。

関連資料:

1065 ページの『コミット・モード (commitMode)』

トランザクションの有効期限 (transExpiration)

実行タイムアウト値に達した場合に、トランザクションを期限切れと見なし、OTMA が処理しなくてもよいようにするかどうかを、IMS OTMA に指示します。

このプロパティーが true に設定されている場合、IMS TM リソース・アダプター・クライアント・アプリケーションは、実行タイムアウトに達した後にトランザクションを破棄できることを OTMA に示します。この機能により、OTMA は不要なメッセージを処理する必要がなくなります。

デフォルトは `false` です。

関連資料:

1066 ページの『実行タイムアウト (executionTimeout)』

関連情報:

OTMA のトランザクション有効期限サポート (IMS バージョン 15 コミュニケーションおよびコネクション)

IMS Connect のトランザクション有効期限サポート (IMS バージョン 15 コミュニケーションおよびコネクション)

会話 ID の使用 (useConvID)

このプロパティーは、IMS 会話が、その会話のすべての反復で同じ接続を使用するか、それぞれ異なる接続を使用するかを示します。

デフォルトでは、共用可能永続ソケット上における 1 つの会話の後続の反復は IMS Connect によって拒否されます (ソケットが反復間で異なっている場合)。IMS Connect が、最初の反復で使用されたソケットに関する関連情報を見つけられないためです。

このプロパティーが `true` に設定されている場合、IMS は固有の会話トークンを割り当てます。このトークンは、Java アプリケーションと IMS Connect の間で相互に受け渡しする必要があります。IMS Connect はクライアントの会話の状況を追跡しなくなります。その結果、IMS Connect コマンド VIEWHWS では、そのコマンドが IMS Connect による会話型反復の処理時に入力されない限り、クライアントの CONV 状況は表示されません。会話の反復を管理するためのこのアプローチは、IMS Connect 管理の会話状態プログラミング・モデルとは対照的に、クライアント管理の会話状態プログラミング・モデルと呼ばれます。

重要: デフォルトは、後方互換性を確保するために `false` となっています。IMS TM Resource Adapter バージョン 12 以降を使用するためにマイグレーションされているアプリケーションでは、1 つの会話型トランザクションの複数の異なる反復に共用可能ソケット接続を使用していない場合、変更は必要ありません。新規アプリケーションを開発する際は、このプロパティーを true に設定して、1 つの会話トランザクションの複数の異なる反復にそれぞれ異なる共用可能永続ソケット接続を使用できるようにしてください。

Java API 仕様

IMS TM リソース・アダプターの Java API 仕様には、Java のクラス、インターフェース、およびメソッドに関する参照情報が含まれています。

バージョン 15 Java API の資料

第 9 部 付録

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。本書の他言語版を IBM から入手できる場合があります。ただし、ご利用にはその言語版の製品もしくは製品のコピーを所有していることが必要な場合があります。

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。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒*103-8510*
東京都中央区日本橋箱崎町*19*番*21*号
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

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
US

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。

記載されている性能データとお客様事例は、例として示す目的でのみ提供されています。実際の結果は特定の構成や稼働条件によって異なります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名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で提供されるものであり、いかなる保証も提供されません。 IBM は、お客様の当該サンプル・プログラムの使用から生ずるいかなる損害に対しても一切の責任を負いません。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生的創作物にも、次のように、著作権表示を入れていただく必要があります。
© (お客様の会社名) (年).
このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。
© Copyright IBM Corp. _年を入れる_.

プログラミング・インターフェース情報

この情報では、IMS が提供するプロダクト・センシティブ・プログラミング・インターフェースとそれに関連する情報と同時に、IMS が提供する診断、修正、またはチューニング情報についても記述しています。

プロダクト・センシティブ・プログラミング・インターフェースにより、お客様のインストール済み環境で、このソフトウェア製品の診断、修正、モニター、修復、調整、またはチューニングなどの作業を実行することができます。これらのインターフェースを使用すると、IBM のソフトウェア製品の詳細設計や実装に対する依存関係が生じます。このためプロダクト・センシティブ・プログラミング・インターフェースは上記の特別な目的にだけ使用してください。詳細設計やその実現方法に依存しているので、このようなインターフェースに合わせて作成したプログラムは、新しい製品のリリース、バージョンで実行するとき、または保守サービスの結果として、変更が必要になることがあります。プロダクト・センシティブ・プログラミング・インターフェースとそれに関連する情報は、セクションやトピックの単位の場合はその冒頭で識別され、それ以外の場合は「プロダクト・センシティブ・プログラミング・インターフェース」というマーキングで識別されます。IBM では、上記の冒頭部での識別の記述、およびその記述を参照する本書内のすべての記述を、そのような記述によって示される全体コピーまたは部分コピーに含めるよう求めています。

