IBM z/VSE
バージョン **6** リリース **2**

e-business Connectors ユーザーズ・ガイド

SC43-4402-00

IBM z/VSE
バージョン **6** リリース **2**

e-business Connectors ユーザーズ・ガイド

SC43-4402-00

お願い

注: 本書および本書で紹介する製品をご使用になる前に、 579 ページの『特記事項』に記載されている情報をお読みください。

本書は、IBM z/Virtual Storage Extended (z/VSE) バージョン 6 リリース 2 (プログラム番号 5686-VS6)、および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

本書は SA88-4412-02 の改訂版です。

資料のご注文方法については、http://www.ibm.com/jp/manuals の「ご注文について」をご覧ください。 (URL は、変更になる場合があります)

また、FAX により、またはインターネット経由で送付することもできます。

Internet: s390id@de.ibm.com
FAX (Germany): 07031-16-3456
FAX (other countries): (+49)+7031-16-3456

なお、お寄せいただいたご意見は、弊社にて随時利用させていただきますので、ご承諾のうえご記入くださいますようお願い申し上げます。

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

原典： SC34-2693-00
IBM z/VSE
Version 6 Release 2
e-business Connectors, User's Guide

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

目次

図

表

本書について

本書では、z/VSE e-business コネクターを使用して、z/VSE ホスト・ベースでプログラムおよびデータにアクセスする e-business アプリケーションを開発する方法について説明します。

本書の対象読者

本書は、z/VSE プログラムを追加インストールするシステム・プログラマー、および Java™ オブジェクト指向プログラム言語を理解しているアプリケーション・プログラマーを対象としています。

本書の使用法

本書は、以下のように 4 つのパートに分かれています。

- 第 **1** 部 では、z/VSE e-business コネクターの概要、および e-business ソリューションとして可能な基本構成を記載します。
- 第 **2** 部 では、z/VSE システム内で e-business 接続を確立するために実行できるインストールおよびカスタマイズのアクティビティーについて説明します。
- 第 **3** 部 では、Java ベース・コネクターと Db2 ベース・コネクターを使用して独自の e-business アプリケーションを開発する方法に関する理論的および実践的な情報を記載します。

本書の追加情報の入手先

以下に、参考になる IBM® の資料および他のベンダーの資料をリストします。 ここにリストした IBM Redbooks は、最新の情報ではない場合がありますが、記載されている技術分野の内容について記述された時点 での最新の情報になります。

- *IBM z/VSE* 計画、SC43-2937
- *IBM z/VSE TCP/IP* サポート、SC43-2945
- *CICS Transaction Server for z/VSE* 拡張ガイド、SC34-2685
- *WebSphere MQ for z/VSE® System Management Guide*、GC34-6981
- *WebSphere V5 for Linux on zSeries Connectivity Handbook (Redbook)*、SG24-7042
- *e-business Solutions for z/VSE (Redbook)*、SG24-5662
- *WebSphere Application Servers: Standard and Advanced Editions (Redbook)*、SG24-5460
- *Security on IBM z/VSE (Redbook)*、SG24-7691

VSE コネクター・クライアント で提供されるオンライン・ドキュメンテーションには、参考として役立つインターネット・サイトおよびオンライン資料のリストが記載されています。記述については、 30 ページの『オンライン・ドキュメンテーション・オプションの使用』を参照してください。

さらに、以下の情報も役立ちます。

z/VSE ホーム・ページ

z/VSE は、WWW 上にホーム・ページがあります。ホーム・ページでは、VSE 関連プロダクトやサービス、z/VSE の新機能、およびその他の VSE ユーザーにとって興味深い項目に関する最新情報が記載されています。

z/VSE ホーム・ページは、次のアドレスで見ることができます。

http://www.ibm.com/systems/z/os/zvse/

次のサイトで VSE ユーザーの例 (ZIP 形式) を検索することもできます。

http://www.ibm.com/systems/z/os/zvse/downloads/samples.html

変更の要約

z/VSE 6.2 の一般出荷版で使用可能になった機能強化は以下のとおりです。

- *Connectivity Systems Incorporated* が提供する *TCP/IP for z/VSE* 製品 (現在 IPv4 のみをサポート) は、ライブラリー PRD2.TCPIPC にインストールされ、PRD1.BASE にはインストールされなくなりました。このため、TCP/IP for z/VSE を使用するバッチ・ジョブは、そのライブラリー検索チェーンの最初のライブラリーとして PRD2.TCPIPC を指定する必要があります。いくつかの例については、 571 ページの『バッチで稼働する VSE スクリプト・クライアントの使用』を参照してください。
- 509 ページの『第 25 章 REST を使用したプログラム間通信』
- 533 ページの『第 26 章 z/VSE MQ Client Trigger Monitor を非同期プログラム間通信に使用』

以下の機能強化は、z/VSE 5.1 のサービス・アップグレードで使用可能になっています。

- データベース呼び出しレベル・インターフェース は、z/VSE アプリケーションが適切なデータベース・サーバーでリレーショナル・データベース (IBM DB2®、Oracle、Microsoft SQL Server、MySQL など) にアクセスできるようにするデータベース・コネクターです。 85 ページの『第 9 章 データベース呼び出しレベル・インターフェースのインストール』および 321 ページの『第 22 章 データにアクセスするための データベース呼び出しレベル・インターフェース の使用』を参照してください。
- データベース呼び出しレベル・インターフェース では接続プーリング がサポートされます。 86 ページの『接続プーリングの概要』および 91 ページの『接続プーリング・マネージャーの構成と開始/停止』を参照してください。

注: z/VSE 6.2 で導入されたすべての 項目の概要については、「IBM z/VSE リリース・ガイド」(SC43-2938) を参照してください。

第 1 部 概要

第 1 章 z/VSE における e-business の概要

お客様のインターネットおよび e-business 要件を満たすために、 IBM は *Application Framework for e-business* を開発しました。 このフレームワークは、既存の情報資産への投資を保護できるように開発され、新たに e-business の好機を活かすことを可能にします。 Application Framework for e-business は、次の 3 つの論理層で構成されます。

1. クライアント。通常は、標準 Web ブラウザーをインストールしたワークステーションです。 しかし、携帯電話または携帯情報端末 (PDA) も可能です。
2. Web アプリケーション・サーバー。クライアントからの要求を処理し、ビジネス・ロジックおよびデータへのアクセスを制御する「ハブ」です。 ロジックまたはデータが別のシステムに保管されている場合、Web アプリケーション・サーバーはコネクターを使用してアクセスします。 また、Web アプリケーション・サーバーは、静的および動的コンテンツを統合してから Web ページをクライアントに戻します。
3. コネクター。ビジネス・ロジックおよびデータなどの外部サービスへのアクセスを可能にします。 z/VSE で提供されるコネクターは今回新しく導入されました。

z/VSE e-business コネクター は、Application Framework for e-business をサポートし、お客様のコア・アプリケーション を *e-business* アプリケーション に拡張できるリソースを用意しています。 これにより、既存のコア・アプリケーションへの投資を保護および活用することができます。

- コア・アプリケーション (一般に CICS®、COBOL、VSAM) は通常、z/VSE ホスト上で実行され、会社のオペレーションに不可欠で、今後長期間にわたる実動を期待されるものであり、これまでのリソースへの膨大な投資を指しています。
- *e-business* アプリケーション は一般的に、TCP/IP、HTML、XML、Secure Sockets Layer (SSL)/Transport Layer Security (TLS)、Secure Electronic Transaction (SET) などの共通規格に基づいており、Java で作成されたサーバーとクライアントの両コードを組み込み、リレーショナル・データにローカルまたはリモートでアクセスし、標準 Web ブラウザーを通じてエンド・ユーザーにインターフェースで連結します。

z/VSE e-business コネクター を使用した接続の可能性

現在の z/VSE e-business コネクター は、以下のようになっています。

- *Java* ベース・コネクター。これにより、VSE コネクター・クライアントのクラス・ライブラリーを使用して、すべての VSE ファイル・システムへのアクセス、ジョブのサブミット、コンソール・コマンドの発行などを行う Java アプリケーションを実装できます。 27 ページの『Java ベース・コネクター の概要』で紹介されています。
- *Db2* ベース・コネクター。これにより、物理/論理中間層で DB2 Connect™ を使用し、z/VSE ホスト上で Db2 ストアード・プロシージャーを呼び出すことで

VSAM および DL/I データにアクセスする、Java アプリケーションを実装できます。 95 ページの『Db2 ベース・コネクター の概要』で紹介されています。

- *VSE* スクリプト・コネクター。これにより、*VSE* スクリプト言語 を使用して作成されたステートメントを含む VSE スクリプト (バッチ) ファイルを使用して、z/VSE ホストからデータを取得できます。 45 ページの『VSE スクリプト・コネクター の概要』で紹介されています。
- *VSAM* リダイレクター・コネクター。これは VSAM データ・セットに対する要求を処理します。次いでその要求を同期的に (VSAM リダイレクター・クライアントを介して) リダイレクトするか、または VSAM データ・セットに加えられた変更をさらに処理するために、z/VSE ホストに (VSAM 取り込み 出口を介して) 保管します。 53 ページの『VSAM リダイレクター・コネクター の概要』で紹介されています。
- データベース呼び出しレベル・インターフェース。これにより、z/VSE 以外のプラットフォーム上で実行するデータベース・サーバー (IBM Db2、Oracle、Microsoft SQL Server、MySQL など) を選択できます。 85 ページの『データベース呼び出しレベル・インターフェース の概要』で紹介されています。
- CICS Web サポートに基づいて CICS 内で稼働する SOAP サーバーおよび SOAP クライアントを提供する、*Simple Object Access Protocol (省略形 SOAP)* 接続。 SOAP サーバーを使用すれば、CICS プログラムとして実装された Web サービスを、あらゆる種類の Web サービス・クライアント (Apache、AXIS、Microsoft .Net、C# など) から呼び出すことができます。 471 ページの『Web サービスおよび SOAP での z/VSE サポートの概要』で紹介されています。
- CICS Web サポートに基づいて CICS 内で稼働する REST サーバーおよび REST クライアントを提供する、*Representational State Transfer (省略形 REST)* 接続。 REST サーバーを使用すれば、CICS プログラムとして実装された RESTful Web サービスを、あらゆる種類の RESTful Web サービス・クライアントから呼び出すことができます。 509 ページの『REST のための z/VSE サポートの概要』で紹介されています。
- WebSphere MQ Client に基づく非同期通信を提供する、*MQ Client Trigger Monitor*。 533 ページの『第 26 章 z/VSE MQ Client Trigger Monitor を非同期プログラム間通信に使用』で紹介されています。

5 ページの図 1 は、次のものの間に確立できる接続を示します。

- Web クライアント/中間層
- z/VSE ホスト

注:

1. 中間層は、「物理」中間層 (IBM System p プロセッサーなど) または「論理」中間層 (Z メインフレーム上の *Linux on z Systems*) のいずれかです。
2. Java ベース・コネクターのコンポーネントは、陰影で示します。
3. Db2 ベース・コネクターは、物理/論理中間層上の Db2 Connectと z/VSE ホスト上の Db2 Server for VSE の間の接続を使用します。

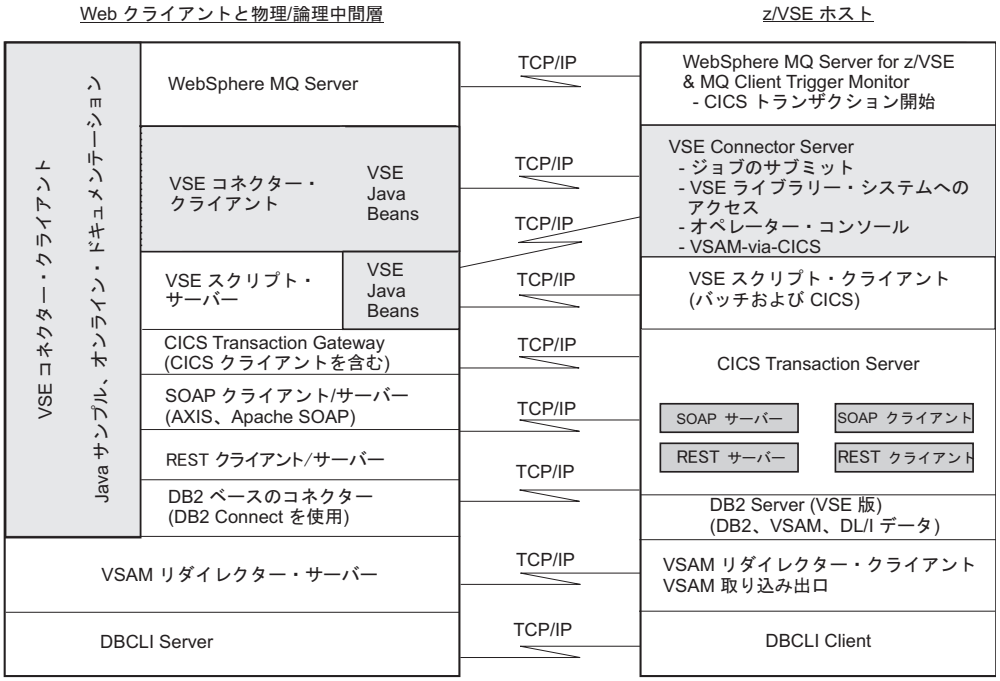


図 *1. z/VSE* 下における接続可能性の概説

CICS 接続の概要

2 層環境における CICS 接続では、CICS Web サポートおよび 3270 ブリッジ (CICS Transaction Server for z/VSE に付属の機能) を使用して CICS アプリケーションにアクセスできるようにします。

3 層環境 における CICS 接続により、以下のことが可能です。

- Java ゲートウェイ・アプリケーション (通常は物理/論理中間層に保管) は、CICS Universal Clientが提供する ECI (External Call Interface) または EPI (External Presentation Interface) を通じて CICS TS で実行中の CICS アプリケーションと通信します。
  - *ECI* インターフェース によって、非 CICS クライアント・アプリケーションは CICS プログラムをサブルーチンとして同期または非同期で呼び出すことができます。
  - *EPI* インターフェース によって、非 CICS クライアント・アプリケーションは論理 3270 端末として動作し、CICS 3270 アプリケーションを制御することができます。

  CICS Universal Clientは、APPC プロトコルを介して CICS TS と通信します。
- Java ゲートウェイ・アプリケーションと Java アプリケーション (アプレットまたはサーブレット) の間の通信に CICS Java クラス・ライブラリーを使用します。 CICS Java クラス・ライブラリーには、アプリケーション・プログラミング・インターフェース (API) を提供する以下のクラスも含まれます。

  - Java プログラムは、*JavaGateway* クラスを使用してゲートウェイ・プロセスとの通信を確立できます。このクラスは Java のソケット・プロトコルを使用します。
  - Java プログラムは、以下を使用できます。
    - *ECIRequest* クラス。ゲートウェイに流される **ECI** コールを指定します。
    - *EPIRequest* クラス。ゲートウェイに流される **EPI** コールを指定します。
- Terminal Servlet を介して CICS Transaction Server for z/VSE で実行される 3270 CICS アプリケーションのエミュレーターとして Web ブラウザーを使用します。
- 既存の CICS 3270 アプリケーション用の Java フロントエンドをプログラミングを行わずに作成する際に Java EPI Beans のセットを使用します。
- Simple Object Access Protocol (省略形は SOAP) を使用して、インターネットを介して CICS プログラムおよびその他のモジュール間で情報を送受信します。SOAP の詳細については、 471 ページの『第 24 章 SOAP を使用したプログラム間通信』を参照してください。
- Representational State Transfer (省略形は REST) を使用して、インターネットを介して CICS プログラムおよびその他のモジュール間で情報を送受信します。REST の詳細については、 509 ページの『第 25 章 REST を使用したプログラム間通信』を参照してください。

WebSphere MQ 接続の概要

WebSphere MQ 接続 によって以下が可能になります。

- z/VSE プログラムは、リモート WebSphere MQ Server 上の WebSphere® MQ キューにアクセスできます。 WebSphere MQ Client Trigger Monitor が提供するトリガー機能を利用することで、z/VSE ホスト上の CICS データおよびリソースにアクセスする CICS アプリケーションを開始できます。
- Web クライアントは、単に情報を提供して受信するのではなく、トランザクションに参加 します。

3 層環境における WebSphere MQ 接続 について詳しくは、 19 ページの表 2 を参照してください。

IBM WebSphere Application Server の概要

11 ページの図 3 に示すように、WebSphere Application Server を *3* 層 Application Framework for e-business 環境の物理/論理中間層にインプリメントします。 WebSphere Application Server は、Web サーバーと併用されます (IBM HTTP Server または Apache サーバーと同様)。

WebSphere Application Server は、Enterprise Java Beans (EJBs)、eXtensible Markup Language (XML)、および Common Object Request Broker Architecture (CORBA) などの業界標準との完全な互換性があり、e-business アプリケーションをインプリメントできる確実なフレームワークを提供します。

WebSphere Application Server を使用して、物理/論理中間層用に (Web 開発ツールを使用する) アプリケーションを作成し、z/VSE ホストに保管されているデータ

およびプログラム (CICS、Db2 など) にアクセスできます。これらのアプリケーションは、Java およびインターネット・テクノロジーを完全に活用して作成できます。

3 層環境内で WebSphere Application Server が使用される場面の概説については、 11 ページの図 3 を参照してください。

WebSphere Application Server について詳しくは、次のインターネット・アドレスも参考にしてください。

http://www.ibm.com/software/products/en/appserv-was

第 2 章 2 層および 3 層環境の概要

Web クライアントと、z/VSE ホストに保管されているプログラムおよびデータとの通信に 2 層と 3 層環境のいずれか、または両方を選択できます。 これらの環境については、 10 ページの図 2 および 11 ページの図 3 でそれぞれ解説されています。

- 2 層環境では、Web クライアントと z/VSE ホストは相互に直接通信を行います。
- 3 層環境では、Web クライアントまたは非 Java クライアントと z/VSE ホストは、中間層 と呼ばれる中間層を介して相互に通信を行います。この中間層は、以下のいずれかです。
  - 「物理」中間層 (Linux、AIX®、または Windows を実行しているプロセッサー)。
  - 「論理」中間層 (Z メインフレーム上の *Linux on z Systems*)。

注:

1. 2 層環境は、以下の要件を満たしていないため、Java プログラムの開発における標準的な環境ではありません。
   - IBM *Application Framework for e-business* のパーツである (例えば、中間層にある IBM WebSphere Application Server を使用しません)。
   - IBM Application Framework for e-business が提供する最新のセキュリティー・サービス (ファイアウォールなど) によって保護されている。

   2 層環境は、一般的にイントラネット・ソリューションにのみ適しています。
2. Java ベース・コネクターは通常、3 層環境で使用されますが、Db2 ベース・コネクターは 3 層環境でのみ 使用できます。 3 層環境には WebSphere Application Server がインストールされている必要があります。

このトピックには以下が含まれます。

2 層環境の概要

次の 2 層環境は 10 ページの図 2 のようになります。

1. VSE コネクター・クライアント・クラス・ライブラリーは、VSE コネクター・サーバーと通信する、Web クライアントで実行されている各 Java プログラムからアクセス可能でなければなりません。これは、ファイル **VSEConnector.jar** (VSE Java Beans を含む) を、Java プログラムが実行される各 Web クライアントにコピーすることによって可能です。
2. VSE Java Beans は、Web クライアントで実行される Java プログラムと z/VSE ホストで実行される VSE コネクター・サーバー の間で接続を確立するのに使用されます。 そのとき、Web クライアントで実行される Java アプリケ

ーションまたはアプレットは標準 Web ブラウザーを使用して、z/VSE ホストで実行中の VSE HTTP Server および VSE コネクター・サーバー と直接通信できます。 作業が完了すると、VSE コネクター・サーバー から Web クライアントへ返信が送られます。

3. VSAM リダイレクター・コネクターを使用して、Java に対応したプラットフォームに VSAM 要求をリダイレクトすることができます。また、データベース呼び出しレベル・インターフェースを介して、z/VSE アプリケーションで、適切なデータベース・サーバーにあるリレーショナル・データベースにアクセスすることもできます。

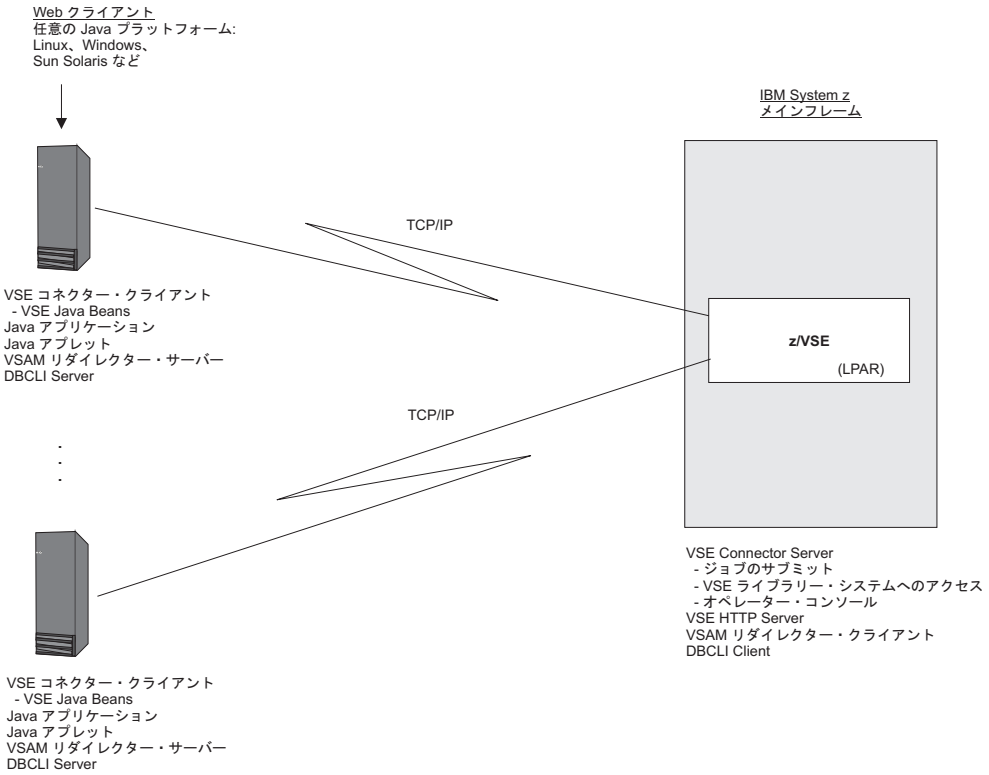


図 2. 2 層環境および使用可能なプログラムの概要

2 層インプリメンテーションの例としては、VSE コネクター・サーバー と直接通信して POWER® データにアクセスする Java アプリケーションがあります。

2 層環境でのアプレットの使用方法の詳細については、 208 ページの図 102 を参照してください。

3 層環境の概要

11 ページの図 3 に示すような 3 層環境では、*WebSphere Application Server* は中央の「ハブ」になります。以下の OS の下で稼働します。

- 物理中間層で稼働する Linux、AIX、または Windows
- メインフレーム上の *Linux on z Systems* (論理 中間層)

1. 以下のいずれかとなります。

- Web クライアントは、標準 Web ブラウザーを使用して、物理/論理中間層にある WebSphere Application Server をベースとしてアプリケーションと通信するか、または
- 非 Java クライアント (例えば、Windows 下で実行されるスプレッドシート・アプリケーション) は、VSE スクリプトを使用して、物理/論理中間層で実行される VSE スクリプト・サーバーと通信します。

2. WebSphere アプリケーション (サーブレット、EJB、または VSE スクリプト・サーバー など) は、VSE Java Beans を使用して、VSE データにアクセスしたり、z/VSE の下で実行されるアプリケーションを始動したりします。
3. 応答は、VSE コネクター・サーバー から、*WebSphere Application Server* の下で実行される (物理/論理中間層の) アプリケーションへ送信されます。アクティビティーが完了すると、アプリケーションはそのデータとその他の情報をパッケージして、Web クライアントまたは非 Java クライアントへ返信を戻します。