診断、修正、チューニングの情報は、 IMS の診断、変更、またはチューニングをお客さまが行う手助けをするために提供されます。診断、修正、またはチューニング情報は、プログラミング・インターフェースとしては使用しないでください。

診断、修正、またはチューニング情報は、節またはトピックの場合はその冒頭で識別され、それ以外の場合は次のようにマーク付けされています。診断、変更、またはチューニング情報。

商標

IBM、IBM ロゴおよび ibm.com® は、世界の多くの国で登録された International Business Machines Corporation の商標です。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 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。

Linux は、Linus Torvalds の米国およびその他の国における商標です。

Microsoft、Windows、Windows NT および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

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。

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。

適用される条件

このご使用条件は、IBM Web サイトのすべてのご利用条件に追加して適用されます。

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。

商業的使用

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 は、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で提供され、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・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確認ください。

この「ソフトウェア・オファリング」は、Cookie もしくはその他のテクノロジーを使用して個人情報を収集することはありません。

この「ソフトウェア・オファリング」が Cookie およびさまざまなテクノロジーを使用してエンド・ユーザーから個人を特定できる情報を収集する機能を提供する場合、 お客様は、このような情報を収集するにあたって適用される法律、ガイドライン等を遵守する必要があります。これには、エンドユーザーへの通知や同意の要求も含まれますがそれらには限られません。

このような目的での Cookie を含む様々なテクノロジーの使用の詳細については、IBM の『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/privacy/details) を参照してください。

参考文献

この参考文献のリストには、IMS 15 ライブラリーのすべての資料が記載されています。

表題	頭字語	資料番号
IMS V15 アプリケーション・プログラミング	APG	SC27-6778
IMS V15 アプリケーション・プログラミング *API*	APR	SC27-6779
IMS V15 コマンド 第 *1* 巻: *IMS* コマンド *A-M*	CR1	SC27-6780
IMS V15 コマンド 第 *2* 巻: *IMS* コマンド *N-V*	CR2	SC27-6781
IMS V15 コマンド 第 *3* 巻: *IMS* コンポーネントおよび *z/OS* コマンド	CR3	SC27-6782
IMS V15 コミュニケーションおよびコネクション	CCG	SC27-6783
IMS V15 データベース管理	DAG	SC27-6784
IMS V15 データベース・ユーティリティー	DUR	SC27-6785
IMS Version 15 Diagnosis	DGR	GC27-6786
IMS V15 出口ルーチン	ERR	SC27-6787SC43-3856
IMS V15 インストール	INS	SC27-6788
IMS Version 15 Licensed Program Specifications	LPS	GC27-6799
IMS V15 メッセージおよびコード 第 *1* 巻: *DFS* メッセージ	MC1	GC27-6789
IMS V15 メッセージおよびコード 第 *2* 巻: *DFS* 以外メッセージ	MC2	GC27-6790
IMS V15 メッセージおよびコード 第 *3* 巻: *IMS* 異常終了コード	MC3	GC43-4469
IMS V15 メッセージおよびコード 第 *4* 巻: *IMS* コンポーネント・コード	MC4	GC43-4470
IMS V15 オペレーションおよびオートメーション	OAG	SC27-6793
IMS V15 リリース計画	RPG	GC27-6794
IMS V15 システム管理	SAG	SC27-6795
IMS V15 システム定義	SDG	GC27-6796
IMS V15 システム・プログラミング *API*	SPR	SC27-6797
IMS V15 システム・ユーティリティー	SUR	SC27-6798

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]

[カ行]

［サ行］

[タ行]

[ナ行]

[ハ行]

[マ行]

[ヤ行]

[ラ行]

[ワ行]

[数字]

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

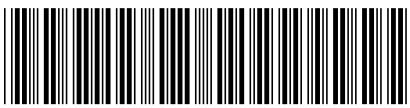
X

Z

プログラム番号: 5635-A06
5655-DS5
5655-TM4

Printed in Japan

SC43-4281-00



日本アイ・ビー・エム株式会社
〒103-8510 東京都中央区日本橋箱崎町19-21

Spine information:

IMS バージョン **15.1.0**

アプリケーション・プログラミング