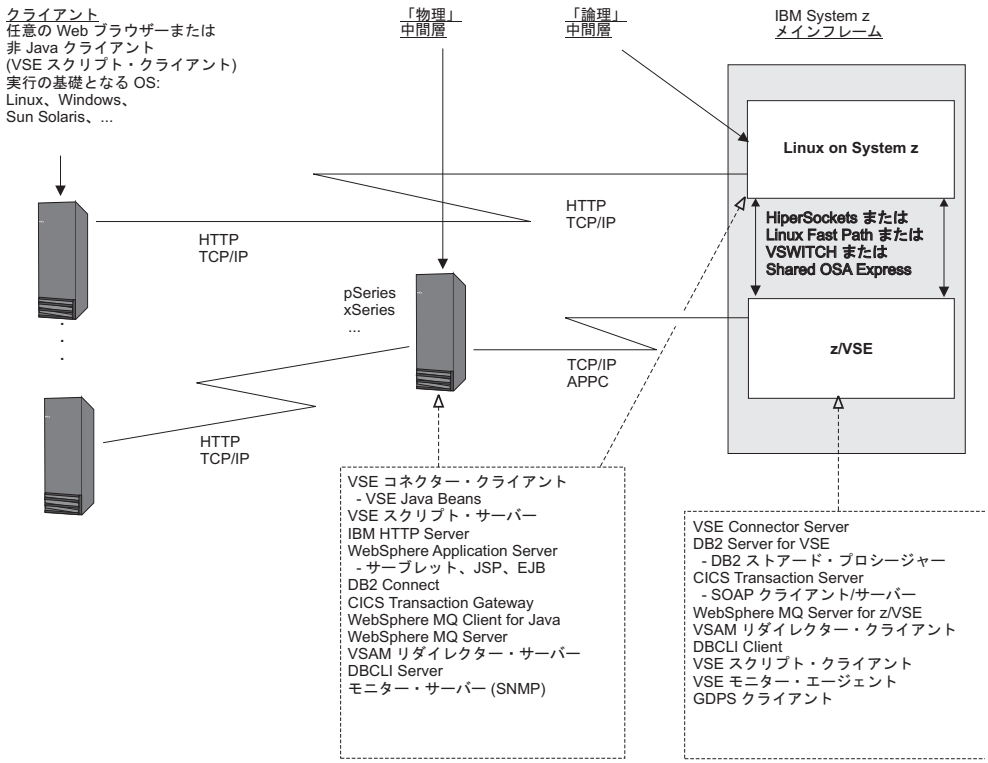


図 *3. 3* 層環境および使用可能なプログラムの概要

注**:** 本書では、3 層環境で使用されるファイアウォール を別の層とは見なしません。

3 層インプリメンテーションの例としては、VSAM データへアクセスするために Web アプリケーション・サーバー (WebSphere) で実行される Java サーブレットがあります。このサーブレットによって、必要なセキュリティー層を使用して正規のお客様がインターネットを介してオーダーを入力することができます。

3 層インプリメンテーションのもう一つの例 (非 **Java**) としては、Windows 下で実行される Microsoft Excel アプリケーション (VSE スクリプト・クライアント)

があります。ここでは、物理/論理中間層で実行中の VSE スクリプト・サーバーを介して、z/VSE ホストに保管されている VSAM データを取得します。VSE スクリプト・コネクターは、VSE スクリプト・クライアントと VSE スクリプト・サーバーの両方を指す場合に使用される用語です。

コネクターの詳細の入手先

詳細情報については、以下の該当する箇所を参照してください。

- 使用できるアプレットについては、 207 ページの『第 17 章 Java アプレットによるデータへのアクセス』 を参照してください。
- 使用できるサーブレットについては、 253 ページの『第 18 章 Java サーブレットによるデータへのアクセス』 を参照してください。
- 使用できる Java Server Pages (JSPs) については、 271 ページの『第 19 章 Java Server Pages によるデータへのアクセス』 を参照してください。
- Enterprise Java Beans (EJB) の使用方法については、 275 ページの『第 20 章 EJB を使用したデータの表現』を参照してください。
- VSE/VSAM データへのアクセスに使用できる Db2 ストアード・プロシージャーについては、 451 ページの『Db2 ストアード・プロシージャーによる VSAM データへのアクセス』 を参照してください。
- DL/I データへのアクセスに使用できる Db2 ストアード・プロシージャーについては、 459 ページの『Db2 ストアード・プロシージャーによる DL/I データへのアクセス』 を参照してください。
- VSAM データへの要求のリダイレクトに使用できる VSAM リダイレクター・コネクターについては、 53 ページの『第 8 章 VSAM リダイレクター・コネクターのインストール』 を参照してください。
- データベース呼び出しレベル・インターフェースを使用して、z/VSE アプリケーションが適切なデータベース・サーバー上でリレーショナル・データベースにアクセスできるようにする方法については、 85 ページの『第 9 章 データベース呼び出しレベル・インターフェースのインストール』を参照してください。
- Simple Object Access Protocol (省略形は SOAP) を使用して、インターネットを介して CICS プログラムおよびその他のモジュール間で情報を送受信する方法については、 471 ページの『第 24 章 SOAP を使用したプログラム間通信』を参照してください。
- Representational State Transfer (省略形は REST) を使用すれば、インターネットを介して CICS プログラムおよびその他のモジュール間で情報を送受信できます。 REST の詳細については、 509 ページの『第 25 章 REST を使用したプログラム間通信』を参照してください。
- z/VSE ホストに保管されている関数およびデータへの非 Java アクセスに使用できる VSE スクリプト・コネクターについては、 551 ページの『第 27 章 非 Java アクセスのための VSE スクリプト・コネクターの使用』を参照してください。
- VSE Monitoring Agentを使用すれば、SNMP バージョン 1 「トラップ」をバッチ・ジョブ、COBOL、C、および CICS プログラムから SNMP モニターに送信できます。 131 ページの『第 13 章 VSE モニター・エージェントを使用してデータを収集』 を参照してください。

- GDPS クライアントを使用すれば、可用性データを z/VSE システムから収集して、z/OS の下で稼働する中央 *GDPS® K-System* にそのデータを送信できます。 149 ページの『第 14 章 高可用性を実現するために GDPS サポートを使用』を参照してください。

第 2 部 インストールおよびカスタマイズ

第 3 章 必要な接続の選択

このトピックでは、2 層および 3 層の環境で、Java ベース・コネクター、Db2 ベース・コネクター、VSAM リダイレクター・コネクター、VSE スクリプト・コネクター、CICS (SOAP サーバーと SOAP クライアント、REST サーバーと REST クライアントを含む)、および WebSphere MQ Client Trigger Monitor を使用して確立できる接続のタイプについて説明します。

このトピックに含まれるのは次のとおりです。

- 『2 層環境における接続の可能性』
- 18 ページの『3 層環境における接続の可能性』

2 層環境における接続の可能性

2 層環境では、以下のものを使用することができます。

- Java ベース・コネクター
- CICS *Web* サポート・フィーチャー
- VSAM リダイレクター・コネクター
- データベース呼び出しレベル・インターフェース

上記の機能の実行について、表 1 で説明します。

表 *1. 2* 層環境における接続の可能性

接続	**Java** アプリケーション	サーブレット **/ Java Server Pages**	アプレット	**Enterprise Java Beans**
Java ベース・コネクター	VSE Beans クラス・ライブラリーを使用して、すべての VSE ファイル・システムへのアクセス (下記の注を参照)、ジョブのサブミット、コンソール・コマンドの発行などを行う Java アプリケーションをクライアントにインプリメントします。	可能性なし	VSE から Web ブラウザーへアプレットをダウンロードします。 そこで、VSE Beans クラス・ライブラリーを使用して、すべての VSE ファイル・システムへのアクセス (下記の注を参照)、ジョブのサブミット、コンソール・コマンドの発行などを行います。 VSE Beans クラスは、アプレット・コードと一緒に同じ JAR ファイルに組み込むことができます。	可能性なし

表 *1. 2* 層環境における接続の可能性 *(*続き*)*

接続	**Java** アプリケーション	サーブレット **/ Java Server Pages**	アプレット	**Enterprise Java Beans**
CICS	CICS Transaction Server for z/VSE が提供する *Web* サポート・フィーチャーを使用して、Web ブラウザーから直接 CICS トランザクションにアクセスします。 CICS トランザクションが必要な Web ページを生成し、そのページは Web ブラウザーで表示されます。 ただし、z/VSE 環境での Web サポート・フィーチャーの使用は本書の対象外であるため、詳しくは、「*CICS Transaction Server for VSE/ESA* インターネット・ガイド」(SC88-8667) を参照してください。			
VSE Web サービスおよび SOAP	SOAP は、業界全体での標準 XML ベース・プロトコルで、アプリケーションが HTTP を介してインターネット上で情報を交換できるようにするものです。 SOAP は、XML と HTTP の両方の利点を 1 つに合わせた標準アプリケーション・プロトコルです。 それにより、さまざまなプラットフォーム間で情報を送受信できます。 z/VSE が SOAP プロトコルをサポートすることによって、ユーザーは Web サービスをインプリメントできます。z/VSE の SOAP エンジンは、CICS Web サポートに基づいて CICS 内で稼働する SOAP サーバーおよび SOAP クライアントを提供します。サーバーで、CICS プログラムとしてインプリメントされた Web サービスを、あらゆる種類の Web サービス・クライアント (Apache、AXIS、Microsoft .Net、C#、など) から呼び出すことができます。 クライアントは、いずれの Web サービスも呼び出せます (例: WebSphere)。			
RESTful Web サービス	Representational State Transfer (REST) は、Web サービスのガイドラインとベスト・プラクティスから構成される、ソフトウェア・アーキテクチャー・スタイルです。 REST は、SOAP や WSDL ベースの Web サービスに対する最もシンプルな代替として、Web で幅広い支持を受けています。 RESTful システムは一般的に、Hypertext Transfer Protocol (HTTP) を介し、Web ブラウザーで使用されるものと同じ HTTP verb (GET、POST、PUT、 DELETE など) を使用して通信を行います。 z/VSE の REST エンジンは、CICS Web サポートに基づいて CICS 内で稼働する REST サーバーおよび REST クライアントを提供します。 REST サーバーを使用すれば、CICS プログラムとして実装された RESTful Web サービスを、あらゆる種類の RESTful Web サービス・クライアントから呼び出すことができます。 509 ページの『REST のための z/VSE サポートの概要』で紹介されています。			
VSAM リダイレクター・コネクター	VSAM リダイレクター・コネクターは、VSAM データ・セットへの要求を処理し、以下のいずれかを行います。 • 要求を同期にリダイレクト (VSAM リダイレクター・クライアントを介して)。 • その先の処理のために、z/VSE ホストに VSAM データ・セットへの変更を保管 (VSAM 取り込み 出口を介して)。 ソース・プログラムを変更する必要はありません。			
データベース呼び出しレベル・インターフェース	データベース呼び出しレベル・インターフェースを使用して、z/VSE アプリケーションで、任意の適切なデータベース・サーバー上のリレーショナル・データベース (IBM Db2、Oracle、Microsoft SQL Server、MySQL など) にアクセスできます。			

注**:** *VSE* ファイル・システム という用語には、VSE/VSAM、VSE/POWER、VSE/Librarian、および VSE/ICCF が含まれます。

3 層環境における接続の可能性

3 層環境では、以下のものを使用することができます。

- Java ベース・コネクター
- Db2 ベース・コネクター
- VSE スクリプト・コネクター

- VSAM リダイレクター・コネクター
- データベース呼び出しレベル・インターフェース
- CICS 接続、Simple Object Access Protocol (省略形 SOAP) および Representational State Transfer (省略形 REST) を含む
- WebSphere MQ 接続

上記の機能の実行について、表 2 で説明します。

表 2. 3 層環境における接続の可能性

接続	**Java** アプリケーション	サーブレット / **Java Server Pages**	アプレット	**Enterprise Java Beans**
Java ベース・コネクター	適用外。	VSE Beans クラス・ライブラリーを使用するサーブレットをインプリメントして、z/VSE ホスト・データにアクセスし、HTML ページを通じてデータを表示します。 詳しくは、(30 ページの『オンライン・ドキュメンテーション・オプションの使用』 ページに記載する) オンライン・ドキュメンテーションを参照してください。	中間層の Web サーバーからアプレットをダウンロードし、*VSEAppletServer* を使用して (説明については、 211 ページの『VSEAppletServer の使用方法』 ページを参照)、リモート z/VSE ホストに接続し、VSE ファイル・システムにアクセスします (この表の最後にある注を参照)。 詳しくは、(30 ページの『オンライン・ドキュメンテーション・オプションの使用』 ページに記載する) オンライン・ドキュメンテーションを参照してください。	VSE Java Beans クラス・ライブラリーを使用して z/VSE ホストと通信し、z/VSE ホスト上のデータベースにアクセスする EJB を作成します。通常、これは Db2 データまたは VSE/VSAM データのいずれかになります。詳しくは、(30 ページの『オンライン・ドキュメンテーション・オプションの使用』 ページに記載する) オンライン・ドキュメンテーションを参照してください。
Db2 ベース・コネクター	Web クライアントに Java アプリケーションをインプリメントします。これは、物理/論理中間層で Db2 Connect を使用して、z/VSE ホスト上で Db2 ストアード・プロシージャーを呼び出すことで VSAM および DL/I データにアクセスするものです。	Db2 Connect を使用して z/VSE ホスト上で Db2 ストアード・プロシージャーを呼び出すことで VSAM および DL/I データにアクセスする、サーブレットまたは JSP を作成します。このサーブレットまたは JSP には、任意の Web クライアントから WebSphere を通じてアクセスできます。	物理/論理中間層の Web サーバーからアプレットをダウンロードします。このアプレットは、物理/論理中間層にある Db2 Connect に接続します。これは、z/VSE ホスト上で Db2 ストアード・プロシージャーを呼び出すことで VSAM および DL/I データにアクセスします。	Db2 Connect を使用して z/VSE ホスト上で Db2 ストアード・プロシージャーを呼び出すことで VSAM および DL/I データにアクセスする EJB を作成します。

表 2. 3 層環境における接続の可能性 (続き)

接続	**Java** アプリケーション	サーブレット / **Java Server Pages**	アプレット	**Enterprise Java Beans**
VSE スクリプト・コネクター	非 Java プログラムをインプリメントすることで、VSE スクリプトを呼び出し、VSE 関数およびデータにアクセスします。 例えば、Lotus® 1-2-3 などのオフィス製品内に Visual Basic™ スクリプトを記述して、スプレッドシートまたは文書に VSE データを組み込みます。	適用外。	適用外。	適用外。
VSAM リダイレクター・コネクター	VSAM リダイレクター・コネクターは、VSAM データ・セットへの要求を処理し、以下のいずれかを行います。 • 要求を同期にリダイレクト (VSAM リダイレクター・クライアントを介して)。 • その先の処理のために、z/VSE ホストに VSAM データ・セットへの変更を保管 (VSAM 取り込み 出口を介して)。 ソース・プログラムを変更する必要はありません。	適用外。	適用外。	適用外。
データベース呼び出しレベル・インターフェース	データベース呼び出しレベル・インターフェースを使用して、z/VSE アプリケーションで、任意の適切なデータベース・サーバー上のリレーショナル・データベース (IBM Db2、Oracle、Microsoft SQL Server、MySQL など) にアクセスできます。	適用外。	適用外。	適用外。

表 2. 3 層環境における接続の可能性 (続き)

接続	**Java** アプリケーション	サーブレット / **Java Server Pages**	アプレット	**Enterprise Java Beans**
CICS	Web クライアントに Java アプリケーションをインプリメントします。これは、物理/論理中間層にある CICS Transaction Gateway に接続するものです。 CICS TS との通信には ECI/EPI インターフェースを使用します。	CICS TS との通信に CICS ECI/EPI インターフェースを使用するサーブレットまたは JSP を作成します。 このサーブレットまたは JSP には、任意の Web クライアントから WebSphere を通じてアクセスできます。	物理/論理中間層の Web サーバーからアプレットをダウンロードします。このアプレットは、物理/論理中間層にある CICS Transaction Gateway に接続してから、CICS TS と通信します。	CICS TS との通信に CICS クライアントの ECI/EPI インターフェースを使用する EJB を作成します。
WebSphere MQ	Web クライアントに Java アプリケーションをインプリメントします。これは、WebSphere MQ Client for Java を使用して物理/論理中間層にある WebSphere MQ Server for Windows などに接続するものです。 z/VSE は次に、WebSphere MQ Client および WebSphere MQ Client Trigger Monitor を使用して、例えば物理/論理中間層にある、WebSphere MQ Server for Windows に接続し、z/VSE アプリケーションのトリガーまたはキューへのメッセージ到着を使用します。	z/VSE ホスト上でトリガーを通じて CICS トランザクションを開始するために、WebSphere MQ Client for Java を使用してメッセージを MQ キューに書き込むサーブレットまたは Java Server Page (JSP) を作成します。このサーブレットまたは JSP には、任意の Web クライアントから WebSphere を通じてアクセスできます。	物理/論理中間層の Web サーバーからアプレットをダウンロードし、「ルーター」を使用してリモート z/VSE ホストへ接続し、VSE ベースのデータにアクセスします。	WebSphere MQ Client for Java を使用して z/VSE ホストと通信し、そのホスト上のデータベースにアクセスする Enterprise Java Bean (EJB) を作成します。

注: *VSE* ファイル・システム という用語には、VSE/VSAM、VSE/POWER、VSE/Librarian、および VSE/ICCF が含まれます。

第 4 章 共通前提条件プログラムのインストール

このトピックでは、コネクターの選択に関わらずに実行しなければならないアクティビティーを記載します。

以下の項目があります。

- 『TCP/IP の構成およびアクティブ化』
- 『VSE HTTP Server の構成およびアクティブ化』
- 24 ページの『Java のインストールと構成』
- 25 ページの『IBM HTTP Serverのインストール』
- 25 ページの『WebSphere Application Serverのインストール』

TCP/IP の構成およびアクティブ化

以下のものを使用するには、TCP/IP が必要です。

- Java ベース・コネクター
- Db2 ベース・コネクター
- VSAM リダイレクター・コネクター
- VSE スクリプト・コネクター

z/VSE で提供される 3 つの TCP/IP スタック (TCP/IP for z/VSE、IPv6/VSE、または LFP) から選択できます。 TCP/IP for z/VSE と IPv6/VSE をアクティブにするためには、鍵が必要です。この鍵は購入する必要があります。

TCP/IP のアクティブ化および構成方法について詳しくは、「*IBM z/VSE TCP/IP* サポート」を参照してください。

注: TCP/IP の代わりに VTAM / APPC を使用できる場面については、この資料に記載しています。

VSE HTTP Server の構成およびアクティブ化

VSE HTTP Server は、TCP/IP for z/VSE *Application Pak* (アプリケーション・パッケージ) に含まれています。したがって、VSE HTTP Server を構成してアクティブ化する方法について詳しくは、「*TCP/IP for VSE Installation Guide*」(SC33-6741) を参照してください。この資料は *PDF* ファイルのみ で入手可能です。この資料は、*VSE* コレクション のオンライン・ライブラリーから入手できます。 VSE コレクションのオンライン・ライブラリーは、以下から取得できます。

- CD-ROM (番号 SK2T-0060)。
- DVD (番号 SK3T-8348)。

Java のインストールと構成

Java は、Java ソース・ファイルからバイトコード が作成されるプログラム言語です。 このバイトコードは、Java クラス・ファイル に保管されます。 これらのクラス・ファイルは、Java インタープリターによって読み取られて実行されます (Windows および OS/2 では、このプログラムは **java.exe** です)。

前述のように、z/VSE Java ベース・コネクターは VSE コネクター・クライアント および VSE コネクター・サーバー から構成されます。 *VSE* コネクター・クライアント を使用する Web アプリケーションを開発するには、開発プラットフォームに Java Development Kit (JDK) 1.4 以降をインストールする必要があります。

注**:**

1. 同じ物理/論理中間層プラットフォームに別のバージョンの Java をインストールできますが、それには実行する Java バージョン用の正しいパスが設定されていることを確認する必要があります。
2. Java ベース・コネクターで提供されるサンプルを使用する際に、Java *Swing* クラスが必要になる場合があります。 このクラスは、 VSE コネクター・クライアント のインストール時に提供されます (28 ページの『VSE コネクター・クライアントのインストール』に記載)。

Java 基本コードのダウンロード

以下の Web サイトから Java 関連コードをダウンロードできます。

- `http://www.ibm.com/developerworks/java/jdk/`

  (英語のみ)。例えば、AIX、OS/2、z/OS® (Unix サービス)、OS/400®、z/VM®、Linux、および Windows 版の Java Development Kits (JDKs) をダウンロードできます。
- `http://www.ibm.com/developerworks/java/`

  ツールと資料の両方を含む各種ライブラリーにアクセスできます。
- `http://www.oracle.com/technetwork/java/`

  例えば、Windows、Linux、Solaris のいずれかのプラットフォーム版の Java 2 Enterprise Edition をダウンロードできます。

インストールする Java パッケージの決定

次の 2 つのいずれかの方法で Java を物理/論理中間層サーバーにインストールできます。

- Java Development Kit (JDK) のインストール。 **java.exe** を使用して Java プログラムを実行します。 JDK にはランタイム環境と、デバッガー、コンパイラーなどのツールおよび機能が含まれます。独自の Java プログラムを開発する場合に JDK が必要です。
- Java ランタイム環境 (JRE) のインストール。 JRE には、Java アプリケーションの実行に必要なランタイム・ライブラリーのみが含まれます (開発ツールは含まれません)。 ここでは、**jre.exe** を使用して Java プログラムを実行します。

IBM HTTP Serverのインストール

物理/論理中間層に、以下のような *Web* サーバー をインストールする必要があります。

- IBM HTTP Server
- Lotus Domino Go Webserver
- Apache Server

IBM HTTP Server は WebSphere Application Server パッケージのパーツです。WebSphere Application Server のインストール時に、以下の質問があります。

- IBM HTTP Server を Web サーバーとしてインストールするか。
- 別の Web サーバーを使用するか (例えば、Apache Server)。

IBM HTTP Server を物理/論理中間層にインストールする方法について詳しくは、WebSphere Application Server に付属のインストール手順を参照してください。

WebSphere Application Serverのインストール

中間層 (「物理」中間層または「論理」中間層のいずれか。 10 ページの『3 層環境の概要』を参照) にアプリケーション・サーバー (IBM WebSphere Application Server または他のベンダーのアプリケーション・サーバー) をインストールする必要があります。

WebSphere Application Server には、以下のエディションがあります。

- Standard Edition。サーブレットおよび Java Server Pages (JSPs) をサポートします。
- Advanced Edition または Enterprise Edition。いずれもサーブレット、JSPs、および Enterprise Java Beans (EJBs) をサポートします。

さまざまなプラットフォームでの WebSphere Application Server のインストールは複雑で、変更される可能性があるため、このトピックでは、参考資料または一般的なインストール・ステップの説明 (あるいはその両方) のみを記載します。

Linux on z Systems への WebSphere Application Server のインストール

WebSphere Application Server およびサポートする製品をメインフレーム上の Linux on z Systems プラットフォームにインストールする方法については、関連する IBM 資料を参照してください。

z/OS への WebSphere Application Server のインストール

WebSphere Application Server およびサポートする製品をメインフレーム上の z/OS プラットフォームにインストールする方法については、関連する IBM 資料を参照してください。

他のプラットフォームへの WebSphere Application Server のインストール

Windows、Linux、AIX、Sun Solaris などのプラットフォームに WebSphere Application Server をインストールするときに従う標準的な 手順があります。

1. インストールしようとするプラットフォームおよび構成のハードウェアおよびソフトウェアの前提条件を満たしていることを確認します。
2. 行う必要があるインストール・ステップを決定します。 例えば、お客様のシステムにまだ WebSphere Application Server、IBM Java Development Kit、IBM HTTP Server、および Db2 Universal Database がインストールされていない場合には、これらの製品がすでにインストールされている場合とは異なる手順になります。
3. 以下の方法で IBM Java Development Kit をインストールします。
   a. **exec** ファイルの実行。
   b. 各ウィンドウに表示される手順に従います。
4. IBM HTTP Server (または Apache などの別の Web サーバー) をインストールします。
5. IBM Db2 Universal Database (または Oracle などの別のデータベース・システム) をインストールします。
6. お客様の構成用の前提条件製品のインストールをテストします。
7. 選択したプラットフォームに WebSphere Application Server をインストールします。
8. WebSphere Application Server および前提条件製品のインストールをテストします。

ただし、Windows、Linux、AIX、または Sun Solaris などのプラットフォームへの WebSphere Application Server のインストールについての情報源は主に、IBM の Web ページであり、常に更新されています。IBM Web ページは、次のアドレスにあります。

```
http://www.ibm.com/software/webservers/appserv/library.html
```

第 5 章 Java ベース・コネクターのインストールおよび操作

このトピックでは、Java ベース・コネクターをインストールして操作する方法について説明します。

以下の項目があります。

- 『Java ベース・コネクター の概要』
- 28 ページの『VSE コネクター・クライアントのインストール』
- 31 ページの『VSE コネクター・クライアント のアンインストール』
- 31 ページの『VSE コネクター・サーバー の構成』
- 39 ページの『VSE コネクター・サーバー のための日付形式の構成』
- 39 ページの『VSE コネクター・サーバー の始動』
- 40 ページの『VSE コネクター・クライアント とコネクター・サーバーの間の通信のテスト』
- 41 ページの『VSE コネクター・サーバー のコマンドのリストの取得』
- 41 ページの『VSE コネクター・サーバー のコマンドの入力』
- 41 ページの『VSE コネクター・サーバー を使用したセキュリティーの維持』

Java ベース・コネクター の概要

Java ベース・コネクターは、クライアント・パーツ (*VSE* コネクター・クライアント) とサーバー・パーツ (*VSE* コネクター・サーバー) から構成されます。

VSE コネクター・クライアント の概要

VSE コネクター・クライアント は、ほとんどの Java 対応のプラットフォームにインストールできます。 このパーツは、以下のコンポーネントにより構成されます。

- メイン・ファイル **VSEConnector.jar**。VSE Java Beans クラス・ライブラリーが含まれます。 VSE Java Beans は、z/VSE ホスト上の VSE/VSAM、VSE/Librarian、VSE/POWER、VSE/ICCF、およびオペレーター・コンソールとの通信用の Java プログラミング・インターフェースを提供します。
- 3 つの追加 JAR ファイル (**ibmjsse.jar**、**cci.jar**、および **ibmpkcs.jar**)。
- Java ソース・コードを組み込んだサンプルのセット。VSE Java Beans を使用して Java プログラムを作成する方法を示します。
- オンライン・ドキュメンテーション (HTML ページ・セット)。さまざまな概念およびサンプルを記載しています。

独自の Web アプリケーションを開発するには、上記のコンポーネントをすべて使用することになります。 ただし、完成した Web アプリケーションの実行時には VSE Java Beans クラス・ライブラリーのみが必要です。 したがって、実稼働環境 (エンド・ユーザー環境) 向けの独自の Web アプリケーションの準備ができたら、**VSEConnector.jar** ファイル (VSE Java Beans クラス・ライブラリーを含む) のみを以下の場所にインストールします。

- 物理/論理中間層 (3 層環境の場合)
- Web クライアント (2 層環境の場合)

2 層環境と 3 層環境の違いについては、 9 ページの『第 2 章 2 層および 3 層環境の概要』で説明します。

VSE コネクター・サーバー の概要

VSE コネクター・サーバー は z/VSE ホストにインストールされます。 これは、デフォルトでは動的クラス R で実行されるバッチ・アプリケーションです。VSE コネクター・サーバー の構成後に操作可能にするために、このバッチ・アプリケーションを開始する必要があります。 開始されると、このアプリケーションは複数のクライアントを扱うことができる TCP/IP ソケット・リスナー を提供します。

Web クライアントまたは物理/論理中間層で実行される Java プログラムは VSE Java Beans を使用して、z/VSE ホスト上で実行される VSE コネクター・サーバー への接続を構築します。

VSE コネクター・サーバー を開始するには、ジョブ **STARTVCS** を使用します。これは、z/VSE の基本インストール中に POWER 読み取り待ち行列に入れられます。開始されると、VSE コネクター・サーバー はデフォルトではポート 2893 上の着信 TCP/IP トラフィックを listen します。 VSE コネクター・サーバー の開始方法について詳しくは、 39 ページの『VSE コネクター・サーバー の始動』を参照してください。

VSE コネクター・サーバー は、使用できるように事前に構成済みで、お客様が大幅に構成を変更する必要はありません。 しかし、構成メンバーを変更して以下の項目を指定することができます。

- VSE コネクター・サーバー によってアクセスできる VSE/AF ライブラリー。このリストは、お客様の要件に応じて拡張したり、制限したりできます。
- VSE コネクター・サーバー のスタートアップ時にロードされるプラグイン。297 ページの『第 21 章 Java ベース・コネクターの拡張』の説明に従って、独自のホスト・サイドのプラグインを指定することにより Java ベース・コネクターを拡張できます。
- VSE コネクター・サーバー へのログオンを許可されるユーザーまたはユーザー・グループ。

2 層および 3 層環境内で VSE コネクター・クライアント および VSE コネクター・サーバー が使用される場面の概要については、 10 ページの図 2 および 11 ページの図 3 を参照してください。

VSE コネクター・クライアントのインストール

このトピックでは、*3* 層環境 の物理/論理中間層サーバーに VSE コネクター・クライアントをインストールする方法について説明します。

3 層環境については、 10 ページの『3 層環境の概要』に説明があります。

Web クライアントで実行されるアプレットおよび Java アプリケーションを使用する 2 層環境をインプリメントする予定であれば、 VSE コネクター・クライアント

の VSE Java Beans パーツ (ファイル **VSEConnector.jar**) をアプレットおよび Java アプリケーションが実行される各 Web クライアントにコピーする必要があります。 Web クライアントと z/VSE ホスト上で実行される VSE コネクター・サーバー の間に接続を確立するために VSE Java Beans が必要です。

VSE コネクター・クライアントは VSE 中央機能に組み込まれていて、1 ファイル (**iesincon.w**) で構成されています。

VSE コネクター・クライアント のコピーの取得

注**:** 始める前に、VSE コネクター・クライアントをインストールしようとする開発プラットフォームに Java Development Kit (JDK) 1.4 以降がすでにインストールされていなければなりません。 JDK 1.4 以降をインストールしていない場合のインストール方法について詳しくは、 24 ページの『Java のインストールと構成』を参照してください。

VSE コネクター・クライアント のコピーを取得するには、以下のいずれから取得するかを決定する必要があります。

- インターネットから
- Extended Base Tape から VSE コネクター・ワークステーション・コード・コンポーネントをインストールすることによって

インターネットから クライアントを取得するには、以下を行ってください。

1. Web ブラウザーを起動して次の URL に進みます。

   ```
   http://www.ibm.com/systems/z/os/zvse/downloads/
   ```

2. VSE コネクター・クライアント のセクションから、VSE コネクター・クライアント をインストールするディレクトリーに、ファイル **vsecon*nnn*.zip** をダウンロードします。 注**:** *nnn* は現行 VSE バージョン (**vsecon620.zip**) です。

クライアントを *VSE* コネクター・ワークステーション・コード・コンポーネントをインストールすることによって 取得するには、以下を行ってください。

1. Extended Base Tape から VSE コネクター・ワークステーション・コード・コンポーネントをインストールします。このコンポーネントのインストール後、VSE コネクター・クライアント の VSE コネクター・ワークステーション・コード **iesincon.w** が、z/VSE サブライブラリー PRD2.PROD に保管されます。
2. TCP/IP for z/VSE の FTP (ファイル転送プログラム) ユーティリティーを使用して、VSE コネクター・クライアント をインストールするディレクトリーに **iesincon.w** をダウンロードします。

注**:**

1. **iesincon.w** はバイナリー でダウンロードする必要があります。
2. UNIX モードがオフ になっていることを確認してください。 オフにしないと、バイナリー を指定しても、**iesincon.w** は ASCII モードでダウンロードされます。 *Unix* モード は VSE FTP デーモンのパラメーターの 1 つです。 FTP クライアントの中には、UNIX モードを強制的に オンにするものがあります。 以下の例に、バッチ FTP クライアントを使用して正常に **iesincon.w** を転送する方法を示します。 UNIX モードが設定される場所は太字で示しています。

```
c:¥temp>ftp 9.164.155.2
Connected to 9.164.155.2.
220-TCP/IP for VSE -- Version 01.05.F  -- FTP Daemon
    Copyright (c) 1995,2006 Connectivity Systems Incorporated
220 Service ready for new user.
User (9.164.155.2:(none)): sysa
331 User name okay, need password.
Password:
230 User logged in, proceed.
ftp> cd prd2
250 Requested file action okay, completed.
ftp> cd prod
250 Requested file action okay, completed.
ftp> binary
200 Command okay.
ftp> get iesincon.w
200 Command okay.
150-File: PRD2.PROD.IESINCON.W
    Type: Binary Recfm: FB Lrecl:    80 Blksize:      80
    CC=ON  UNIX=OFF RECLF=OFF TRCC=OFF CRLF=ON  NAT=NO
150 File status okay; about to open data connection
226-Bytes sent:       4,756,400
    Records sent:        59,455
    Transfer Seconds:        16.52 (    290K/Sec)
    File I/O Seconds:         3.94 (  1,548K/Sec)
226 Closing data connection.
4756400 bytes received in 17,12 seconds (277,91 Kbytes/sec)
ftp> bye
221 Service closing control connection.
c:¥temp>ren iesincon.w vsecon.zip
```

VSE コネクター・クライアント のインストールの実行

VSE コネクター・クライアントのインストールを実行するには、以下の作業を行う必要があります。

1. ファイル **vsecon.zip** を unzip します。これには以下のファイルが入っています。
   - setup.jar (VSE コネクター・クライアント・コードを含む)
   - setup.bat または setup.cmd (Windows 用インストール・バッチ・ファイル)
   - setup.sh (Linux/Unix 用インストール・スクリプト)
2. (ファイルをダブルクリックして) オペレーティング・システム・プラットフォームに適用可能なバッチ・ファイルを開始します。
3. インストール・プロセスが開始され、さまざまなインストール・メニューによるガイドが表示されます。
4. HTML ベースのドキュメンテーションにアクセスするには、ここで Web ブラウザーを使用してファイル **VSEConnectors.html** をオープンすることができます。

オンライン・ドキュメンテーション・オプションの使用

VSE コネクター・クライアント をインストールしたら、オンライン・ドキュメンテーションを介して すべての情報 (サンプルおよびソース・コードを含む) にアクセスできます。

オンライン・ドキュメンテーションのメイン・ウィンドウには、以下の項目へのリンク (このウィンドウの左側) が含まれています。

- Java ベース・コネクターおよび Db2 ベース・コネクターと、 CICS 接続および WebSphere MQ 接続のセットアップ。ここには、これらのコネクターおよびセットアップの情報および例があります。
- Java アプレット、サーブレット、Java Server Pages (JSP)、Enterprise Java Beans (EJB)、および JDBC などの概念の説明。 ここには、これらのトピックに関する情報および例があります。
- オンライン解説書 (「**All classes (**全クラス**)**」の下)。独自のプラグイン を作成することで VSE Java Beans を拡張する方法についての情報の解説が含まれます。
- Web ブラウザーから直接実行できるサンプル。 これらのサンプルを実行するには、TCP/IP for z/VSE および z/VSE ホスト で実行される VSE コネクター・サーバー が必要です。

  注**:** いずれかのサンプルを実行する前、あるいは VSE Java Beans を使用した独自の Java プログラムを作成する前に、ローカル CLASSPATH 変数に **VSEConnector.jar** ファイルを含める必要があります。

WebSphere サポートの構成

IBM WebSphere Application Server を物理/論理中間層にインプリメントする予定であれば、 WebSphere への物理/論理中間層の構成方法について記載したオンライン・ドキュメンテーションにあるステップを、ここで完了する必要があります。 この情報を検索するには、オンライン・ドキュメンテーション・ウィンドウ (30 ページの『オンライン・ドキュメンテーション・オプションの使用』で説明) にある「詳細情報 **(Further Information)**」をクリックしてから、ご使用のインストール済み環境に適切なバージョンの WebSphere を選択します。

ここで、オンライン・ドキュメンテーションに説明されているステップに従います。

VSE コネクター・クライアント のアンインストール

VSE コネクター・クライアント のアンインストールには、以下の 2 つの方法があります。

- 通常の *Windows* プログラムの追加/削除機能を使用。これは、Windows の「コントロール パネル」から選択します。
- VSE コネクター・クライアント アンインストール・プログラムを手動で実行。このプログラムは、**_uninst** サブディレクトリーにあります。このサブディレクトリーは、VSE コネクター・クライアント をインストールしたディレクトリー内にあります。 このオプションは、すべてのプラットフォームで使用できます。

VSE コネクター・サーバー の構成

VSE コネクター・サーバー は、いずれかの z/VSE 区画でバッチで実行されるアプリケーションで、TCP/IP 接続をインプリメントしています。

VSE コネクター・サーバーについては、 28 ページの『VSE コネクター・サーバーの概要』に説明があります。

このトピックでは、VSE コネクター・サーバー の構成に使用するジョブについて説明します。

SKVCSSTJ
スケルトン・スタートアップ・ジョブ。

SKVCSCAT
VSE コネクター・サーバーの構成メンバーをカタログするジョブ。ジョブ SKVCSCAT 内に含まれるスケルトンは、以下のとおりです。

SKVCSCFG
VSE コネクター・サーバー の一般的な設定を指定する VSE ライブラリー・メンバー。

SKVCSLIB
VSE コネクター・サーバー がアクセスできる VSE ライブラリーを指定する VSE ライブラリー・メンバー。

SKVCSPLG
VSE コネクター・サーバー のスタートアップ時にロードされるサーバー・プラグインを指定する VSE ライブラリー・メンバー。

SKVCSUSR
VSE コネクター・サーバー にログオンできるユーザーまたはユーザーのグループを指定する VSE ライブラリー・メンバー。

SKVCSSSL
Secure Sockets Layer/Transport Layer Security (SSL/TLS) セキュリティー用に VSE コネクター・サーバー を構成する VSE ライブラリー・メンバー。

ジョブ SKVCSSTJ - スタート・アップ・ジョブ

スケルトン・ジョブ SKVCSSTJ は VSE/ICCF ライブラリー 59 にあります。

SKVCSSTJ を使用してスタートアップ・ジョブ (VSE コネクター・サーバー のスタートアップ用) を VSE/POWER 読み取り待ち行列に入れます。 SKVCSSTJ は Z ブックとしても入手可能で、z/VSE の初期インストール時、または z/VSE のコールド・スタートアップ時に POWER 読み取り待ち行列にロードされます。

```
* $$ JOB JNM=CATSTVCS,DISP=D,CLASS=0
// JOB CATSTVCS       CATALOG STARTVCS AND LDVCS, LOAD STARTVCS
// EXEC LIBR,PARM='MSHP'
ACC S=IJSYSRS.SYSLIB
CATALOG  STARTVCS.Z        REPLACE=YES
$$$$ JOB JNM=STARTVCS,DISP=L,CLASS=R
$$$$ LST CLASS=A,DISP=D
// JOB STARTVCS START UP VSE CONNECTOR SERVER
// ID USER=VCSRV
*   WAITING FOR TCP/IP TO COME UP
// EXEC REXX=IESWAITR,PARM='TCPIP00'
// LIBDEF *,SEARCH=(PRD2.CONFIG,PRD2.TCPIPC,PRD1.BASE,PRD2.SCEEBASE)
// OPTION SYSPARM='00'
// EXEC IESVCSRV,PARM='DD:PRD2.CONFIG(IESVCSRV.Z)'
$$/*
$$/&
$$$$ EOJ
/+
CATALOG  LDVCS.PROC     REPLACE=YES DATA=YES
// EXEC DTRIINIT
   LOAD STARTVCS.Z
/*
/+
/*
// EXEC PROC=LDVCS        TO LOAD  VCS STARTUP  INTO RDR QUEUE
/&
* $$ EOJ
```

図 *4.* ジョブ *SKVCSSTJ (*読み取り待ち行列でのスタートアップ・ジョブの配置用*)*

ジョブ SKVCSCAT - カタログ・メンバー

ジョブ SKVCSCAT は VSE/ICCF ライブラリー 59 にあります。

SKVCSCAT を使用して、このトピックにリストしている構成メンバーをカタログします。

```
* $$ JOB JNM=VCSCAT,DISP=D,CLASS=0
// JOB VCSCAT CATALOG VCS CONFIGURATION MEMBERS
// EXEC LIBR,PARM='MSHP'
ACCESS S=PRD2.CONFIG
CATALOG IESVCSRV.Z REPLACE=Y
* $$ SLI ICCF=(SKVCSCFG),LIB=(YY)
/+
CATALOG IESLIBDF.Z REPLACE=Y
* $$ SLI ICCF=(SKVCSLIB),LIB=(YY)
/+
CATALOG IESUSERS.Z REPLACE=Y
* $$ SLI ICCF=(SKVCSUSR),LIB=(YY)
/+
CATALOG IESPLGIN.Z REPLACE=Y
* $$ SLI ICCF=(SKVCSPLG),LIB=(YY)
/+
CATALOG IESSSLCF.Z REPLACE=Y
* $$ SLI ICCF=(SKVCSSSL),LIB=(YY)
/+
/*
/&
* $$ EOJ
```

図 *5.* ジョブ *SKVCSCAT (VSE* コネクター・サーバー のメンバーのカタログ用*)*

VSE ライブラリー・メンバー SKVCSCFG - 一般的な設定

スケルトン SKVCSCFG は VSE/ICCF ライブラリー 59 にあります。これを使用して VSE コネクター・サーバー の一般的な設定を指定します。

図 *6.* メンバー *SKVCSCFG (VSE* コネクター・サーバー の一般的な設定の指定用*)*

```
; *****************************************************************
;      MAIN CONFIGURATION MEMBER FOR VSE CONNECTOR SERVER
; *****************************************************************

; *****************************************************************
; TRACING SPECIFIC SETTINGS:
; - TRACEON    : A 32 BIT HEX VALUE PREFIXED WITH '0X'
;                0X00000000 IS OFF, 0XFFFFFFFF IN ON
; - TRACEFILE  : DESTINATION FOR TRACE MESSAGES
;                DD:SYSLOG, DD:SYSLST OR DD:LIB.SLIB(NAME.TYPE)
; *****************************************************************
  TRACEON      = 0X00000000  ; TRACE IS OFF
  TRACEFILE    = DD:SYSLOG   ; TRACE GOES TO SYSLOG

; *****************************************************************
; TCP/IP - SERVER SPECIFIC CONFIGURATIONS
; - SERVERPORT     : THE TCP PORT WHERE THE SERVER IS LISTENING
; - MAXCLIENTS     : THE MAXIMUM NUMBER OF CONCURRENT CLIENTS
; - SSLENABLE      : YES/NO - USE SECURE SOCKET LAYER
; - RESTRICTLISTEN : OFF/IPV4/IPV6 - ACCEPT ONLY IPV4 OR IPV6 (OPTIONAL)
; - BINDADDR       : SPECIFIES AN IP ADDRESS USED TO BIND (OPTIONAL)
; *****************************************************************
  SERVERPORT     = 2893
  MAXCLIENTS     = 256
  SSLENABLE      = NO

; *****************************************************************
; SECURITY CONFIGURATION
; - SECURITY: EXTENDED - LOGON, RESOURCE, USER TYPE AND FACILITY
;                        CHECKING.
;             FULL     - LOGON, RESOURCE AND USER TYPE CHECKING
;             RESOURCE - LOGON AND RESOURCE, BUT NO USER TYPE
;                        CHECKING.
;             LOGON    - LOGON, BUT NO RESOURCE AND USER TYPE
;                        CHECKING
;             NO       - NO LOGON, RESOURCE AND USER TYPE CHECKING
;
;   NOTE: FOR SECURITY = EXTENDED THE FOLLOWING FACILITY RESOURCES
;         ARE USED: VSE.CONNECTOR          - GENERAL ACCESS
;                   VSE.CONNECTOR.CONSOLE  - CONSOLE ACCESS
;                   VSE.CONNECTOR.ICCF     - ICCF ACCESS
;                   VSE.CONNECTOR.LIBR     - LIBR ACCESS
;                   VSE.CONNECTOR.POWER    - POWER ACCESS
;                   VSE.CONNECTOR.VSAM     - VSAM ACCESS
;         RESOURCE VSE.CONNECTOR IS CHECKED FOR READ AUTHORITY
;         ONLY. ALL OTHER FACILITIES ARE CHECKED ACCORDING TO
;         THE DESIRED ACCESS REQUEST (READ, UPDATE, ALTER)
; *****************************************************************
  SECURITY = FULL

; *****************************************************************
; TIMEOUT FOR VSAM AUTO CLOSE
; - AUTOCLOSE : THE TIMEOUT IN SECONDS FOR VSAM AUTOCLOSE
; *****************************************************************
  AUTOCLOSE    = 600

; *****************************************************************
; DEFAULT CLASS FOR JOBS
; - DEFAULTCLASS : THE JOB CLASS WHERE UTILITY JOBS SHOULD RUN
; *****************************************************************
```

```
  DEFAULTCLASS = P

; ******************************************************************
; CODE PAGE CONVERSIONS
; - ASCII_CP  : ASCII CODE PAGE
; - EBCDIC_CP : EBCDIC CODE PAGE
; ******************************************************************
  ASCII_CP     = IBM-850
  EBCDIC_CP    = IBM-1047

; ******************************************************************
; SYSTEM LANGUAGE
; - LANGUAGE : THE SYSTEM'S LANGUAGE (E, J)
; ******************************************************************
  LANGUAGE = E

; ******************************************************************
; DESCRIPTION SENT AS IDENTIFY
; - DESCRIPTION : THIS STRING IS SENT AS IDENTIFY TO THE CLIENT
;                 CHANGE '<YOUR SYSTEM>' TO YOUR SYSTEM'S NAME
; ******************************************************************
  DESCRIPTION   = VSE CONNECTOR SERVER ON <YOUR SYSTEM>

; ******************************************************************
; MESSAGES CONFIGURATION
; THIS SETTING CONTROL WHERE THE CONNECTION AND USER MESSAGES ARE
; WRITTEN TO.
; - MESSAGES: OFF      - NO MESSAGES WILL BE SHOWN.
;             SYSLOG   - THE MESSAGES WILL GO TO SYSLOG
;             SYSLST   - THE MESSAGES WILL GO TO SYSLST
;             BOTH     - THE MESSAGES WILL GO TO SYSLOG AND SYSLST
; ******************************************************************
  MESSAGES = SYSLOG

; ******************************************************************
; SUB CONFIGURATION MEMBERS NEEDED FOR VSE CONNECTOR SERVER
; - LIBRCFGFILE   : LIBRARY DEFINITION FILE. CONTAINS THE LIBRARIES
;                   THAT ARE VISIBLE FOR THE VSE CONNECTOR SERVER.
; - USERSCFGFILE  : USER/SECURITY CONFIG FILE. DEFINES ADDITIONAL
;                   SECURITY FOR USERS AND IP ADDRESSES.
; - PLUGINCFGFILE : PLUGIN CONFIG FILE. DEFINES THE PLUGINS THAT
;                   ARE LOADED AT SERVER STARTUP.
; - SSLCFGFILE    : SSL CONFIG FILE. DEFINES SSL PARAMETERS
; NOTE: YOU HAVE TO CHANGE THE NAMES AND LOCATIONS OF THESE MEMBERS
;       IN THIS MEMBER IF YOU MOVE THEM TO ANOTHER LIBRARY!
; ******************************************************************
  LIBRCFGFILE   = DD:PRD2.CONFIG(IESLIBDF.Z)
  USERSCFGFILE  = DD:PRD2.CONFIG(IESUSERS.Z)
  PLUGINCFGFILE = DD:PRD2.CONFIG(IESPLGIN.Z)
  SSLCFGFILE    = DD:PRD2.CONFIG(IESSSLCF.Z)
```

VSE ライブラリー・メンバー SKVCSLIB - アクセスされるライブラリーの指定

スケルトン SKVCSLIB は VSE/ICCF ライブラリー 59 にあります。

SKVCSLIB を使用して、VSE コネクター・サーバー でアクセスできるライブラリーを指定します。このスケルトンはライブラリーのリストで構成され、独自の要件に応じて拡張したり、制限したりできます。 ただし、各ライブラリー名は別々の行に入力する必要があります。

```
PRD1
PRD2
PRIMARY
IJSYSRS
```

図 *7. メンバー SKVCSLIB (VSE* コネクター・サーバー でアクセスするライブラリーの指定用*)*

VSE ライブラリー・メンバー SKVCSPLG - ロードされるプラグインの指定

スケルトン SKVCSPLG は VSE/ICCF ライブラリー 59 にあります。

SKVCSPLG を使用して、VSE コネクター・サーバー の始動時にロードされる VSE コネクター・サーバー のプラグインを指定します。

PLUGIN ステートメントの構文は以下のとおりです。

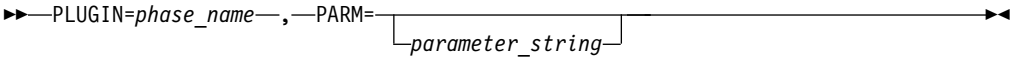


図 8 に、VSE コネクター・サーバー のプラグインの指定に使用するメンバーを示します。

```
* ****************************************************************
* VSE CONNECTOR SERVER PLUGIN CONFIGURATION MEMBER
* ****************************************************************
* THE FOLLOWING PLUGINS ARE LOADED DURING STARTUP OF THE SERVER
* UNCOMMENT THE SAMPLES BELOW TO CHANGE THEM
* ****************************************************************
PLUGIN=IESSAPLG,PARM=CICS=F2,CONS=IESA,TRANS=IEXM,EXIT=IESSAEXT
PLUGIN=IESHTOHP,PARM=
PLUGIN=IESCOMPH,PARM=
PLUGIN=IESVSAPL,PARM=
PLUGIN=IESDLIPL,PARM=
* PLUGIN=SAMPLE,PARM=MY PARAMTER STRING
* PLUGIN=<PHASE NAME>,PARM=<PARAMETER STRING>
```

図 *8. メンバー SKVCSPLG (VSE* コネクター・サーバー のプラグインの指定用*)*

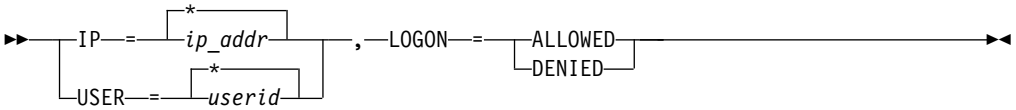
独自のプラグインの作成方法について詳しくは、 297 ページの『第 21 章 Java ベース・コネクターの拡張』を参照してください。

VSE ライブラリー・メンバー SKVCSUSR - ログオン・アクセスの指定

スケルトン SKVCSUSR は VSE/ICCF ライブラリー 59 にあります。

SKVCSUSR を使用して、VSE コネクター・サーバー にログオンできるユーザーまたはユーザーのグループを指定します。 このスケルトンを使用することで、VSE コネクター・サーバー が特定のユーザーまたは IP アドレスからアクセスされないようにもできます。

このメンバーの構文は以下のとおりです。




```
* ****************************************************************
* VSE CONNECTOR SERVER USER SECURITY CONFIGURATION MEMBER
* YOU CAN EITHER ALLOW OR DENY THE LOGON FOR A SPECIFIED
* USER OR IP OR GROUP OF USERS AND IP ADDRESSES.
* ****************************************************************
* USERS WITH THE FOLLOWING IP ADDRESSES ARE ALLOWED OR DENIED TO LOGON
* WHEN SPECIFYING AN IPV6 ADDRESS, USE THE FULLY QUALIFIED FORMAT.
* UNCOMMENT THE SAMPLES AND MODIFY THEM
* ****************************************************************
IP   = *,             LOGON = ALLOWED
* IP = 9.164.123.456, LOGON = DENIED
* IP = 9.165.*      , LOGON = DENIED
* IP = 10.0.0.*     , LOGON = ALLOWED
* IP = 2001:0DB8:85A3:0000:0000:8A2E:0370:7334 , LOGON = DENIED
* IP = 2001:0DB8:85A3:0000:0000:8A2E:0370:*    , LOGON = ALLOWED
* ****************************************************************
* THESE USERS ARE ALLOWED OR DENIED TO LOGON
* UNCOMMENT THE SAMPLES AND MODIFY THEM
* ****************************************************************
USER = *,             LOGON = ALLOWED
* USER = BOBY,        LOGON = ALLOWED
* USER = SYS*,        LOGON = DENIED
```

図 *9.* メンバー *SKVCSUSR (VSE* コネクター・サーバー へのログオン・アクセスの指定用*)*

注**:**

1. 図 9 で何も入力しないと、アクセス許可は何も 定義されません。
2. IP アドレスまたはユーザー名にワイルドカード (*) を使用できます。

VSE ライブラリー・メンバー SKVCSSSL - SSL/TLS の構成

スケルトン SKVCSSSL は VSE/ICCF ライブラリー 59 にあります。

SKVCSSSL を使用して、以下を指定します。

- 使用する SSL/TLS のバージョン。
- VSE コネクター・サーバー で使用される *VSE* 鍵リング・ライブラリー の名前 (詳しくは、「*IBM z/VSE* 管理」の『SSL/TLS を使用するためのシステムの準備』のトピックを参照)。
- VSE コネクター・サーバー で使用される サーバー証明書 の名前 (詳しくは、「*IBM z/VSE* 管理」の『SSL を使用するためのシステムの準備』のトピックを参照)。
- セッション・タイムアウト (秒単位)。完全なハンドシェーク を必要とせずに VSE コネクター・サーバー が VSE コネクター・クライアント の再接続を許可する秒数。

```
; ****************************************************************
;      SSL CONFIGURATION MEMBER FOR VSE CONNECTOR SERVER
; ****************************************************************

; ****************************************************************
; SSLVERSION   SPECIFIES THE MINIMUM VERSION THAT IS TO BE USED
;              POSSIBLE VALUES ARE:
;              - SSL30
;              - TLSV1
;              - TLS12   (CSI ONLY)
;              - SSLV3   (OPENSSL ONLY)
;              - TLSV1.2 (OPENSSL ONLY)
;              - ALL     (OPENSSL ONLY)
; KEYRING      SPECIFIES THE SUBLIBRARY WHERE THE KEY FILES ARE
;              STORED.
; CERTNAME     NAME OF THE CERTIFICATE THAT IS USED BY THE SERVER
; SESSIONTIMEOUT NUMBER OF SECONDS THAT THE SERVER WILL USE TO
;              ALLOW A CLIENT TO RECONNECT WITHOUT PERFORMING A
;              FULL HANDSHAKE. (86400 SEC = 24 HOURS)
; AUTHENTICATION TYPE OF AUTHENTICATION. POSSIBLE VALUES ARE:
;              SERVER - SERVER AUTHENTICATION ONLY
;              CLIENT - SERVER AND CLIENT AUTENTICATION
;              LOGON  - SERVER AND CLIENT AUTENTICATION WITH LOGON
;                       THE CLIENT CERTIFICATE IS USED TO LOGON.
; ****************************************************************
SSLVERSION     = TLSV1
KEYRING        = CRYPTO.KEYRING
CERTNAME       = SAMPLE
SESSIONTIMEOUT = 86400
AUTHENTICATION = SERVER

; ****************************************************************
; CIPHERSUITES SPECIFIES A LIST OF CIPHER SUITES THAT ARE ALLOWED.
; YOU CAN OMIT THE COMPLETE CIPHERSUITES KEYWORD. IN THIS CASE
; ALL CIPHER SUITES SUPORTED BY THE SSL LIBRARY WILL BE ALLOWED.
; ****************************************************************
CIPHERSUITES =   ; COMMA SEPARATED LIST OF NUMERIC VALUES
           ; C027, ; TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 OPENSSL ONLY
           ; C014, ; TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA  - OPENSSL ONLY
           ; C013, ; TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA  - OPENSSL ONLY
           ; C012, ; TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA - OPENSSL ONLY
           ;   6B, ; TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 - OPENSSL ONLY
           ;   67, ; TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 - OPENSSL ONLY
           ;   39, ; TLS_DHE_RSA_WITH_AES_256_CBC_SHA    - OPENSSL ONLY
           ;   33, ; TLS_DHE_RSA_WITH_AES_128_CBC_SHA    - OPENSSL ONLY
           ;   16, ; SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA   - OPENSSL ONLY
           ;   3D, ; TLS_RSA_WITH_AES_256_CBC_SHA256     - OPENSSL ONLY
           ;   3C, ; TLS_RSA_WITH_AES_128_CBC_SHA256     - OPENSSL ONLY
           ;   3B, ; TLS_RSA_WITH_NULL_SHA256 - DEPRECATED,OPENSSL ONLY
               35, ; TLS_RSA_WITH_AES_256_CBC_SHA
               2F  ; TLS_RSA_WITH_AES_128_CBC_SHA
           ;   0A, ; RSA1024_3DESCBC_SHA                - DEPRECATED
           ;   09, ; RSA1024_DESCBC_SHA                 - DEPRECATED
           ;   08, ; RSA512_DES40CBC_SHA                - DEPRECATED
           ;   02, ; RSA512_NULL_SHA                    - DEPRECATED
           ;   01, ; RSA512_NULL_MD5                    - DEPRECATED
```

図 *10.* メンバー *SKVCSSSL (SSL/TLS* のための *VSE* コネクター・サーバー の構成用*)*

VSE コネクター・サーバー のための日付形式の構成

VSE コネクター・サーバー は、現在の日付形式を *LE/VSE* ランタイム・オプション から取得します。 デフォルトは、**mm/dd/yyyy** (USA 日付形式) です。 z/VSE システムの日付形式を変更した場合、LE/VSE 日付形式も変更する必要があります。 そうでない場合、VSE コネクター・クライアント アプリケーションは、日、月、年に対して正しくない値を取得します。

LE/VSE 日付形式を変更するには、以下を行ってください。

1. このパラメーターをメンバー CEEDOPT (これは、VSE/ICCF ライブラリー 62 に保管されています) で変更します。

   ```
   COUNTRY=((US),OVR),
   ```

2. ジョブ・テンプレート CEEWDOPT (これは、VSE/ICCF ライブラリー 62 に保管されています) をサブミットして、新しい値をカタログに入れます。

また、この例に示すように、VSE コネクター・サーバー の日付形式をオンラインで指定します。

```
// EXEC IESVCSRV,PARM='COUNTRY(DE)/DD:PRD2.CONFIG(IESVCSRV.Z)'
```

以下の詳細については、

- LE/VSE ランタイム・オプションの指定方法
- 使用できる国別コード

「*LE/VSE Customization Guide*」(SD88-7048) を参照してください。

VSE コネクター・サーバー の始動

VSE コネクター・サーバー をスタートアップするには、 VSE/POWER 読み取り待ち行列からスタートアップ・ジョブ STARTVCS をリリースします。 デフォルトでは、サーバーは動的クラス R の区画で実行されます。

STARTVCS は、次のいずれかで POWER 読み取り待ち行列に入れられます。

- z/VSEの初期インストール時
- z/VSEのコールド・スタートアップ時

注**: TCP/IP** の実行が必須です。 このジョブは、TCP/IP のスタートアップを待ちます。TCP/IP の開始方法について詳しくは、「*IBM z/VSE TCP/IP* サポート」を参照してください。

開始されると、VSE コネクター・サーバー は、以下のようにしてポート **2893** 上の着信 TCP/IP トラフィックを listen します。

- サーバーのセキュリティー・サブシステムは RACROUTE 要求を使用して、ログオンおよびリソース権限を調べます。
- サーバーは複数のクライアントを同時に処理できます。

40 ページの図 11 にスタートアップ・ジョブ STARTVCS を示します。

```
* $$ JOB JNM=STARTVCS,CLASS=R,DISP=L
* $$ LST CLASS=A,DISP=D
// JOB STARTVCS START UP VSE CONNECTOR SERVER
// ID USER=VCSRV
*   WAITING FOR TCP/IP TO COME UP
// EXEC REXX=IESWAITR,PARM='TCPIP00'
// LIBDEF *,SEARCH=(PRD2.TCPIPC,PRD2.CONFIG,PRD1.BASE,PRD2.SCEEBASE)
// OPTION SYSPARM='00'
// EXEC IESVCSRV,PARM='DD:PRD2.CONFIG(IESVCSRV.Z)'
/*
/&
* $$ EOJ
```

図 *11.* スタートアップ・ジョブ *STARTVCS (VSE Connector Server* のスタートアップ用*)*

VSE コネクター・クライアント とコネクター・サーバーの間の通信のテスト

VSE コネクター・クライアント および VSE コネクター・サーバー のインストールが完了したら、 Java ベース・コネクターの 2 つのパーツが相互に正しく通信するかどうかを、いくつかのプリコンパイル済みのサンプル Java アプリケーションを実行することでテストできます。

始める前に、以下の項目を確認する必要があります。

- TCP/IP for z/VSE が z/VSE ホスト上で実行されている。
- VSE コネクター・サーバー が z/VSE ホスト上で実行されている。
- ローカル・ワークステーションと z/VSE ホストの間に TCP/IP 接続が確立されている。

VSE コネクター・クライアント のインストールには、Windows および Unix/Linux オペレーティング・システム上で実行されるバッチ・ファイルのセットが組み込まれています。 それらは、VSE コネクター・クライアント をインストールしたディレクトリーの下のサブディレクトリーの **samples** に保管されています。

samples に含まれる各バッチ・ファイルは、 1 つのサンプル *Java* アプリケーション またはサンプル・アプレット を実行します。 サンプルを実行するには、以下の作業を行う必要があります。

1. コマンド・プロンプトを開きます。
2. **samples** サブディレクトリーに移動します。
3. バッチ・ファイルを実行します (パラメーターを入力する必要はありません)。

実行できるすべてのサンプルのリスト (説明付き) を取得するには、以下の作業を行う必要があります。

1. VSE コネクター・クライアント を始動します。
2. このウィンドウの左のフレームにある「**Run examples (**サンプルを実行**)**」をクリックします。

VSE コネクター・サーバー のコマンドのリストの取得

使用できる VSE コネクター・サーバー のコマンドのリストを取得するには、オペレーター・コンソールで **Help** または **?** を入力するだけです。 図 12 に、表示される項目を示します。

```
SYSTEM:  z/VSE               z/VSE 4.3        TURBO (01)       USER:  JSCH
                                                                TIME:  14:25:53
F7-0112 IPN300I Enter TCP/IP Command
msg startvcs,data=?
AR 0015 1I40I  READY
R1 0045 IESC1043I HELP COMMAND
R1 0045   HELP|?                                    PRINTS THIS MESSAGE
R1 0045   STATUS [ALL|CONFIG|CLIENTS|PLUGINS|VSAM] PRINTS STATUS INFORMATION
R1 0045   SENDMSG <USER(S)> <MESSAGE TEXT>          SENDS A MESSAGE TO A USER
R1 0045   SHUTDOWN [NOPROMPT]                      SHUTS DOWN THE SERVER
R1 0045   SETTRACE <TRACEFILE> <TRACELEVEL>        SET TRACING ON/OFF
R1 0045   STOP CLIENT <CLIENT-ID|ALL>              STOPS THE SPECIFIED CLIENT
R1 0045   CLOSE VSAM <SLOT-ID|ALL>                 CLOSES A VSAM CLUSTER


==>

1=HLP 2=CPY 3=END 4=RTN 5=DEL 6=DELS 7=RED 8=CONT 9=EXPL 10=HLD 11=PCUU 12=RTRV
```

図 *12. VSE* コネクター・サーバー が提供するコマンドの表示

VSE コネクター・サーバー のコマンドの入力

VSE コネクター・サーバー に処理させるコマンドを入力するには、以下のコマンド構文を使用する必要があります。

```
msg <jobname>,data=<command>
```

ここで、

- *<jobname>* は VSE コネクター・サーバー スタートアップ・ジョブの実際の名前です。
- *<command>* は 図 12 に示すコマンド文字列のいずれかです。

例えば、次のようになります。

```
msg startvcs,data=status
```

VSE コネクター・サーバー を使用したセキュリティーの維持

Web アプリケーションが VSE Java Beans クラス・ライブラリーを使用して (165 ページの『VSE Java Beans を使用したホストへの接続の例』で説明) z/VSE ホストに接続する際に、 Web アプリケーションは (有効なユーザー ID およびパスワードを指定して) 最初に z/VSE ホストへのログオンを行う必要があります。このログオンが成功すると、Web アプリケーションは提供されたユーザー ID からのアクセス権限を取得します。

VSE コネクター・サーバー は Web アプリケーションから要求を受け取ると、この要求を現在アクティブな VSE セキュリティー・マネージャー (基本セキュリティー・マネージャーまたは外部セキュリティー・マネージャーのいずれか) に渡しま

す。 そこで、セキュリティー・マネージャーは、要求されたリソースまたはデータに Web アプリケーションがアクセスできるかどうかを調べます。

VSE コネクター・サーバー は、構成ファイル SKVCSUSR も使用します。このファイルには、ユーザー ID および IP アドレスについての以下の 2 つのリストが含まれます。

1. VSE コネクター・サーバー への接続を許可されているもの。
2. VSE コネクター・サーバー への接続を許可されていないもの。

これについては、 36 ページの『VSE ライブラリー・メンバー SKVCSUSR - ログオン・アクセスの指定』で説明しています。

サーブレット を作成すると、特殊な z/VSE ユーザー ID を使用することで、エンド・ユーザーを VSE コネクター・サーバー にログオンさせずにそのサーブレットを VSE コネクター・サーバー に接続することができます。 このユーザー ID を使用することにより、サーブレットで要求のタイプを制限でき、データへのアクセスも制限できます。

アプレット を作成するときには、アプレット・コード内にユーザー ID およびパスワードを絶対に「ハードコーディング」しないでください。アプレットが Web ブラウザーにダウンロードされ、Web ブラウザーのキャッシュに保管されると、無許可のユーザー (ハッカー) によってこの情報が表示されてしまうおそれがあります。

TCP/IP for z/VSE には独自のセキュリティー機能もあります。 しかし、これらの機能は、 FTP、HTTP、または Telnet デーモンなどの TCP/IP アプリケーションを使用するときにのみ適用可能です。 VSE コネクター・サーバー は、常時 VSE セキュリティー・マネージャーと通信しています。

注**:** 物理/論理中間層と Web クライアントの間のセキュリティーは SSL/TLS を使用して確立されます。ただし、物理/論理中間層と z/VSE ホストの間には SSL/TLS は使用しません。

第 6 章 VSE Java Beans を介してアクセスするための DL/I の構成

このトピックでは、VSE Java Beans を介してアクセスできるように DL/I システムを構成する方法について説明します。

以下の項目があります。

- 『完了しておくべきホストのインストール・アクティビティー』
- 『ステップ 1: スケルトン SKDLISMP - サンプル・データベースの定義』
- 44 ページの『ステップ 2: CICS TS をカスタマイズする』

DL/I アクセスには、VSEDli、VSEDliPsb、および VSEDliPcb の 3 つの VSE Java Beans が使用されます。これらの VSE Java Beans については、 159 ページの『VSE Java Beans クラス・ライブラリーの内容』を参照してください。

VSE Java Beans を介した DL/I データのアクセスに使用できるコーディングの例については、 179 ページの『VSE Java Beans を使用した DL/I データへのアクセスの例』を参照してください。

完了しておくべきホストのインストール・アクティビティー

このトピックでは、VSE Java Beans を使用して DL/I データにアクセスする前に z/VSE ホストで完了しておく必要があるインストール・アクティビティーについての要約を提供します。

- VSE Java Beans を介した DL/I データのアクセス用に AIBTDLI インターフェースをインストールする必要があります。 AIBTDLI インターフェースを使用するには、以下の作業を行います。
  - DL/I VSE をインストールする必要があります。
  - CICS/DLI システムでは、AIBTDLI インターフェースと合わせて、使用するすべてのデータベース (DBD) を CICS に定義しておく必要があります。
  - CICS/DLI システムには、以下が必要です。
    - DL/I オンライン中核 DLZNUCxx にすべての PSB が定義済みである。
    - アクティブな MPS システム。
  - DL/I タスク終了出口 *DLZBSEOT* (469 ページの『タスクの終了および異常終了の処理』で説明) を SVA 内に常駐させておく必要があります。

ステップ 1: スケルトン SKDLISMP - サンプル・データベースの定義

以下の場合に、スケルトン SKDLISMP (VSE/ICCF ライブラリー 59 から入手可能) を使用して、DL/I サンプル・データベースを定義およびロードします。

- IBM 提供のサンプル・データベースをテストおよびラーニングの目的で使用する場合。

- 以前のインストールでこのデータベースを定義およびロードしなかった場合。

スケルトン SKDLISMP には、以下のジョブが含まれます。

1. STJDBDGN (サンプル・データベース用の DBD を生成する)。
2. STJPSBGN (サンプル・データベース用の PSB を生成する)。
3. STJACBGN (サンプル・データベース用の ACB を生成する)。
4. STJPREOR (事前再編成ユーティリティー)。
5. STJDFINV (サンプル・データベース用のクラスターを定義する)。
6. STJLDCST (サンプル・データベースをロードする)。
7. STJPRRES (解決をプレフィックス)。
8. STJPRUPD (更新をプレフィックス)。

ステップ 2: CICS TS をカスタマイズする

VSE Java Beans を介して DL/I データにアクセスするには、 (以前のインストール・アクティビティーでカスタマイズしていない場合には) 以下の手順で CICS TS - DL/I オンライン・システムをカスタマイズする必要があります。

1. 以下の説明に従って、CICS/DLI オンライン・システムを構成します。
   - 「*DL/I Resource Definition and Utilities*」資料の『Part 6』。
   - 「*DL/I Resource Definition and Utilities*」資料の『CICS - DL/I Tables - Requirements』のトピック。
   - 「*DL/I Release Guide*」の『Migrating to DL/I VSE 1.11 and the CICS Transaction Server for z/VSE 1.1』トピック。
2. CICS FCT にサンプル・データベース STDIDBP およびアクセスしたいその他のデータベースを定義します。
3. サンプル・データベース STDIDBP (// DLBL STDIDBC ...) およびアクセスしたいその他のデータベースにラベルを指定します。
4. 使用するすべての DL/I オンライン・プログラムおよび PSB を組み込むことによって、新規 DL/I オンライン中核 (DLZACT 生成) を作成します。CICS/DLI ミラー・プログラム DLZBPC00 は、DL/I サンプル・データベースにアクセスするために PSB STBICLG に対して許可されている必要があります。 CICS/DLI ミラー・プログラム DLZBPC00 は、他の DL/I データベースにアクセスするその他の PSB に対しても許可しておく必要があります。
5. CICS/DLI オンライン・システム内の並行 DLZBPC00 ミラー・タスクの増分数を計算します。結果に応じて、DLZACT 生成時の MAXTASK および CMAXTSK パラメーターを調整する必要があります。
6. DL/I 出口ルーチン DLZBSEOT を SVA にロードします。
7. MPS システムを始動します。

DLZMPX00 は SVA 適格で、*AIBTDLI* インターフェース 経由の DL/I データへのアクセスに使用されます (DLZMPX00 および AIBTDLI インターフェースの説明については、 460 ページの『AIBTDLI インターフェースの概要』を参照してください)。 AIBTDLI インターフェースは、 DLZMPX00 がすでにあれば SVA から使用するか、DLZMPX00 を区画スペースにロードしてそこから使用します。

第 7 章 VSE スクリプト・コネクターのインストール

このトピックでは、まず、VSE スクリプト・コネクターの使用方法について概説します。その後、VSE スクリプト・コネクターのサーバー・パーツ (VSE スクリプト・サーバー) をインストールする方法について説明します。

VSE スクリプト・コネクターのクライアント・パーツ (VSE スクリプト・クライアント) はここではインストールされません が、以下のいずれかにできます。
- ユーザー作成の Java アプリケーション。
- ユーザー作成の非 Java アプリケーション。
- 既存のオフィス製品。ワード・プロセッシング・プログラムまたはスプレッドシート・プログラムなど。

このトピックに含まれるのは次のとおりです。
- 『VSE スクリプト・コネクター の概要』
- 46 ページの『ステップ 1: インストール・ファイルをダウンロードしてインストールを実行する』
- 48 ページの『ステップ 2: VSEScriptServer プロパティー・ファイルを構成する』
- 50 ページの『ステップ 3: 接続プロパティー・ファイルを構成する』

注:
1. VSE スクリプト・サーバーは Java 対応プラットフォームで実行する必要があります。
2. クライアント・パーツ (VSE スクリプト・クライアント) は Java 対応プラットフォームまたは非 Java 対応プラットフォームのいずれかで実行できます。
3. VSE スクリプト・コネクターを使用するために独自の Java コードを作成する必要はありません。 IBM 提供の VSE スクリプト言語を使用して独自の VSE スクリプトを作成するだけです。

関連トピック:
- 551 ページの『第 27 章 非 Java アクセスのための VSE スクリプト・コネクターの使用』

VSE スクリプト・コネクター の概要

27 ページの『Java ベース・コネクター の概要』に記載したように、 VSE Java Beans によって、*Java* プラットフォーム 上で実行されるいずれの Java プログラム (サーブレット、アプレット、EJB など) からも z/VSE ホストに直接アクセスできます。 さらに、*VSE* スクリプト・コネクターを使用して非 *Java* プラットフォームからも z/VSE ホスト・データにアクセスできます。 (ほとんどの Java プラットフォームから z/VSE ホスト・データにアクセスすることもできますが) この点が VSE スクリプト・コネクターを使用する主な利点になります。

VSE スクリプト・コネクターは、Java ベース・コネクターのパーツ として提供されます。これは、3 層環境でのみ使用でき (10 ページの『3 層環境の概要』で説明)、以下のもので構成されます。

- Java または非 Java プラットフォームで実行する VSE スクリプト・クライアント。次のいずれかにできます。
  - ユーザー作成の Java アプリケーション (例えば、Web サービス)。
  - ユーザー作成の非 Java アプリケーション (例えば、Windows C プログラム、 Windows CGI プログラム、または COBOL アプリケーション)。
  - ワード・プロセッシング・プログラムまたはスプレッドシート・プログラムなどのオフィス製品 (例えば、Lotus 1-2-3 または Lotus WordPro)。例えば、VSE スクリプトの呼び出しには Visual Basic スクリプトを使用します。
- 3 層環境の物理/論理中間層で実行する VSE スクリプト・サーバー。VSE スクリプト・ファイルを解釈して実行します。
- プログラミング解説書を含むオンライン・ドキュメンテーション。

VSE スクリプト・コネクターの一般的な動作方法は以下のとおりです。

1. VSE スクリプト・クライアントは、VSE スクリプトを呼び出して、z/VSE ホストに保管されているデータへ要求を出します。 これらの VSE スクリプト (バッチ) ファイルには、特殊なプログラム言語である VSE スクリプト言語を使用して記述されたステートメントが含まれます。 VSE スクリプト言語は、いずれの環境でも使用できます (Visual Basic スクリプトでも可能です)。
2. Java 対応物理/論理中間層プラットフォームで実行する VSE スクリプト・サーバーは、VSE スクリプト・ファイルを読み取り、解釈して、VSE Java Beans 要求に変換します。 VSE スクリプト・サーバーは、VSE Java Beans を使用して z/VSE ホストで実行中の VSE コネクター・サーバー に接続し、VSE Java Beans 要求を転送します。
3. VSE コネクター・サーバー は、必要な z/VSE データおよび関数にアクセスし、返信を VSE スクリプト・サーバーに戻します。
4. VSE スクリプト・サーバーは、VSE スクリプト・クライアントが使用できるフォーマットにデータを変換し、VSE スクリプト・クライアントにデータを戻します。

ステップ 1: インストール・ファイルをダウンロードしてインストールを実行する

VSE スクリプト・サーバーは Java 対応プラットフォームにインストールします。

注: 始める前に、VSE スクリプト・サーバーをインストールしようとする開発プラットフォームに Java Development Kit (JDK) 1.5 以降がすでにインストールされていなければなりません。 JDK 1.5 以降をインストールしていない場合のインストール方法について詳しくは、 24 ページの『Java のインストールと構成』を参照してください。

ステップ 1.1: VSE スクリプト・サーバーのコピーを取得する

VSE スクリプト・サーバー のコピーを取得するには、以下のいずれから取得するかを決定する必要があります。

- インターネットから

- Extended Base Tape から VSE コネクター・ワークステーション・コード・コンポーネントをインストールすることによって

インターネットから VSE スクリプト・サーバー を取得するには、以下を行ってください。

1. Web ブラウザーを起動して次の URL に進みます (英語のみ)。

   ```
   http://www.ibm.com/systems/z/os/zvse/downloads/
   ```

2. VSE スクリプト・サーバー のセクションから、VSE スクリプト・サーバー をインストールするディレクトリーにファイル **vsescript*nnn*.zip** をダウンロードします。 注: *nnn* は現行 VSE バージョン (**vsescript620.zip**) です。

VSE スクリプト・サーバー を *VSE* コネクター・ワークステーション・コード・コンポーネントをインストールすることによって 取得するには、以下を行ってください。

1. Extended Base Tape から VSE コネクター・ワークステーション・コード・コンポーネントをインストールします。このコンポーネントのインストール後、VSE スクリプト・サーバーの VSE コネクター・ワークステーション・コード **iesscrpt.w** が、z/VSE サブライブラリー PRD2.PROD に保管されます。
2. TCP/IP for z/VSE の FTP (ファイル転送プログラム) ユーティリティーを使用して、VSE スクリプト・サーバー をインストールしたいディレクトリーに **iesscrpt.w** をダウンロードします。

注:

1. **iesscrpt.w** はバイナリー でダウンロードする必要があります。
2. UNIX モードがオフ になっていることを確認してください。 オフにしないと、バイナリー を指定しても、**iesscrpt.w** は ASCII モードでダウンロードされます。 *Unix* モード は VSE FTP デーモンのパラメーターの 1 つです。 FTP クライアントの中には、UNIX モードを強制的に オンにするものがあります。 以下の例に、バッチ FTP クライアントを使用して正常に **iesscrpt.w** を転送する方法を示します。 UNIX モードが設定される場所は太字で示しています。

```
c:¥temp>ftp 9.164.155.2
Connected to 9.164.155.2.
220-TCP/IP for VSE -- Version 01.05.F  -- FTP Daemon
    Copyright (c) 1995,2006 Connectivity Systems Incorporated
220 Service ready for new user.
User (9.164.155.2:(none)): sysa
331 User name okay, need password.
Password:
230 User logged in, proceed.
ftp> cd prd2
250 Requested file action okay, completed.
ftp> cd prod
250 Requested file action okay, completed.
ftp> binary
200 Command okay.
ftp> get iesscrpt.w
200 Command okay.
150-File: PRD2.PROD.IESSCRPT.W
    Type: Binary Recfm: FB Lrecl:    80 Blksize:      80
    CC=ON  UNIX=OFF RECLF=OFF TRCC=OFF CRLF=ON  NAT=NO
150 File status okay; about to open data connection
226-Bytes sent:       2,378,200
    Records sent:        29,728
```

```
    Transfer Seconds:         8.26 (    290K/Sec)
    File I/O Seconds:         1.97 (  1,548K/Sec)
226 Closing data connection.
2378200 bytes received in  8,56 seconds (277,82 Kbytes/sec)
ftp> bye
221 Service closing control connection.
c:¥temp>ren iesscrpt.w vsescript.zip
```

ステップ 1.2: VSE スクリプト・サーバーのインストールを実行する

VSE スクリプト・サーバー のインストールを実行するには、以下を行います。

1. ファイル **vsescript.zip** を unzip します。これには以下のファイルが入っています。
   - setup.jar (VSE スクリプト・コネクター・コードを含む)
   - setup.bat (Windows 用インストール・バッチ・ファイル)
   - setup.cmd (OS/2 用インストール・バッチ・ファイル)
   - setup.sh (Linux/Unix 用インストール・スクリプト)
2. (ファイルをダブルクリックして) オペレーティング・システム・プラットフォームに適用可能なバッチ・ファイルを開始します。
3. インストール・プロセスが開始され、さまざまなインストール・メニューによるガイドが表示されます。
4. HTML ベースのドキュメンテーションにアクセスするには、ここで Web ブラウザーを使用して **/doc** サブディレクトリーにあるファイル **server.html** をオープンすることができます。
5. SSL/TLS を使用するには、以下の作業を実行済みである必要があります。
   - TCP/IP for z/VSE での SSL/TLS を有効にする。
   - 必要な鍵および証明書を作成する。

   注**:** VSE スクリプト・サーバーは、*z/VSE 4.2* 以降からの SSL/TLS 暗号化接続をサポートしています。このサポートは、VSE スクリプト・クライアントから VSE スクリプト・サーバーへの接続および VSE スクリプト・サーバーから z/VSE への接続のためです。

ステップ 2: VSEScriptServer プロパティー・ファイルを構成する

VSE スクリプト・サーバーのプロパティー・ファイルは **VSEScriptServer.properties** という名前で、テキスト・エディターを使用して編集できるテキスト・ファイルです。

コメント行には、最初の桁に # 文字が付いています。

VSEScriptServer.properties に定義する設定は次のとおりです。

messages= on|**off**
: **messages= on** を定義すると、すべてのメッセージが印刷されます。
messages= off を定義すると、メッセージは印刷されません (「静止モード」がアクティブになります)。

listenport = *TCP/IP* ポート番号
: VSE スクリプト・サーバーが要求の listen に使用するポート番号。

maxconnections = 数値
VSE スクリプト・クライアントから同時に接続できる最大数。

scriptdirectory = ./scripts
スクリプトが含まれるルート・ディレクトリー。

connectionconfig = Connections.properties
接続構成ファイルの名前 (50 ページの『ステップ 3: 接続プロパティー・ファイルを構成する』で説明しています)。

codepage
VSE スクリプト・クライアントのコード・ページ。指定しなかった場合は、現行のデフォルト・コード・ページが使用されます。

traceon
サーバー始動中に指定のトレース域を使用可能にします。
INSTRUCTION、PARAMETER、CONDITION のいずれかを指定できます。詳細については、 556 ページの『VSE スクリプト言語トレース・サポート』を参照してください。

sslversion[1]
SSL または TLS を指定できます。

clientauthentication[1]
クライアント認証が `true` のとき、VSE スクリプト・サーバーでは VSE スクリプト・クライアントに証明書の送信を要求し、その証明書を検査します。クライアント認証が `false` であるか、指定されていないとき、クライアント認証は実行されません。

keyringfile[1]
鍵リング・ファイルのファイル名。

keyringpwd[1]
鍵リング・ファイルを開くためのパスワード。

ciphersuites[1]
VSE スクリプト・サーバーが受け入れる、暗号スイートのリスト (コンマ区切り)。このプロパティーを指定しなかった場合は、サポートされるすべての暗号スイートを使用できます。

デフォルトとは異なるプロパティー・ファイルを使用する場合は、次のコマンドを使用してパラメーターとしてプロパティー・ファイルのファイル名を指定する必要があります。

```
java com.ibm.vse.script.VSEScriptServer MyPropertiesFile.properties
```

1. VSE スクリプト・クライアントと VSE スクリプト・サーバーの間の SSL 暗号通信用です。 SSL/TLS を使用するには、必要な鍵および証明書を作成済みである必要があります。

ステップ 3: 接続プロパティー・ファイルを構成する

VSE スクリプト・サーバーから z/VSE ホストへの各接続を、プロパティー・ファイルに含まれる 5 つの プロパティー (connection.1.name から connection.1.password まで) のセットを使用して定義します。 これは、テキスト・エディターを使用して編集できるテキスト・ファイルです。 接続の使用方法の概説については、 552 ページの図 186 を参照してください。

次に、Connections プロパティー・ファイルの例を示します。

```
#Connection.properties

connection.1.name=vsecon
connection.1.ip=9.164.155.2
connection.1.port=2893
connection.1.userid=fran
connection.1.password=mypasswd

connection.2.name=vsefran
connection.2.ip=9.164.155.95
connection.2.port=2893
connection.2.userid=sysa
connection.2.password=mypasswd

 ...
connection.timeout=100
connection.2.sslversion=SSL
connection.2.ciphersuites=TLS_RSA_WITH_AES_256_CBC_SHA,TLS_RSA_WITH_AES_128_CBC_SHA
connection.2.keyringfile=keyring.pfx
connection.2.keyringpwd=ssltest
connection.2.logonwithcert=no
 ...
connection.timeout=100
 ...
```

コメント行には、最初の桁に # 文字が付いています。

それぞれの 接続ごとに **Connections.properties** に定義する設定は、次のとおりです。

name = 名前
: z/VSE ホストに指定する論理名。 VSE スクリプトは、z/VSE ホストにアクセスする際にこの名前を参照します。 **name** の使用例については、560 ページの『ステップ 4: サンプル VSE スクリプトの変更』を参照してください。

ip = *IP* アドレス
: VSE スクリプト・サーバーを接続する z/VSE ホストの TCP/IP アドレス。

port = 数値
: VSE コネクター・サーバー が着信要求の listen に使用するポート番号。

userid = ユーザー *ID*
: z/VSE ホストへの接続の構築に VSE スクリプト・サーバーが使用する z/VSE ユーザー ID。

password = パスワード
: VSE スクリプト・サーバーから z/VSE ホストへの接続に割り当てるパス

ワード。 このパスワードは、VSE スクリプト・サーバーの初期スタートアップ時にプロパティー **connection.n.encpassword** を使用して暗号化されて保管されます。

sslversion[2]= SSL30 | TLS31
指定するときは、他の SSL 固有のプロパティー (下記) も指定する必要があります。

ciphersuites[2] = 暗号スイートのリスト **(**コンマ区切り**)**
z/VSE では現在、以下の暗号スイートをサポートしています。

SSL_RSA_WITH_NULL_MD5 (非推奨)
SSL_RSA_WITH_NULL_SHA (非推奨)
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA (非推奨)
SSL_RSA_WITH_DES_CBC_SHA (非推奨)
SSL_RSA_WITH_3DES_EDE_CBC_SHA (非推奨)
TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA

OpenSSL の使用時に追加でサポートされる暗号スイートは以下のとおりです。

TLS_RSA_WITH_NULL_SHA256 (非推奨)
TLS_RSA_WITH_AES_128_CBC_SHA256
TLS_RSA_WITH_AES_256_CBC_SHA256
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_DHE_RSA_WITH_AES_128_CBC_SHA
TLS_DHE_RSA_WITH_AES_256_CBC_SHA
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256

keyringfile[2] = 名前
鍵および証明書を含んだ、鍵リング・ファイルのファイル名。

keyringpwd[2] = パスワード
鍵リング・ファイルを暗号化するためのパスワード。このパスワードは、VSE スクリプト・サーバーの初期スタートアップ時にプロパティー `connection.n.enckeyringpwd` を使用して暗号化されて保管されます。

logonwithcert[2] = true | false
`true` を設定すると、VSE スクリプト・クライアントの証明書が z/VSE へのサインオンに使用されます。このためには、z/VSE 上でクライアント認証が有効にされている必要があります。

さらに、**Connections.properties** に以下のグローバル設定を定義できます。

2. VSE スクリプト・サーバーと z/VSE システムの間の SSL 暗号通信用です。 SSL/TLS を使用するには、必要な鍵および証明書を作成済みである必要があります。

connection.timeout = 秒数
プール内の (z/VSE ホストへの) 未使用接続をクローズして破棄するまでの秒数。

第 8 章 VSAM リダイレクター・コネクターのインストール

このトピックでは、VSAM リダイレクター・コネクターをインストールしてインプリメントする方法について説明します。

以下の項目があります。

- 『VSAM リダイレクター・コネクター の概要』
- 58 ページの『VSAM 統合に関する考慮事項』
- 58 ページの『VSAM リダイレクター・ クライアント/VSAM 取り込み 出口の構成』
- 74 ページの『VSAM リダイレクター・サーバーのインストール』
- 79 ページの『IBM 提供の VSAM リダイレクター・ハンドラーの使用』
- 81 ページの『IBM 提供の VSAM リダイレクター・ローダーの使用』
- 82 ページの『Db2 関連要求処理プログラムの IBM 提供の例』

VSAM リダイレクター・コネクター の概要

VSAM リダイレクター・コネクターは、VSE プログラムで以下を処理します。

- リモート・データベースまたはファイル・システムと同期化する VSAM データ。
- データのすべてがリモート・プラットフォームに移動された VSAM ファイル。

VSAM リダイレクター・コネクターを使用することで、以下のことが可能です。

- VSAM データを別のファイル・システムまたはデータベースへマイグレーションできます。
- データを異なるシステム上で VSE VSAM データと同期させることができます。
- VSE プログラムが、他のファイル・システムまたはデータベース上のデータをトランスペアレントに処理できます。
- VSAM データへの変更は、取り込まれて、後の処理のために一時的に z/VSE システムに保管できます。

これで、以下のような既存の z/VSE ホスト・プログラムでは、修正および再コンパイルを行わずにマイグレーションした VSAM データを処理できます。

- 任意の言語 (COBOL、PL/I、アセンブラー) で記述されている
- バッチまたは CICS プログラム

これらの z/VSE ホスト プログラムを修正および再コンパイルする必要はなく、マイグレーションした VSAM データを処理できます。 VSAM リダイレクター・コネクターは、すべての接続およびデータ変換を管理します。

VSAM リダイレクター・コネクターは、以下により構成されます。

- 同期リダイレクトの *VSAM* リダイレクター・ クライアント。*z/VSE* ホストにインストールされます。 VSAM 要求の通信およびリダイレクトを行います。

- 非同期リダイレクトの *VSAM* 取り込み 出口。*z/VSE* ホストにインストールされます。 これによって特定の VSAM ファイルに行われた変更を取り込み、以下に保管します。
  - 別の VSAM ファイル (VSAM デルタ・クラスター)。
  - WebSphere MQ キュー。
- *VSAM* リダイレクター・サーバー。 55 ページの図 13 に示すように各リモート *Java* プラットフォーム にインストールされます。 これは、以下のような Java プログラムになります。
  1. コネクション処理を管理します。
  2. データ変換を実行します。
  3. 別のファイル・システム *VSAM* リダイレクター・ハンドラー (単に、リダイレクター・ハンドラー とも言う) へのインターフェースを構築します。
  4. VSAM データ・セットのマイグレーション前に生成されていたものと同じエラー・メッセージを生成します。 このため、アプリケーション・プログラムのエラー・メッセージ処理に変更を加える必要はありません。
- *VSAM* リダイレクター・ローダー (単にリダイレクター・ローダー とも言う)。VSAM データのロードおよび処理に使用されるリモート・プラットフォーム上の Java プログラムです。IBM では、以下のリダイレクター・ローダーを提供しています。
  - 基本リダイレクター・ローダー。これは、任意の VSAM ファイルからデータをロードして、そのデータを、指定したリダイレクター・ハンドラーで処理するとき (データベースに保管する場合など) に使用します。
  - デルタ・リダイレクター・ローダー。これは、VSAM デルタ・クラスターからのデータから取り込んだデータをロードして、そのデータを指定したリダイレクター・ハンドラーによって処理します。
  - *MQ* リダイレクター・ローダー 。これは、ワークステーション上の WebSphere MQ から「MQ トリガー」として呼び出され、指定したリダイレクター・ハンドラーによって取り込んだデータを処理します。
- グラフィカル・ユーザー・インターフェース。これは、`DBHandler` リダイレクター・ハンドラーのデータ・マッピングを、手動で、または COBOL コピーブックをインポートして構成したいときに使用します。グラフィカル・ユーザー・インターフェースを使用して、対応するデータベース表を作成することもできます。

リダイレクター・ハンドラー は、*Java* プラットフォーム にも保管され、共通インターフェースを持っています。 これらのプログラムは、処理するファイル・システムに固有のものです。 どんな接続の場合でも、ファイルおよび要求に関する情報がリダイレクター・ハンドラーに送信されます。

同期データ・リダイレクト を使用して、例えばリモート・システムにある Db2 表で、VSAM データをマイグレーションまたは同期化できます。すると、これらの VSE プログラムを変更せずに VSE プログラムで、このデータを処理できます。リモート・システムでは、*Java* ハンドラー によって、リモート・システムにある特定のファイル・システムまたはデータベースにアクセスできます。 例えば、VSAM データをリモート・システム上にある Db2 表にマイグレーションすると、ご使用の VSE プログラムを変更せずにそのデータを処理できます。

図 13 には、VSAM リダイレクター・コネクターを使用して、z/VSE ホストで実行しているアプリケーションからの VSAM 要求を Java プラットフォームに保存されている Db2 データベースへ同期で リダイレクトする方法の概要が示されています。Java プラットフォームでは、以下の IBM 提供のリダイレクター・ハンドラーを使用できます。

- DB2Handler
- DBHandler
- CSVFileHandler

各ハンドラーの特性は、79 ページの『IBM 提供の VSAM リダイレクター・ハンドラーの使用』 に示されています。

各要求処理プログラムは、上記の要求ごとにリモート・データに必要な処理を決定できます。

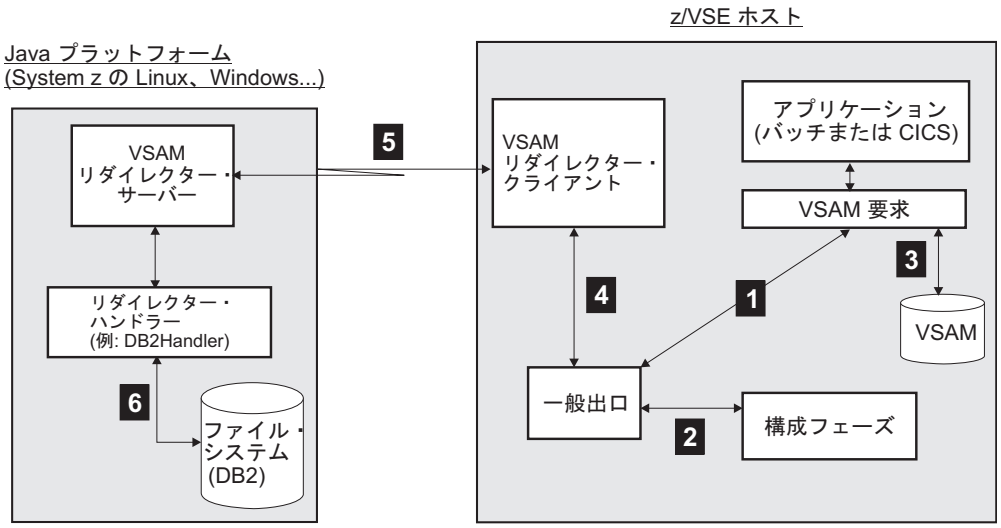


図 *13.* 同期データ・リダイレクトの使用方法

図 13 に示す一般的な処理は、以下のとおりです。

1 z/VSE ホストで実行されるアプリケーションは VSAM コマンド (例えば、 VSAM ファイルのオープン) を発行します。 要求は一般出口 (IKQVEX01.PHASE) に渡されます。

2 一般出口は、VSAM ファイルをリダイレクトするようにセットアップされているかどうかを調べます。 このために、構成フェーズ (IESRDCFG.PHASE) を調べます。

3 VSAM ファイルが別の Java プラットフォームにリダイレクトされていない 場合 (そのため、z/VSE ホスト上の VSAM レコードに保管されたままである)、一般出口は VSAM ファイルがリダイレクトされなかったことを示して戻ります。 また、一般出口がこの VSAM ファイルに対していずれの要求も再度呼び出すことができないことも示されます。 通常の VSAM 処理が続行されます。

4 VSAM ファイルが別の Java プラットフォームにリダイレクトされた 場合、一般出口 (IESREDIR.PHASE) は VSAM リダイレクター・ クライアントを呼び出します。

5 VSAM リダイレクター・ クライアントは Java プラットフォーム上で実行中の VSAM リダイレクター・サーバーへの接続を確立し、VSAM ファイル要求と一緒に任意のデータ (VSAM レコードの内容など) を VSAM リダイレクター・サーバーに転送します。

6 VSAM リダイレクター・サーバーは、構成フェーズで指定されたリダイレクター・ハンドラーを使用して、ターゲット・ファイル・システムまたはターゲット・データベースへのアクセスを実行します。 55 ページの図 13 では、リダイレクター・ハンドラーとして *DB2Handler* が指定されています (VSAM リダイレクター・サーバーのインストール時に IBM が指定)。*DB2Handler* は Db2 データベースへのアクセスを実施します。

VSAM データの非同期リダイレクト は、*VSAM* 取り込み 出口によって提供されます。それによって以下が行えます。
1. 特定の VSAM ファイルに変更を取り込みます。
2. 後の処理のためのこれらの変更を保管します。

使用できるオプションは 2 つあります。
- VSAM クラスター (VSAM デルタ・クラスター) に変更を保管します。
- WebSphere MQ 待ち行列に変更を保管します。

取り込んだ変更は「デルタ・レコード」またはメッセージとして書き込まれます。以下が含まれます。
- 変更されたレコードのデータ。
- レコードが変更された日時 (タイム・スタンプ) および 項目 (区画、フェーズ名、元の値など) についての情報。

保管した変更は、リモート・プラットフォームで実行している Java プログラムによって、Java ベース・コネクターを通して後でアクセスできます。 ローダー・プログラムは、VSAM デルタ・クラスターから取り込んだデータのロードと処理 (例えば、変更をデータベースに適用する) のためにも提供されます。

57 ページの図 14 には、以下についての概要が示してあります。
1. VSAM リダイレクター・コネクターを使用して、z/VSE ホストの特定の VSAM ファイルに変更を取り込みます。
2. Java ワークステーションで実行されているリダイレクター・ローダー・プログラムで、次に *Java* ベース・コネクターを使用して、この取り込んだデータを処理します。

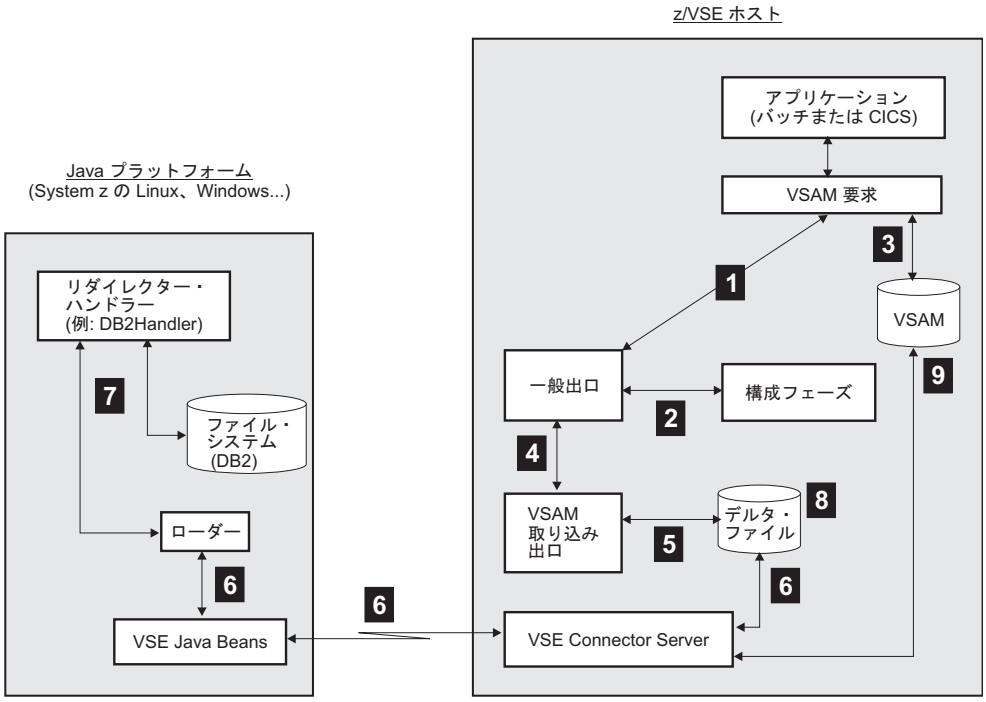


図 *14.* 非同期データ・リダイレクトの使用方法

55 ページの図 13 に示す一般的な処理は、以下のとおりです。

1 z/VSE ホストで実行されるアプリケーションは VSAM コマンド (例えば、 VSAM ファイルのオープン) を発行します。 要求は一般出口 (IKQVEX01.PHASE) に渡されます。

2 一般出口は、VSAM ファイルをリダイレクトするようにセットアップされているかどうかを調べます。 このために、構成フェーズ (IESRDCFG.PHASE) を調べます。

3 VSAM ファイルが別の Java プラットフォームにリダイレクトされていない場合、一般出口は VSAM ファイルがリダイレクトされなかったことを示して戻ります。また、一般出口がこの VSAM ファイルに対していずれの要求も再度呼び出すことができないことも示されます。 通常の VSAM 処理が続行されます。

4 VSAM ファイルに行われた変更 (UPDATE、INSERT、または DELETE) を VSAM 取り込み 出口 (IESVSCAP.PHASE) を使用して取り込みます。

5 変更が VSAM デルタ・クラスターに書き込まれます。

6 リダイレクター・ローダーが VSAM デルタ・クラスターを読み取ります。

7 リダイレクター・ローダーはリダイレクター・ハンドラーを呼び出し、デルタ・ファイルの変更をデータベース、またはファイル・システムに適用できるようにします。

8 完了時に、リダイレクター・ローダーは処理済みのレコードを VSAM デルタ・クラスターから削除します。

9 (オプション)。 リダイレクター・ローダーは、VSAM ファイルの内容をリモートのデータベースまたはファイル・システムにコピーします (例えば、初期ロードのため)。

VSAM 統合に関する考慮事項

VSAM 内部処理 (POINT to END OF FILE など) について、VSAM リダイレクター・ クライアントがその処理を実行できるように VSAM への変更が加えられました。しかし、リダイレクトされたファイルのオリジナルの VSAM クラスターは、まだ z/VSE ホスト上にあります。 この中には、ダミー・レコードも入れる必要があります (DITTO ユーティリティーなどを使用して挿入できます)。

すべての VSAM 要求をリダイレクトするには、出口フェーズ (IKQVEX01.PHASE) が -1 の戻りコードを戻されなければなりません。 これにより、このファイルに対する VSAM 処理が必要ないことを VSAM に示します。

出口フェーズが構成フェーズ (IESRDCFG.PHASE) を開けない場合、または TCP/IP 接続が使用できない場合、出口フェーズは以下の 2 つのエラー定義を使用してこれを VSAM に報告します。

- 出口フェーズが -3 を VSAM に戻すと、DDNAME NOT FOUND エラー・メッセージに変換されます。これは、指定された Java プラットフォームに出口フェーズを接続できないことを示します。
- 2 番目のエラー・コードは -4 で、UNABLE TO CDLOAD エラーに変換されます。これは、出口フェーズが VSAM リダイレクター・ クライアントまたは構成フェーズのいずれかをロードできなかったことを意味します。

処理および戻りコードの変更について詳しくは、VSAM リダイレクター・サーバーで提供されているオンライン・ドキュメンテーションを参照してください (30 ページの『オンライン・ドキュメンテーション・オプションの使用』を参照)。

VSAM リダイレクター・ クライアント/VSAM 取り込み 出口の構成

VSAM リダイレクター・ クライアント、VSAM 取り込み 出口、および関連プログラムは、z/VSE のインストール中に *z/VSE* ホスト に自動的にインストールされます。

VSAM リダイレクター・ クライアントおよび VSAM 取り込み 出口は、以下の PHASE ファイルで構成されます。

- IKQVEX01.PHASE (一般出口フェーズ)
- IESREDIR.PHASE (*VSAM* リダイレクター・ クライアント・フェーズ)
- IESVSCAP.PHASE (*VSAM* 取り込み 出口フェーズ)
- IESRDCFG.PHASE (構成フェーズ)
- IESRDANC.PHASE (リダイレクター・アンカー・フェーズ)
- IESRDLDA.PHASE (新規構成の登録に使用されるフェーズ)

IKQVEX01 は、VSAM OPEN 要求が出されるごとに呼び出されます。 構成フェーズによっては、そのとき、VSAM クラスターをリダイレクトするかどうかが IKQVEX01 によって決定されます。

- VSAM クラスターをリダイレクトする 場合、IKQVEX01 は必要な出口 (例: IESREDIR または IESVSCAP) をロードして開始します。これらの出口のパラメーターを構成フェーズ IESRDCFG.PHASE に指定する必要があります (詳しくは『ステップ 1: z/VSE で VSAM リダイレクター・ クライアント/VSAM 取り込み 出口を使用可能にする』 を参照)。
- VSAM クラスターをリダイレクトしない 場合、通常の VSAM 処理が行われます。

この処理の概略については、 55 ページの図 13 に示しています。

VSAM リダイレクター・ クライアントを構成するには、以下のステップを行う必要があります。

- 『ステップ 1: z/VSE で VSAM リダイレクター・ クライアント/VSAM 取り込み 出口を使用可能にする』
- 60 ページの『ステップ 2: リダイレクト・モードを決定する』
- 68 ページの『ステップ 3 (オプショナル): VSAM データを転送する』
- 68 ページの『ステップ 4: 構成フェーズを作成する』

注**:** 高速サービス・アップグレードに関する考慮事項 高速サービス・アップグレード (FSU) を行う場合、構成ジョブ SKRDCFG を実行して、ライブラリー PRD2.CONFIG 内の IKQVEX01.PHASE を更新する必要があります。

ステップ 1: z/VSE で VSAM リダイレクター・ クライアント/VSAM 取り込み 出口を使用可能にする

VSAM リダイレクター・コネクターは既存の VSAM データ・アクセス (VDA) 出口に基づいています。 VDA 出口は、ダミー出口フェーズ IKQVEX01.PHASE によって提供されます。これは、ライブラリー IJSYSRS.SYSLIB に配布されています。

VSAM リダイレクター・ クライアントと VSAM 取り込み 出口も VDA 出口を使用します。 このため、IKQVEX01.PHASE に加えた既存の変更が上書きされないようにするため、VSAM リダイレクター・ クライアント と VSAM 取り込み 出口に属するすべてのフェーズがライブラリー PRD1.BASE で配布されています。

VSAM リダイレクター・ クライアント、または VSAM 取り込み 出口を使用可能にするには、ライブラリー 59 内のスケルトン SKRDCFG を使用して VSAM リダイレクター・ クライアントと VSAM 取り込み 出口を構成する必要があります。このスケルトンを使用して、以下の作業を実行します。

1. メンバー IESRDCFG.PHASE のアセンブル/リンクを行い、ライブラリー PRD2.CONFIG に保管します。
2. IESRDCFG.PHASE を SVA にロードします (オプショナル)。
3. IESVEX01.PHASE を IKQVEX01.PHASE の名前でライブラリー PRD2.CONFIG にコピーし、出口フェーズをアクティブにします。
4. IKQVEX01.PHASE を SVA にロードします (オプショナル)。

   注**:** 通常、オリジナルの IKQVEX01.PHASE が SVA にロードされています。 VSAM リダイレクター・ クライアントを使用可能にするには、PRD2.CONFIG からロードした新規フェーズによって SVA 内のものと置き換える必要があります。 あるいは、z/VSE システムを再 IPL して IKQVEX01 フェーズを置き換えることもできます。

上記のステップを完了した後で、システムを稼働しながら (すなわち、VSAM アプリケーションを再始動せずに) 新しい構成をアクティブにする場合には、以下の作業を行う必要があります。

5. IESRDANC.PHASE を SVA にロードします (これを行うのは IPL 1 回につき一度だけです)。
6. プログラム IESRDLDA を実行します。 ただし、IESRDCFG.PHASE が (上記のとおりに) SVA にすでにロードされている 必要があります。ここで、プログラム IESRDLDA が現行構成である新しい IESRECFG のコピーをアクティブにすると、変更が即時にアクティブになります。 ただし、VSAM ファイルは、すでにオープンしている場合には変更されません。 オープンしている VSAM ファイルへの変更を有効にするには、そのファイルをクローズしてから再オープンする必要があります。

IPL の実行後、VSAM リダイレクター・ クライアントの構成を再びアクティブ化する必要があります。そのためには、USERBG.PROC を更新して以下を行ってください。

1. IESRDCFG.PHASE を SVA にロードします。
2. IKQVEX01.PHASE を PRD2.CONFIG から SVA にロードします。
3. IESRDANC.PHASE を SVA にロードします。
4. プログラム IESRDLDA を実行します。

VSAM リダイレクター・ クライアント (フェーズ IESREDIR) は、通信プロトコルとして TCP/IP を使用します。 そのため、VSAM リダイレクター・コネクターで使用するアプリケーションを必要に応じて変更しなければなりません。 以下のステートメントを追加する必要があります。

```
// OPTION SYSPARM='nn'
```

これらのジョブのため、JCL (ジョブ制御言語) に追加します。 値 'nn' (システム ID) は以下の場所にあります。

```
// EXEC IPNET,SIZE=IPNET,PARM='ID=nn,INIT=... '
```

TCP/IP スタートアップ・ジョブのステートメント。

注: システム ID のデフォルトは '00' の値になります。 このデフォルトを受け入れる場合には、ステートメント // OPTION SYSPARM='00' を追加する必要はありません。

ステップ 2: リダイレクト・モードを決定する

このトピックでは、IBM が提供するリダイレクト・モードの出口について説明します。

IBM は、リダイレクト・モードの 2 つ の出口を提供します。

- VSAM リダイレクター・ クライアント (IESREDIR)。これは、同期 リダイレクト用の出口です。
- VSAM 取り込み 出口 (IESVSCAP)。これは、非同期 リダイレクト用の出口です。

VSAM リダイレクター・ クライアントに使用できるリダイレクト・モード

VSAM リダイレクター・ クライアント (IESREDIR) の場合、使用できるモードが *2* つあります。

1. 別のプラットフォームにあるデータで作業する。
2. 既存の VSAM データと別のプラットフォームにあるデータを同期させる。

操作のモードを設定するには、構成フェーズ内の OWNER パラメーターを使用します。 詳細については、 68 ページの『ステップ 4: 構成フェーズを作成する』を参照してください。

注**:**

1. 各リダイレクト・モードごとに、z/VSE システムに VSAM クラスターを定義する必要があります。このため、このクラスターに対して VSAM OPEN 要求が実行されるときに、VSAM リダイレクター・ クライアントは VSAM クラスターから以下の情報を取得しなければなりません。
   - クラスター・タイプ
   - キー位置
   - キーの長さ
   - レコードの最大長
2. (Java プラットフォーム上で) リダイレクトされたファイルをオープンして READ 処理を行うときに、オリジナルの VSAM クラスターは z/VSE ホストに定義したままにし、少なくとも 1 つの「ダミー」レコードを入れておく必要があります。 そうしないと、OPEN 要求を処理するときに VSAM エラーが起こります。
3. READ 要求は、リモートの Java プラットフォームへのデータ転送を発生させません。

モード **1.** 別のプラットフォームにあるデータの処理**:** このリダイレクト・モード (OWNER=REDIRECTOR) を使用すると、VSAM データを処理するプログラムは VSAM アクセス操作を行いません。 すべての要求が VSAM リダイレクター・ クライアントにリダイレクトされ (**iesredir.phase**)、そこから Java プラットフォームで実行中の VSAM リダイレクター・サーバーに接続します。 そこで、VSAM リダイレクター・サーバーは要求を実行します。

注**:** これらの出口のいずれかで OWNER が REDIRECTOR に設定されている場合には、出口をチェーニングできません。

62 ページの図 15 に、VSAM PUT の実行時の制御のフローを示します。

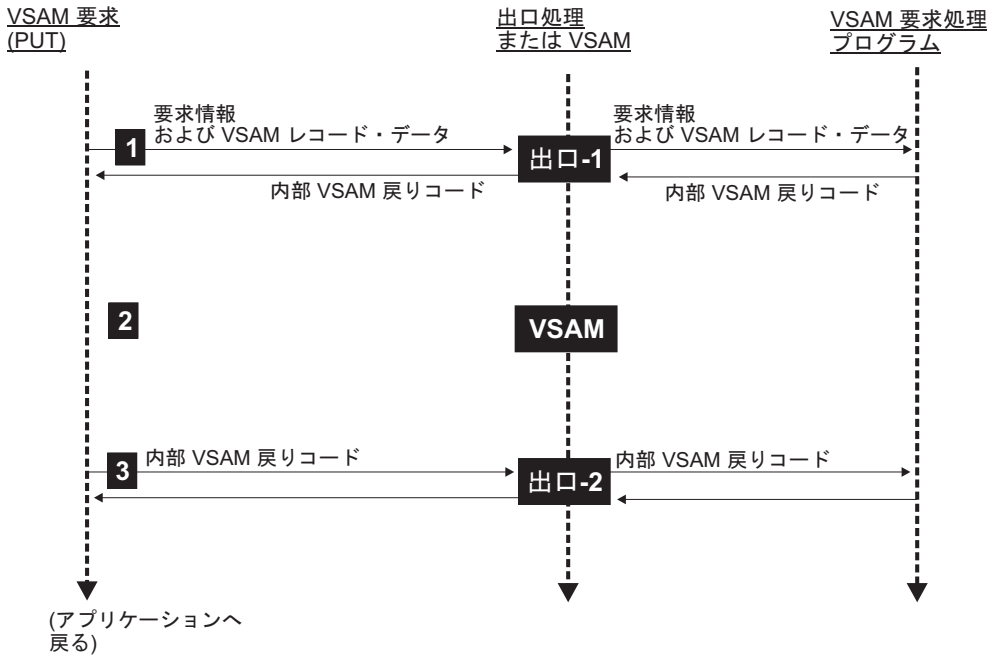


図 *15. VSAM PUT* 要求の制御のフロー

1 VSAM は、要求の処理を開始する前 に **Exit-1** を呼び出します。

2 VSAM はいずれの要求も処理しません。

3 VSAM は、要求の処理を終了した後 で **Exit-2** を呼び出します。

図 16 に、VSAM `GET` の実行時の制御のフローを示します。

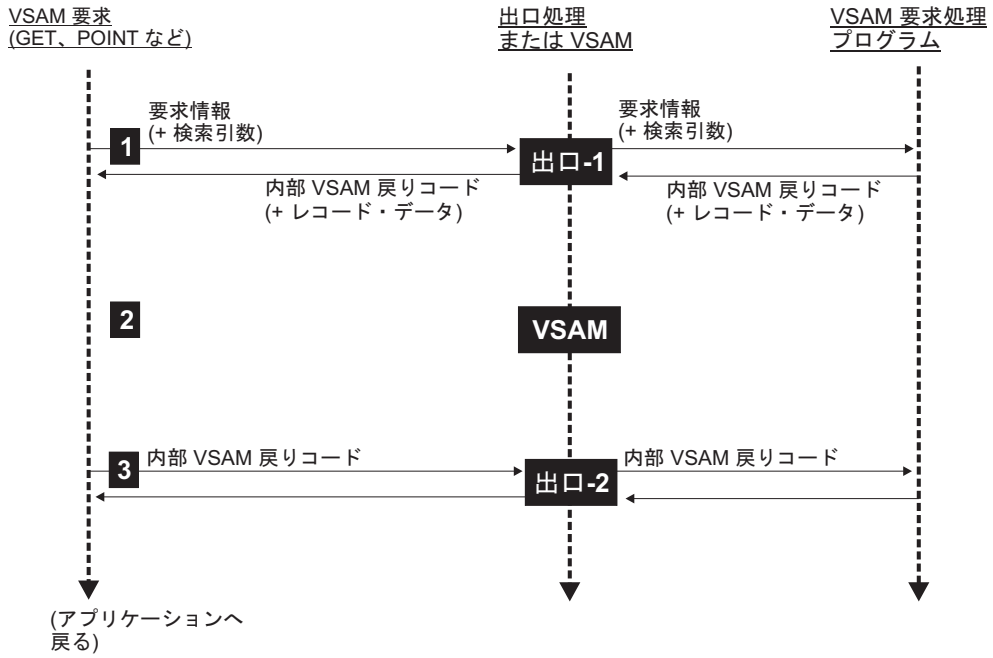


図 *16. VSAM GET* 要求の制御のフロー

1 VSAM は、要求の処理を開始する前 に **Exit-1** を呼び出します。

2 VSAM はいずれの要求も処理しません。

3 VSAM は、要求の処理を終了した後 で **Exit-2** を呼び出します。

モード **2.** 既存の **VSAM** データの同期化**:** このモード (OWNER=VSAM) を使用すると、VSAM データを処理するプログラムは VSAM アクセスおよびリダイレクトされたアクセスを行います。 各 VSAM 要求ごとに 2 つの要求が VSAM リダイレクター・ クライアント (**iesredir.phase**) へ出されます。

図 17 に、VSAM PUT の実行時の制御のフローを示します。

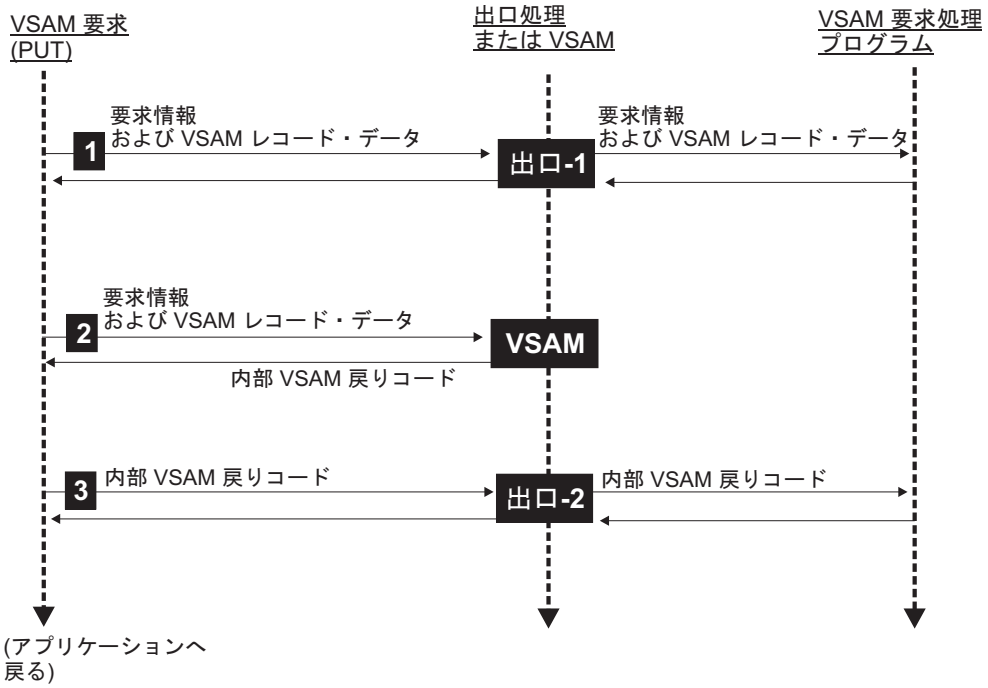


図 *17. VSAM PUT* 要求の制御のフロー

1 VSAM は、要求の処理を開始する前 に **Exit-1** を呼び出します。

2 VSAM は、要求を処理します。

3 VSAM は、要求の処理を終了した後 で **Exit-2** を呼び出します。

64 ページの図 18 に、VSAM GET の実行時の制御のフローを示します。

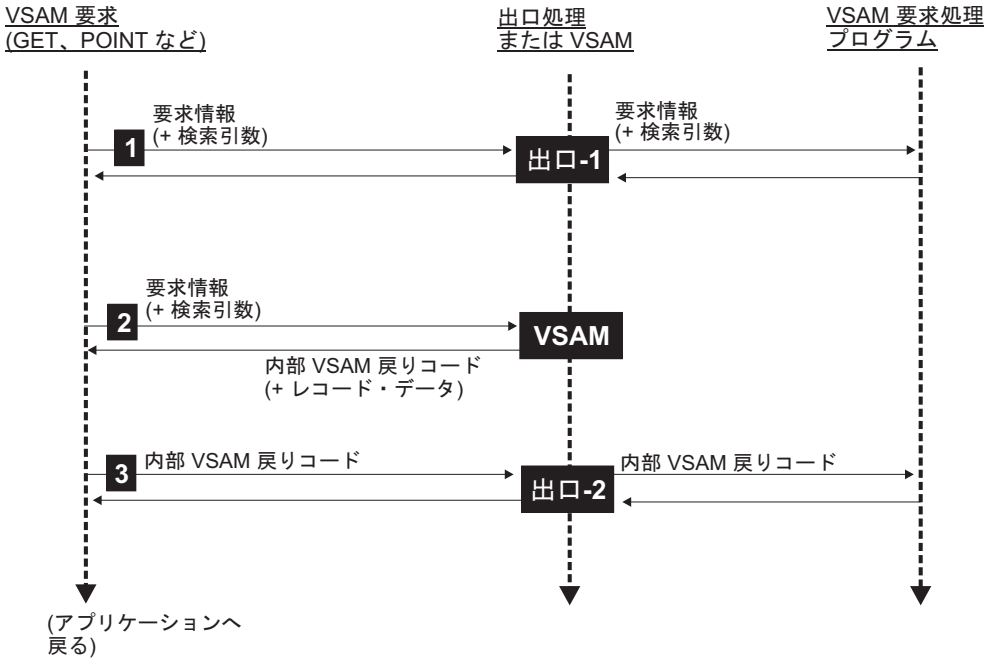


図 *18. VSAM GET* 要求の制御のフロー

1 VSAM は、要求の処理を開始する前 に **Exit-1** を呼び出します。

2 VSAM は、要求を処理します。

3 VSAM は、要求の処理を終了した後 で **Exit-2** を呼び出します。

VSAM 取り込み 出口に使用できるリダイレクト・モード

VSAM 取り込み 出口 (IESVSCAP) の場合、使用できるモードが *3* つあります。

- WebSphere MQ
- VSAM デルタ・クラスター
- ローカル処理

WebSphere MQ リダイレクト・モードの使用: *WebSphere MQ* を使用している場合、*2* つのモードを使用できます。

- *MQ* サーバー・モード を使用している場合は、MQ Series for VSE V2.1 以降を z/VSE ホストにインストールして稼働させておく必要があります。デルタ・メッセージが作成され、z/VSE の MQ 待ち行列に入れられます。 MQ 待ち行列はリモート待ち行列にすることもできます。これは、メッセージが MQ チャネルによって別のシステム上の MQ 待ち行列に非同期に転送されることを意味します。
- *MQ* クライアント・モード を使用している場合は、*WebSphere MQ Client for VSE* を z/VSE にインストールして稼働させておく必要があります。 デルタ・メッセージが作成され、リモート MQ サーバーの MQ 待ち行列に直接入れられます。

VSAM デルタ・クラスター・モードの使用: VSAM デルタ・クラスターを使用している場合、*2* つのモードを使用できます。

- 累積モード では、KSDS タイプのデルタ・クラスターを使用します。 VSAM デルタ・クラスターのキーは、オリジナルのクラスター、または RRN/RBA と

同じです。これは、オフセットが調整されてデルタ・ヘッダー (『デルタ・レコードおよびデルタ・メッセージのレイアウトを理解する』 を参照) を保護することを意味します。 そのため、レコードは 1 回のみ VSAM デルタ・クラスターに保管されます。 これは、最終変更したレコードのみが VSAM デルタ・クラスターに反映されることを意味します。 ただし、これはデータベースを同期化するのに十分な情報です。

- ジャーナリング・モード では KSDS または ESDS タイプの VSAM デルタ・クラスターを使用します。 VSAM デルタ・クラスターに変更が記録されるたびに、キーとして時刻機構を使用して、新しいレコードが追加されます (KSDS の場合)。 そのため、時刻機構は固有なので、レコードの全変更が VSAM デルタ・クラスターに反映されます (変更順序も含む)。

データベースを同期化するには、両方のモードを使用できます。 しかし、環境によっては、オリジナルの順序でレコードの変更を適用する必要がある場合もあります。 この場合、ジャーナリング・モードを使用してください。

ローカル処理モードの使用**:** ローカル処理を実行する場合、デルタ・レコードを処理するために、ユーザー独自のロジックをインプリメントする出口を指定する必要があります。

VSAM 取り込み 出口の 3 つのリダイレクト・モードすべて (WebSphere MQ、VSAM デルタ・クラスター、およびローカル処理) に対して、追加の決定出口 を指定できます。

- 決定出口は、デルタ・レコードまたはデルタ・メッセージが書き込まれる時にはいつでも呼び出されます。
- 決定出口は、デルタ・レコードまたはデルタ・メッセージを書き込むか、または無視するかを決定します。 この決定は、レコードの内容に基づいて行われることがあります (例えば、そのレコードの一部のフィールドの変更が特に意味がない場合など)。
- 決定出口は、古いレコードの内容と新しいレコードの内容を取得します (UPDATE の場合のみ)。
- ローカル処理モードの場合、決定出口は、ロジックをインプリメントしてデルタ・レコードの処理も行います。

決定出口のコーディング例については、ICCF ライブラリー 59 のスケルトン SKDECEXT を参照してください。

デルタ・レコードおよびデルタ・メッセージのレイアウトを理解する

VSAM 取り込み 出口は、それぞれが長さ 38 か 42 バイトのデルタ・ヘッダー で始まるデルタ・レコードを作成します。

```
TODClock        8 bytes     TOD Clock value of update (TOD = Time of day)
JobName         8 bytes     Job Name of Program
Phase           8 bytes     Phase name of program
Origin          8 bytes     Origin value, e.g. Label name
PartID          2 bytes     Partition ID
OpCode          1 byte      'I'=insert,'U'=update,'D'=delete
Flags           1 byte      '01'X= RRD/RBA follows
RecordLen       2 byte      Length of Record (exl. Header and RBA/RRN)
```

ESDS、RRDS、または VRDS データ・セットの場合は、4 バイトの RRN/RBA が取り込みヘッダーに続きます。

```
RBA/RRN         4 bytes     RBA or RelRecNo of record
```

デルタ・レコードの残りは、オリジナル・レコード のコピーで埋められます。 デルタ・レコードの実際の長さは、オリジナル・レコードの長さに 38 か 42 バイト加えた長さになります。

VSAM デルタ・クラスターの使用

VSAM 取り込み 出口を、VSAM デルタ・クラスターと一緒に使用している場合は、VSAM デルタ・クラスターを以下のように定義する必要があります。

- 累積モードの場合 -

```
Original Cluster is a KSDS:
  Cluster Type = KSDS
  MaxRecLen    = Original MaxRecLen + 38 or larger
  KeyPos       = Original KeyPos + 38
  KeyLen       = Original Key Len
Original Cluster is a ESDS:
  Cluster Type = KSDS
  MaxRecLen    = Original MaxRecLen + 42 or larger
  KeyPos       = 38
  KeyLen       = 4
Original Cluster is a RRDS/VRDS:
  Cluster Type = KSDS
  MaxRecLen    = Original MaxRecLen + 42 or larger
  KeyPos       = 38
  KeyLen       = 4
```

- ジャーナリング・モードの場合 -

```
Original Cluster is a KSDS:
  Cluster Type = KSDS or ESDS
  MaxRecLen    = Original MaxRecLen + 38 or larger
  KeyPos       = 0 (applies only if delta cluster is KSDS)
  KeyLen       = 8 (applies only if delta cluster is KSDS)
Original Cluster is a ESDS:
  Cluster Type = KSDS or ESDS
  MaxRecLen    = Original MaxRecLen + 42 or larger
  KeyPos       = 0 (applies only if delta cluster is KSDS)
  KeyLen       = 8 (applies only if delta cluster is KSDS)
Original Cluster is a RRDS/VRDS:
  Cluster Type = KSDS or ESDS
  MaxRecLen    = Original MaxRecLen + 42 or larger
  KeyPos       = 0 (applies only if delta cluster is KSDS)
  KeyLen       = 8 (applies only if delta cluster is KSDS)
```

VSAM デルタ・クラスターを REUSABLE として定義すると便利なことがあります。 これにより、変更が転送されてデータベースにロードされた後、VSAM デルタ・クラスターのクリアが容易になります。

ジャーナリング・モードの場合は、1 つの VSAM デルタ・クラスター で、いくつもの VSAM ファイルの変更を取り込むことができます。このためには、以下を行います。

- VSAM デルタ・クラスターの最大レコード長が、取り込んだクラスターの最大レコード (ヘッダーの長さを加えて) を保持するのに十分な大きさであることを確認してください。
- 後でデルタ・レコードを別の VSAM クラスターと区別するために、構成で固有の ORIGIN 名を指定しておくのも有用です。

- いくつかの区画から同時アクセスを許可するには、通常 VSAM デルタ・クラスターを共用オプション 4 (すなわち SHR(4,3))で定義する必要があります。

VSAM デルタ・クラスターはオプション SPEED を使用して定義できません。 代わりに RECOVERY を使用してください。 VSAM 取り込み 出口によって、VSAM デルタ・クラスターが SPEED オプションを使用していることが検出されると、OPEN 要求が拒否され、該当するメッセージが戻されます。

ジャーナリング・モードでは、VSAM デルタ・クラスターは KSDS または ESDS のいずれかのクラスターにすることができます。 KSDS VSAM デルタ・クラスターでは、デルタ・レコードが処理された時に削除できるようにします。 さらに、KSDS を XXL (特大) として定義して、約 4 GB の制限を回避できます。

WebSphere MQ の使用

モード MQServer または MQClient を使用する場合は、MQ サーバー名を指定する必要があります。以下のいずれかを使用して MQ サーバー名を指定できます。

- SETPARM ステートメント -

  ```
  // SETPARM MQBISRV=servername  (default is MQBISERV)
  ```

- 72 ページの『MODE=MQSERVER および MODE=MQCLIENT の場合のパラメーター』 で説明される構成項目 (MQISRV パラメーター)。

取り込み出口が WebSphere MQ で使用されている場合、VSAM デルタ・クラスターの最大レコード長を保持するために (前に説明したように) MQ 待ち行列を定義する必要があります。

WebSphere MQ を使用するためのガイドライン

- モード MQServer を使用する場合は、MQ サーバーの名前を指定する必要があります (デフォルトは MQBISERV です)。
- モード MQClient を使用する場合は、MQ クライアント・ブリッジの名前を指定する必要があります (デフォルトは MQBISERV です)。
- 同じシステム上に MQ サーバーおよび MQ クライアント・ブリッジの両方が稼働している場合、サーバーとブリッジに対して異なる名前を使用する必要があります。
- WebSphere MQ および/または WebSphere MQ クライアント・ライブラリーを、関係する区画すべての LIBDEF に必ず追加してください。

以下の前提条件を満たす必要もあります。

- MQServer モードを使用するには、MQSeries® for VSE バージョン 2.1.2 以降をインストールして構成する必要があります。MQClient モードを使用するには、WebSphere MQ Client パッケージがインストールされている必要があります。ファイル **mqc5.zip** は、無償で次のアドレスからダウンロードできます。

  ```
  http://www.ibm.com/systems/z/os/zvse/downloads/
  ```

- WebSphere MQ を実行するには、別々の CICS 領域を使用する必要があります。

注: WebSphere MQ を実行するのにも使用している CICS 区画で実行するプログラムによって行われた変更を取り込んだ場合、デッドロックを作成します。これは、両方の モード (WebSphere MQ と MQClient) に対して TRUE です。

- MQ クライアントを使用する場合、MQ クライアント・ブリッジを CICS (MQCI トランザクション) で開始しておく必要があります。 MQ クライアント・ブリッジについて詳しくは、MQ クライアント・パッケージで提供される README ファイルを参照してください。
- MQ クライアント・プログラムとトランザクションを CICS に定義する必要があります。 MQ クライアント・パッケージに含まれるサンプル・メンバー MQCICSD.Z を参照してください。

ステップ 3 (オプショナル): VSAM データを転送する

VSAM リダイレクター・コネクターで既存の VSAM データ・セットを使用するために、必要なファイル・システムに VSAM データをマイグレーションできます (例えば、Db2 フォーマット)。VSAM データを Db2 処理に適したリレーショナル構造にマップする方法について詳しくは、

- 117 ページの『第 12 章 リレーショナル構造への VSE/VSAM データのマッピング』 を参照してください。
- グラフィカル・ユーザー・インターフェース (Mapper Config GUI) を使用してください。

データを z/VSE ホストからターゲット Java プラットフォームのファイル・システムへ転送するには、以下のいずれかを使用できます。

- IDCAMS REPRO ユーティリティー。
  1. リダイレクト/転送プロセス用のターゲットとして VSAM クラスター (ソース・クラスターと同じプロパティーを持つ) を定義します。
  2. VSAM クラスターをリダイレクトする (OWNER=REDIRECTOR) ように構成フェーズを変更します。 詳細については、『ステップ 4: 構成フェーズを作成する』を参照してください。
  3. VSAM リダイレクター・サーバーと、構成フェーズに定義した要求処理プログラムの両方がターゲット Java プラットフォームで実行されていることを確認します。
  4. IDCAMS REPRO ユーティリティーを使用して、リダイレクトされた VSAM クラスターにデータをコピーします。 このアクションが完了すると、データはターゲット Java プラットフォームのファイル・システムに保管されます。
- リダイレクター・ローダー・ユーティリティー。
  1. VSAM リダイレクター・サーバーで提供されるオンライン・ドキュメンテーションで説明されているリダイレクター・ローダーの構成ファイルを設定します (30 ページの『オンライン・ドキュメンテーション・オプションの使用』 を参照)。
  2. リダイレクター・ローダーを実行します。

ステップ 4: 構成フェーズを作成する

VSE ライブラリー 59 には、構成フェーズの作成に使用できるサンプル・スケルトン SKRDCFG が含まれています。 72 ページの『構成フェーズを作成するジョブ

の例』 にサンプル・ジョブ・スケルトン SKRDCFG を示します。 このジョブで設定されるパラメーターは以下のとおりです。

必須パラメーター

CATALOG=

リダイレクトされるファイルの VSAM カタログ名。 パラメーター定義内でワイルドカード * を使用できます。その場合、CLUSTER=* も設定する必要があります。 ただし、すべての カタログ (およびそれらのカタログ内のクラスター) がリダイレクトされるため、CATALOG=* を使用する場合には注意が必要です。

注:

1. マスター・カタログがリダイレクトされると、**z/VSE** システムをスタートアップできなくなる場合があります。
2. ワイルドカードを含む項目は、ほかに一致する項目が検出できなかった場合にのみ使用されます。

CLUSTER=

VSAM クラスター名。 パラメーター定義内でワイルドカード * を使用できます。ただし、指定されたカタログに属するすべての クラスターがリダイレクトされるため、CLUSTER=* を使用する場合には注意が必要です。

使用する出口フェーズの名前。

- IBM 提供の出口 (IESREDIR または IESVSCAP) を指定する場合、下記にリストされたパラメーターが適用されます。
- EXIT に別の値 (例えば、ベンダー提供の出口) を指定すると、それ以上パラメーターが適用されません。

オプション・パラメーター

以下のオプション・パラメーターを使用すると、リダイレクトされるクラスターを指定する際に追加フィルターを指定できます。

CATDD=

カタログのラベル名。 デフォルトは、CATDD='*' です。 CATDD に値 (デフォルト以外) を入力すると、ファイルをリダイレクトするかどうかを調べるために CATALOG と CATDD の両方が使用されます。

CLUDD=

クラスターのラベル名。 デフォルトは、CLUDD='*' です。 CLUDD に値 (デフォルト以外) を指定すると、ファイルをリダイレクトするかどうかを調べるために CLUSTER と CLUDD の両方が使用されます。

MSG=

以下のいずれかの値を指定できます。

YES (デフォルト) リダイレクトされた VSAM クラスターがオープンされると、コンソールにメッセージ IESC2009I が表示されます。

NO リダイレクトされた VSAM クラスターがオープンされたときに、メッセージ IESC2009I は表示されません。

PART=

リダイレクト可能な 区画の区画 ID (例えば、F4)。 デフォルト値は、PART='*' です。

NOTPART=
リダイレクトできない 区画の区画 ID (例えば、F4)。 デフォルトは、すべての 区画をリダイレクトに使用できます。

IESREDIR 出口使用時のパラメーター

OWNER=
以下のうちの 1 つの値を使用できます。

REDIRECTOR
すべての要求が VSAM リダイレクター・ クライアントにリダイレクトされ (**IESREDIR.PHASE**)、そこから Java プラットフォームで実行中の VSAM リダイレクター・サーバーに接続します。 そこで、VSAM リダイレクター・サーバーは要求を実行します。

VSAM
二重処理 (VSAM 処理と VSAM リダイレクター・ クライアントへの要求のリダイレクトの両方) が行われます。

PROTOCOL=31
このパラメーターは、z/VSE 4.1 またはそれ以降でのみ使用できます。 このパラメーターで、VSAM リダイレクター・ クライアントによってプロトコルの古いバージョンにスイッチバックするように命令します。それによって、リモート側の VSAM リダイレクター・サーバーの古いバージョンで z/VSE 4.1 ホストを実行することができます。 このオプションは、VSE システムにアップグレードする際の、円滑なマイグレーションを提供することを意図しています。

IP= 必須パラメーターです。 VSAM リダイレクター・サーバーがインストールされている、接続先のサーバーの IP アドレス。

PORT=
(オプション)。 VSAM リダイレクター・サーバーがインストールされている、接続先のサーバーのポート番号。 VSAM リダイレクター・サーバーで使用されるデフォルトのポート番号は、 Internet Assigned Numbers Authority (IANA) によって割り当てられた **2387** です。

HANDLER=
必須パラメーターです。 開始される Java クラスの名前。この構成項目に使用されるリダイレクター・ハンドラーを示します。

OPTIONS=
必須パラメーターです。 リダイレクター・ハンドラーにオプション文字列として転送されるデータを含む文字列。 これには独自の設定を挿入できます。 68 ページの『ステップ 4: 構成フェーズを作成する』 に示す例では、この文字列に IBM 提供のリダイレクター・ハンドラー *DB2Handler*、*DBHandler*、および *CSVFileHandler* に必要な情報 (DB/2 システム、ユーザー名、パスワードなど) が含まれています。 このパラメーターにブランク値 (「 」) を指定すると、リダイレクター・ハンドラーはオプション文字列としてブランクを受け取ります。

PROTOCOL=
(オプション)。 デフォルトでは、z/VSE 4.1 バージョンのプロトコルが使用されます。 z/VSE 3.1 バージョンのプロトコルを使用するには、パラメーターを **31** にする必要があります。

OWNER=VSAM の場合のパラメーター

IGNOREERROR=

オプション・パラメーター。デフォルトは IGNOREERROR=NO です。 IGNOREERROR=YES を指定した場合、VSAM リダイレクター・サーバーにアクセスできなくても、VSAM OPEN 要求はエラーを戻しません。

PUTREQONLY=

オプション・パラメーター。デフォルトは PUTREQONLY=YES です。 PUTREQONLY=YES に設定すると、INSERT、UPDATE、および DELETE 要求のみ、VSAM リダイレクター・サーバーにリダイレクトされます。 POINT、GET などの要求を除外してリダイレクター・ハンドラーに関する統計を取りたい場合に、このパラメーターが役立つ場合があります。

IESVSCAP 出口使用時のパラメーター

MODE=

以下のうちの 1 つの値を使用できます。

- **JOURNALING** または **CUMULATIVE** (VSAM デルタ・クラスターで使用する場合)。
- **MQSERIES** または **MQCLIENT** (WebSphere MQ で使用する場合)。
- **LOCAL** (ローカル処理の場合)。

ORIGIN=

8 文字長までの名前を指定します。 これは自由に選択できます。 デルタ・ヘッダーの一部となるので、デルタ・クラスターまたは MQ メッセージ内のそれぞれのデルタ・レコードに表示されます。 これは、後でデルタ・レコードまたはデルタ・メッセージを別のクラスター (それぞれの元) と区別するのに使用できます。 ORIGIN が除外されると、クラスターの現行 DLBL 名が使用されます。

DECEXIT=

決定出口をインプリメントするフェーズ名。 MODE=LOCAL の場合を除いて、オプションです。 プログラミング例については、ICCF ライブラリー 59 のスケルトン SKDECEXT を参照してください。

IGNOREERROR= NO | YES

(オプション)。 YES を設定した場合、エラーが発生しても、すべてのエラーが無視され、処理が続行されます。 これは、デルタ・クラスターまたは待ち行列にすべての変更が含まれるということではないので、データの不整合を招くことがあります。

MODE=JOURNALING または MODE=CUMULATIVE の場合のパラメーター

DELTADD=

デルタ・レコードを保管するために使用されるデルタ・クラスターの DLBL 名を指定します。 このパラメーターは、ジャーナリング・モードか累積モードを使用する場合にのみ 許可されます。

DELTACAT=

(オプション)。 デルタ・クラスターがあるカタログの 44 バイトのファイル

ID を指定します。 このパラメーターは、DELTADD が指定されない 場合のみ 許可されます。 これは、DELTACLU と併せて指定する必要があります。

DELTACLU=
(オプション)。 デルタ・レコードを保管するために使用されるデルタ・クラスターの 44 バイトのファイル ID を指定します。 このパラメーターは、DELTADD が指定されない場合のみ 許可されます。 これは、DELTACAT と併せて指定する必要があります。

SHARE=NONE | ENDREQ | TCLOSE
(オプション)。 それぞれのデルタ・レコードの挿入後、ENDREQ か TCLOSE 要求を発行するかどうかを指定します。

DSNSTR=
オプション・パラメーター (デフォルトは 255 です)。 データ・セット名共用で使用される 0 から 255 までのストリング数を指定します。 0 は、データ・セット名共用が使用されないことを示します。

DELTATYPE=KSDS | ESDS
(オプション)。 ESDS は、MODE=JOURNALING の場合のみ許可されます。

MODE=MQSERVER および MODE=MQCLIENT の場合のパラメーター

QMGR=
WebSphere MQ 待ち行列マネージャーの名前 (48 文字まで)。

QNAME=
WebSphere MQ 待ち行列の名前 (48 文字まで)。

MQISRV=
z/VSE MQ サービスの XPCC の名前。 詳細については、『MODE=MQCLIENT の場合のパラメーター』を参照してください。

MODE=MQCLIENT の場合のパラメーター

MQSERVER=
MQServer の IP またはホスト名 (100 文字まで)。

MQCHANNEL=
MQServer で使用されるチャネルの名前 (20 文字まで)。

構成フェーズを作成するジョブの例

以下の例は、サンプル・ジョブ・スケルトン SKRDCFG を示しています。

図 *19. VSAM* リダイレクター・コネクター用の構成フェーズを提供するジョブ

```
* $$ JOB JNM=RDCONFIG,CLASS=A,DISP=D
// JOB RDCONFIG GENERATE REDIRECTOR CONFIG PHASE
* ********************************************************************
* STEP 1: ASSEMBLE AND LINK THE CONFIG TABLE                         *
* ********************************************************************
// LIBDEF *,CATALOG=PRD2.CONFIG
```

```
// LIBDEF *,SEARCH=PRD1.BASE
// OPTION ERRS,SXREF,SYM,NODECK,CATAL,LISTX
   PHASE IESRDCFG,*,SVA
// EXEC ASMA90,SIZE=(ASMA90,64K),PARM='EXIT(LIBEXIT(EDECKXIT)),SIZE(MAXC
              -200K,ABOVE)'
IESRDCFG CSECT
IESRDCFG AMODE ANY
IESRDCFG RMODE ANY
*
         IESRDENT CATALOG='VSESP.USER.CATALOG',                        X
               CLUSTER='MY.TEST.CLUSTER1',                             X
               EXIT='IESREDIR',                                        X
               OWNER=REDIRECTOR,                                       X
               IP='10.0.0.1',                                          X
               HANDLER='com.ibm.vse.db2handler.DB2Handler',            X
               OPTIONS='db2url=jdbc:db2:redir;db2user=hugo;            X
                        db2password=hugospw;db2table=mydata'
*
          IESRDENT CATALOG='VSESP.USER.CATALOG',                       X
               CLUSTER='MY.TEST.CLUSTER2',                             X
               EXIT='IESVSCAP',                                        X
               MODE=JOURNALING,                                        X
               DELTADD='DELTAFI',                                      X
               DELTATYPE=KSDS,                                         X
               SHARE=ENDREQ,                                           X
               ORIGIN='TEST2'
*
         IESRDENT CATALOG='VSESP.USER.CATALOG',                        X
               CLUSTER='MY.TEST.CLUSTER3',                             X
               EXIT='IESVSCAP',                                        X
               MODE=MQSERVER,                                          X
               QMGR='VSE.QUEUE.MANAGER',                               X
               QNAME='CAPTURE.INPUT.QUEUE',                            X
               ORIGIN='TEST3'

         IESRDENT CATALOG='VSESP.USER.CATALOG',                        X
               CLUSTER='MY.TEST.CLUSTER2',                             X
               EXIT='VENDOREX'
*
         END
/*
// IF $MRC GT 4 THEN
// GOTO NOLINK
// EXEC LNKEDT,PARM='MSHP'
/. NOLINK
/*
* $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
* STEP 2: LOAD THE IESRDCFG.PHASE INTO THE SVA (OPTIONAL)            *
* $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
* LIBDEF *,SEARCH=PRD2.CONFIG
* SET SDL
* IESRDCFG,SVA
* /*
* $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
* STEP 3: COPY IESVEX01.PHASE INTO PRD2.CONFIG AS IKQVEX01           *
* $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
// EXEC LIBR,PARM='MSHP'
  CONNECT S=PRD1.BASE:PRD2.CONFIG
  COPY IESVEX01.PHASE:IKQVEX01.PHASE REPLACE=YES
/*
* $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
* STEP 4: LOAD THE IKQVEX01.PHASE INTO THE SVA (OPTIONAL)            *
* $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
* LIBDEF *,SEARCH=PRD2.CONFIG
* SET SDL
* IKQVEX01,SVA
* /*
* $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
* STEP 5: LOAD THE IESRDANC.PHASE INTO THE SVA (OPTIONAL)             *
*         THIS SHOULD BE DONE ONLY ONCE  !!                           *
* $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

```
* // LIBDEF *,SEARCH=PRD2.CONFIG
* SET SDL
* IESRDANC,SVA
* /*
* ********************************************************************
* STEP 6: REGISTER THE CURRENT CONFIGURATION PHASE                    *
* ********************************************************************
* // LIBDEF *,SEARCH=PRD1.BASE
* // EXEC IESRDLDA
* /*
/&
* $$ EOJ
```

VSAM リダイレクター・サーバーのインストール

このトピックでは、マイグレーションしたデータ・セットをインストールしようとする Java プラットフォームごとに実行しなければならないアクティビティーについて説明します。

マイグレーションしたデータ・セットをインストールしようとする Java プラットフォームごとに実行しなければならない主なアクティビティーには、以下のものがあります。

- 『ステップ 1: インストール・ファイルをダウンロードしてインストールを実行する』
- 76 ページの『ステップ 2: プロパティー・ファイルを構成する』
- 76 ページの『ステップ 3: VSAM リダイレクター・ハンドラー をインプリメントする』

ステップ 1: インストール・ファイルをダウンロードしてインストールを実行する

VSAM リダイレクター・サーバーは Java 対応プラットフォームにインストールします。

注: 始める前に、VSAM リダイレクター・サーバーをインストールしようとする開発プラットフォームに Java Development Kit (JDK) 1.5 以降がすでにインストールされていなければなりません。 JDK 1.5 以降をインストールしていない場合のインストール方法について詳しくは、 24 ページの『Java のインストールと構成』を参照してください。

ステップ 1.1: VSAM リダイレクター・サーバーのコピーを取得する

VSAM リダイレクター・サーバー のコピーを取得するには、以下のいずれから取得するかを決定する必要があります。

- インターネットから
- Extended Base Tape から VSE コネクター・ワークステーション・コード・コンポーネントをインストールすることによって

インターネットから VSAM リダイレクター・サーバー を取得するには、以下の作業を行う必要があります。

1. Web ブラウザーを起動して次の URL に進みます (英語のみ)。

   ```
   http://www.ibm.com/systems/z/os/zvse/downloads/
   ```

2. VSAM リダイレクター・サーバー のセクションから、VSAM リダイレクター・サーバー をインストールするディレクトリーにファイル **redir*nnn*.zip** をダウンロードします。 注: *nnn* は現行の VSE バージョンを指します (例えば、**redir430.zip**)。

VSAM リダイレクター・サーバー を *VSE* コネクター・ワークステーション・コード・コンポーネントをインストールすることによって 取得するには、以下を行ってください。

1. Extended Base Tape から VSE コネクター・ワークステーション・コード・コンポーネントをインストールします。このコンポーネントのインストール後、VSAM リダイレクター・サーバーの VSE コネクター・ワークステーション・コード **iesvsmrd.w** が、z/VSE サブライブラリー PRD2.PROD に保管されます。
2. TCP/IP for z/VSE の FTP (ファイル転送プログラム) ユーティリティーを使用して、VSAM リダイレクター・サーバー をインストールしたいディレクトリーに **iesvsmrd.w** をダウンロードします。

注:

1. **iesvsmrd.w** はバイナリーでダウンロードする必要があります。
2. UNIX モードがオフ になっていることを確認してください。 オフにしないと、バイナリー を指定しても、**iesvsmrd.w** は ASCII モードでダウンロードされます。 *Unix* モード は VSE FTP デーモンのパラメーターの 1 つです。 FTP クライアントの中には、UNIX モードを強制的に オンにするものがあります。 以下の例に、バッチ FTP クライアントを使用して正常に **iesvsmrd.w** を転送する方法を示します。 UNIX モードが設定される場所は太字で示しています。

```
c:¥temp>ftp 9.164.155.2
Connected to 9.164.155.2.
220-TCP/IP for VSE -- Version 01.05.F  -- FTP Daemon
    Copyright (c) 1995,2006 Connectivity Systems Incorporated
220 Service ready for new user.
User (9.164.155.2:(none)): sysa
331 User name okay, need password.
Password:
230 User logged in, proceed.
ftp> cd prd2
250 Requested file action okay, completed.
ftp> cd prod
250 Requested file action okay, completed.
ftp> binary
200 Command okay.
ftp> get iesvsmrd.w
200 Command okay.
150-File: PRD2.PROD.IESVSMRD.W
    Type: Binary Recfm: FB Lrecl:    80 Blksize:      80
    CC=ON  UNIX=OFF RECLF=OFF TRCC=OFF CRLF=ON  NAT=NO
150 File status okay; about to open data connection
226-Bytes sent:       4,756,400
    Records sent:        59,455
    Transfer Seconds:        16.52 (    290K/Sec)
    File I/O Seconds:         3.94 (  1,548K/Sec)
226 Closing data connection.
4756400 bytes received in 17,12 seconds (277,91 Kbytes/sec)
ftp> bye
221 Service closing control connection.
c:¥temp>ren iesvsmrd.w redir.zip
```

ステップ 1.2: VSAM リダイレクター・サーバーのインストールを実行する

VSAM リダイレクター・サーバーのインストールを実行するには、以下の作業を行う必要があります。

1. ファイル **redir.zip** を unzip します。これには以下のファイルが入っています。
   - setup.jar (VSAM リダイレクター・サーバー・コードを含む)
   - setup.bat (Windows 用インストール・バッチ・ファイル)
   - setup.cmd (OS/2 用インストール・バッチ・ファイル)
   - setup.sh (Linux/Unix 用インストール・スクリプト)
2. (ファイルをダブルクリックして) オペレーティング・システム・プラットフォームに適用可能なバッチ・ファイルを開始します。
3. インストール・プロセスが開始され、さまざまなインストール・メニューによるガイドが表示されます。
4. HTML ベースのドキュメンテーションにアクセスするには、ここで Web ブラウザーを使用してファイル **... (**リダイレクターのルート・ディレクトリー**) /doc/index.html** をオープンすることができます。

ステップ 2: プロパティー・ファイルを構成する

VSAM リダイレクター・サーバーのプロパティー・ファイルは **VSAMRedirectorServer.properties** という名前で、テキスト・エディターを使用して編集できるテキスト・ファイルです。

コメント行には、最初の桁に # 文字が付いています。

VSAMRedirectorServer.properties に定義する設定は次のとおりです。

messages= on | off
: **messages= on** を定義すると、すべてのメッセージが印刷されます。
messages= off を定義すると、メッセージは印刷されません (「静止モード」がアクティブになります)。

listenport = *TCP/IP* ポート番号
: VSAM リダイレクター・サーバーが要求の listen に使用するポート番号。

maxconnections = 数値
: VSAM リダイレクター・クライアントから接続できる最大数。

codepagetranslator = com.ibm.vse.server.DefaultTranslator
: 以下のストリングの変換に使用されるコード・ページ変換プログラム・クラス。
  - EBCDIC から ASCII
  - ASCII から EBCDIC

  IBM 提供のデフォルトは上記のとおりです。

ステップ 3: VSAM リダイレクター・ハンドラー をインプリメントする

VSAM リダイレクター・ハンドラー (単に、リダイレクター・ハンドラー とも言う) は、VSAM リダイレクター・サーバーによって使用されます。

リダイレクター・ハンドラー は、 55 ページの図 13 に示すとおりのもので、Java でプログラミングされています。

このトピックの以下の見出しの下に、リダイレクター・ハンドラーをインプリメントする方法を説明します。
- 『VSAM ロジックおよびパラメーターのコーディング』
- 『VSAM リダイレクター・ハンドラーの呼び出し』
- 78 ページの『エラー・レポート作成』
- 78 ページの『データ・タイプの変換』
- 79 ページの『リダイレクター・ハンドラーへのマップの動的な取得』
- 82 ページの『Db2 関連要求処理プログラムの IBM 提供の例』

VSAM ロジックおよびパラメーターのコーディング

VSAM データを他のデータ・フォーマット (SQL データベース、フラット・ファイルなど) にリダイレクトして、このデータを既存のアプリケーションに認識されないようにするには、リダイレクター・ハンドラーでのすべての VSAM の振る舞いをシミュレートする必要があります。 そのためには、リダイレクター・ハンドラーに位置決めおよびエラー・レポートの作成ロジックを組み込む必要があります。

VSAM リダイレクター・コネクターで使用する独自のリダイレクター・ハンドラーをプログラムして、組み込むことができます。 以下の項目を処理するために、ケース・メソッド をインプリメントすることで、リダイレクター・ハンドラーが提供するインターフェースもコーディングする必要があります。
- OPEN 要求
- CLOSE 要求
- 記録要求 (要求は以下のとおりです)。
  - GET
  - PUT (UPDATE+INSERT)
  - ENDRQ
  - POINT

(IBM 提供の *DB2Handler* と同じ方法で) VSAM ファイルをコピーするリダイレクター・ハンドラーを作成する場合、そのリダイレクター・ハンドラーで、以下の処理を行う必要があります。

1. VSAM ロジックをコピーする。
2. VSAM 自身と同じ方法で応答する。

VSAM ロジックおよびパラメーターのコーディング方法について詳しくは、以下を参照してください。
- *IBM z/VSE VSE* 中央機能 *VSE/VSAM* ユーザーズ・ガイドおよびアプリケーション・プログラミング, SC34-2704
- *IBM z/VSE VSE* 中央機能 *VSE/VSAM* コマンド, SC43-2946
- *VSAMRequestInfo.java* および *VSAMFileInfo.java* 用の Javadoc。

VSAM リダイレクター・ハンドラーの呼び出し

VSAM リダイレクター・サーバーは Java でインプリメントされ、ソース・コードは配布されません。 VSAM リダイレクター・サーバーは、以下の処理の後で開始できます。

1. プロパティー・ファイルを構成した。

2. リダイレクター・ハンドラー・コードを VSAM リダイレクター・サーバーのインストール先のディレクトリーにコピーした。

VSAM リダイレクター・サーバーが開始されると、以下の処理が行われます。

1. VSAM リダイレクター・サーバーによりプロパティー・ファイルが読み取られます。
2. VSAM リダイレクター・サーバーにより 76 ページの『ステップ 2: プロパティー・ファイルを構成する』で定義した TCP/IP ポート上の VSAM リダイレクター・クライアントが listen されます。
3. VSAM リダイレクター・サーバーが OPEN 要求を受け取ると、以下の処理が行われます。
   a. configuration.phase に定義したリダイレクター・ハンドラーがインスタンス化されます。
   b. リダイレクター・ハンドラーの OPEN メソッドが呼び出されます。
4. リダイレクター・ハンドラーは、*CLOSE* メソッド 呼び出しで定義された方法で終了します。

エラー・レポート作成

リダイレクター・ハンドラーは、例外 を生成してエラーを報告します。 VSAM リダイレクター・サーバーは例外を代行受信して、z/VSE ホスト上で実行中の VSAM リダイレクター・ クライアントへエラー・コードを送信します。 エラー・コードと一緒に送信されるデータはありません。

VSAM リダイレクター・ クライアントへ送信されるエラー・コードは、次のようになります。

- 内部 VSAM 戻りコードである。
- レジスター 15 (R15) に入れて VSAM に戻される。

VSAM リダイレクター・コネクターで使用される新規エラー・コードが作成されました。リダイレクター・ハンドラーが `DUPLICATE RECORD` 警告を VSAM アプリケーションに送信するときに (代替索引アクセス/パス・アクセスで複数のレコードが検出されたとき、およびその他のレコードが後に続くとき)、例外が生成されます。ただし、このときに、レコード・データは送信されません。 この警告メッセージを戻すには、エラー・コード 255 を使用する必要があります (このコードは、サーバー・パーツ内で必要な警告メッセージに変換されます)。

エラー・レポートに関する考慮事項について詳しくは、 58 ページの『VSAM 統合に関する考慮事項』も参照してください。

データ・タイプの変換

z/VSE ホストで実行中の VSAM リダイレクター・ クライアントから転送されるレコード・データは EBCDIC 文字で構成されます。 このため、ストリングとして解釈されたデータはすべて *ASCII* 文字 に変換する必要があります。

z/VSE 4.1 以降、データ・タイプの変換に、以下の IBM 提供のコンバーター を使用できます。

- BINARY - バイナリー・データ、変換なし
- BIT - 単一ビット (バイト)

- DATETIME - さまざまな日時形式のコンバーター
- FIXEDTEXTNUMBER - テキスト形式で保管される固定小数点
- FLOAT - 浮動小数点 (BFP および HFP 形式)
- FLOATTEXTNUMBER - テキスト形式で保管される浮動小数点
- INTEGER - 整数
- PACKED - パック 10 進数
- S2Y - 2000 年問題変換に使用される特殊な形式
- STRING - テキスト文字列
- TOD - 時刻機構 (時刻) コンバーター
- ZONED - ゾーン 10 進数

注:

1. IBM 提供のリダイレクター・ハンドラー Db2Handler (『IBM 提供の VSAM リダイレクター・ハンドラーの使用』 を参照) はこれらのコンバーターを使用します。
2. DBHandler はデータ・タイプの変換にこれらのコンバーターを使用します。
3. コンバーターについて詳しくは、オンライン・ドキュメンテーションを参照してください。

リダイレクター・ハンドラーへのマップの動的な取得

リダイレクター・ハンドラーにマップを動的に 取得する方法は、以下のとおりです。

- **config.phase** にオプション文字列としてマップを定義して、構文解析する。
- z/VSE コネクター・フレームワークを使用してクラスターから VSAM マップを取得する。
- XML ファイルを使用して、構文解析する。(このファイルは *MapTool* を使用して作成できます。例については、*CreateDB2Tables.java* を参照してください。)
- マップを別の場所に保管する。(例えば、データベース表など。例については、*DB2Handler.java* または *DBHandler.java* を参照してください。)

IBM 提供の VSAM リダイレクター・ハンドラーの使用

z/VSE は、以下の VSAM リダイレクター・ハンドラーを提供します。

- **DB2Handler**
- **DBHandler**
- **CSVFileHandler**

表 3 には、これら 3 つのハンドラーの特性の要約が記載されています。

表 *3.* 現在提供されている *VSAM* リダイレクター・ハンドラー

関数	**DB2Handler**	**DBHandler** および **CSVFileHandler (z/VSE 4.1** およびそれ以降**)**
リダイレクトのモード	• owner=VSAM • **owner=REDIRECTOR**	• **owner=VSAM**

表 3. 現在提供されている *VSAM* リダイレクター・ハンドラー (続き)

関数	**DB2Handler**	**DBHandler** および **CSVFileHandler (z/VSE 4.1** およびそれ以降**)**
サポートされる VSAM 要求	**KSDS**、**ESDS** の場合。 **GET**、**POINT**、 INSERT、UPDATE、 DELETE、ENDRQ	**KSDS** 用の **DBHandler**。 INSERT、UPDATE、DELETE、ENDRQ **ESDS**、**VRDS**、**RRDS** 用の **DBHandler**。 INSERT、ENDRQ **KSDS**、**ESDS**、**RRDS**、**VRDS** 用の **CSVFileHandler**。 INSERT、ENDRQ
データのターゲット・タイプ	SQL データベース・システム	SQL データベース・システム、CSV ファイル
データのターゲット	1 つの VSAM ファイルに対して 1 つのデータベース表	1 つの VSAM ファイルに対して複数の表
VSAM キーのマッピング	キーは、タイプ STRING の 1 フィールドにする必要がある	キーに複数のデータ・フィールドが含まれることのサポート
固定長リストのサポート	各リスト項目は、データベース表で 1 つのフィールドにする必要がある	正規化された表をサポートする。各リスト項目はリスト項目表に保管される。
可変長リストのサポート	n/a	サポートあり
レコード・タイプのサポート	n/a	レコードのフィールドの値によって異なるマッピングをサポートする

79 ページの表 **3** への追加情報

- *DB2Handler* リダイレクター・ハンドラーによって、VSAM レコードに属するすべての データ・フィールドを *1* つの SQL データベース表のそれぞれのフィールドにマップできます。 VSAM キー・フィールドは、タイプ STRING の完全なフィールドとしてマップされる必要があります。 このことは、ハンドラーが VSAM GET 要求をサポートするために必須です。ハンドラーは以下を行う必要があるからです。
  - アプリケーションが要求した VSAM キーを見つける。
  - VSAM キーによって前方、後方を検索できる。
- ただし、*DB2Handler* を使用する場合は、VSAM レコードに属するすべてのデータベースを *1* つの SQL データベース表に 1 対 1 でマッピングするために、実用的でないデータベース表設計になってしまいがちだという限界があります。例えば、VSAM レコードにデータ `MYLIST PIC X(20) OCCURS 10` が含まれている場合、データベース表には、`MYLIST1, MYLIST2, ...` といった名前を持つ 10 個のフィールドが含まれます。
- *DBHandler* と *CSVFileHandler* リダイレクター・ハンドラーの使用
  - 上記で説明したマッピング制限は除去されています。 これら 2 つのハンドラーのデータベース表設計によると、例えば `MYLIST PIC X(20) OCCURS 10` のようなリストが、正規化 されます。これは、メイン表のレコードと同じキーを持つ別のデータベース表を作成することによって行われます。このデータベース表には、リストの各 項目に対して *1* つのレコードがあります。データベース表の各レコードに、メイン・データ・レコードのキーが含まれて

いるので、これらのレコードがメイン・レコードにリンクされます。 結果として、各メイン・データ・レコードの異なる数のリスト・レコードを作成する際には、データベース表の変更が不要となります。

- 同じ VSAM ファイル内の別のレコードのタイプを別の データベース表にマップできます。 このマッピングは、VSAM レコードのデータ・フィールドの値に基づいています。
- VSAM を使用するアプリケーションは、データベース表から (VSAM GET を使用して) データを検索できません。 2 つのリダイレクター・ハンドラーは以下を行うために使用できます。(1) VSAM データをデータベースに含まれるデータと同期化、または (2) VSAM ファイルのデータをデータベースに移行。
- これら 2 つのリダイレクター・ハンドラーについて詳しくは、VSAM リダイレクター・サーバーで提供されているオンライン・ドキュメンテーションを参照してください (30 ページの『オンライン・ドキュメンテーション・オプションの使用』 を参照)。

IBM 提供の VSAM リダイレクター・ローダーの使用

IBM 提供の *VSAM* リダイレクター・ローダー (リダイレクター・ローダー とも言う) によって VSAM クラスターからデータをダウンロードできます。転送機構は、データの大量転送のために最適化されました。

すべての IBM 提供のリダイレクター・ローダーは、VSAM リダイレクター・ローダー (リダイレクター・ローダー とも言う) を使用して、ダウンロードしたデータの処理を行います。データを処理 (例えば、データをデータベースに挿入) するためのロジックをコーディングする必要はありません。 代わりに、リダイレクター・ローダーは、以下を行えます。

1. 構成したリダイレクター・ハンドラーを呼び出します。
2. リダイレクター・ハンドラーにデータを処理させます。

通常、リダイレクター・ローダーは、リダイレクター・ハンドラーを INSERT 要求で呼び出します。 これによって、リダイレクター・ローダーと VSAM リダイレクター・サーバーに対して、同じリダイレクター・ハンドラー構成を使用できます。これは、以下ができることを意味します。

1. リダイレクター・ローダーを使用して、最初にデータベースをロードします。
2. リダイレクター・ハンドラーを使用して、データベースを VSAM クラスターで同期化します。

以下が使用可能なリダイレクター・ローダーです。

- **RedirLoader**
- **MQLoader**
- **DeltaLoader**

RedirLoader リダイレクター・ローダーの説明

RedirLoader リダイレクター・ローダーには、リダイレクター・ハンドラーのロード機能があります。 RedirLoader は、

1. VSE コネクター・クライアント を使用して、単純な 2 進数形式の VSAM レコードを読み取ります。

2. VSAM INSERT 要求に見せかけて、これらのレコードをリダイレクター・ハンドラーに送信します。

これにより、リダイレクターが使用するものと同じリダイレクター・ハンドラーを使用しながら、データベースの初期ロードがより高速になります。 z/VSE のリダイレクターを構成する必要はありません。

MQLoader リダイレクター・ローダーの説明

MQLoader リダイレクター・ローダーは VSAM 取り込み 出口 IESVSCAP によって取り込まれた VSAM レコードを挿入するために使用されます。

1. MQLoader は、新規メッセージ (レコード) が MQ 待ち行列で使用可能な時にはいつでも WebSphere MQ トリガー・モニター・プログラムから呼び出されます。
2. VSAM 取り込み 出口 IESVSCAP は、VSAM INSERT、ERASE および UPDATE の各要求の MQ メッセージを作成します。
3. MQ メッセージは WebSphere MQ によって、リモート z/VSE ホスト上で稼働している別の MQ サーバーに転送されます。
   a. 要求が、リモート z/VSE ホストに届くと、MQ は要求を取得してそれを構成したリダイレクター・ハンドラーに送る MQLoader を起動します。
   b. 要求は、処理されないか、エラーを含んでいる場合、エラー待ち行列に保管されます。 これらの要求は後で回復できます。

DeltaLoader リダイレクター・ローダーの説明

DeltaLoader リダイレクター・ローダーは VSAM 取り込み 出口 IESVSCAP によって取り込まれた VSAM レコードを挿入するために使用されます。

1. VSAM INSERT、ERASE または UPDATE の各要求の場合、VSAM 取り込み 出口 IESVSCAP は、別のデルタ VSAM クラスターで VSAM レコードを作成します。
2. DeltaLoader は、VSAM デルタ・クラスターからデルタ・レコードを読み取り、構成したリダイレクター・ハンドラーに送信します。
   a. デルタ・レコードが正常に処理された場合、それらは VSAM デルタ・クラスターから削除されます。
   b. 要求が、処理されないか、エラーを含んでいる場合、VSAM デルタ・クラスターに保持に保管されます。 これらの要求は後で回復できます。

Db2 関連要求処理プログラムの IBM 提供の例

この例 (*DB2Handler*) では、特定のファイルへのすべての VSAM 要求をリモート Db2 データベースにリダイレクトする方法を示します。この例を使用するには、以下のものをリモート・システムにインストールしておく必要があります。

- Db2 データベース
- Java Development Kit

DB2Handler は、Java クラス **com.ibm.vse.db2handler.DB2Handler** で提供され、オプション文字列には以下のものがあります。

- username
- tablename

- password
- systemname

DB2Handler を実行するには、DB/2 JDBC 1.2 ドライバー (**db2java.zip**) を CLASSPATH 変数に組み込む必要があります。

DB2Handler は、ESDS/KSDS および RRDS/VRDS ファイルを処理するさまざまなサブハンドラーを開始します。 各サブハンドラーの動作は、以下のとおりです。

1. フィールド定義を含むデータベース表をオープンします。
2. データベース表を読み取ります。
3. レコード要求用にデータベース表を準備します。
4. それぞれの着信要求を分析して、要求ごとに処理します。
5. 結果を VSAM リダイレクター・ クライアントに戻します。

詳しくは、詳細コメントを含む各サブハンドラーのソース・コードを参照してください。

DB2Handler の使用に関する制限は、以下のとおりです。

- 共用オプション 1 および 2 のみがサポートされます。 同じ Db2 表への並行更新はサポートされません。ただし、リダイレクター・ハンドラーがデータベース・ロックに応答するようにコーディングできます。
- STRING フィールドは、(BASE クラスターおよび AIX 用) KSDS ファイルの KEY フィールドとしてのみサポートされます。
- 「非固有」AIX アクセスはサポートされません。 このタイプのアクセスはまったく行わないでください。
- 0x00 を含むフィールド (NULL フィールド) は、キー列が NOT NULL に定義されているために、(BASE クラスターおよび AIX 用) KEY フィールドとしてサポートされません。

CreateDB2Tables プログラム (**com.ibm.vse.db2handler.create.CreateDB2Tables**) は、DB2 システム上にフィールド定義表を作成します。 *CreateDB2Tables* を使用するには、以下を実行する必要があります。

1. 表の作成に必要なフィールド定義を記述する XML ファイルを作成します。 DTD でテンプレートとして使用できるサンプル XML ファイルが提供されています。 フィールド・タイプには
   STRING、UNSIGNED、SIGNED、PACKED、および BINARY を定義できます。 データベースにおいては、
   - STRING は CHAR です。
   - UNSIGNED、SIGNED、および PACKED は INT です。
   - BINARY は BLOB (バイナリー・ラージ・オブジェクト) にマップされます。
2. プログラム *CreateDB2Tables* を開始して、このプログラムが示す手順に従います。
3. 表 (フィールド定義およびデータ表を含む) が作成され、必要であれば索引が作成されます。

第 9 章 データベース呼び出しレベル・インターフェースのインストール

この情報では、データベース呼び出しレベル・インターフェース (DBCLI) のインストール方法について説明します。これによって、z/VSE アプリケーションが、適切なデータベース・サーバー上でリレーショナル・データベースにアクセスできるようになります。そうなれば、z/VSE 以外のプラットフォームで稼働するデータベース・サーバー (IBM Db2、Oracle、Microsoft SQL Server、MySQL など) を自由に選択できます。

この情報には以下のメイン・トピックがあります。

- 『データベース呼び出しレベル・インターフェース の概要』
- 87 ページの『データベース呼び出しレベル・インターフェースを使用する際の前提条件』
- 87 ページの『DBCLI サーバーのインストール』
- 89 ページの『DBCLI サーバー のアンインストール』
- 89 ページの『DBCLI サーバーのセットアップと構成』
- 91 ページの『接続プーリング・マネージャーの構成と開始/停止』
- 94 ページの『DBCLI サーバー コマンド』

関連トピック:

- 321 ページの『第 22 章 データにアクセスするための データベース呼び出しレベル・インターフェース の使用』

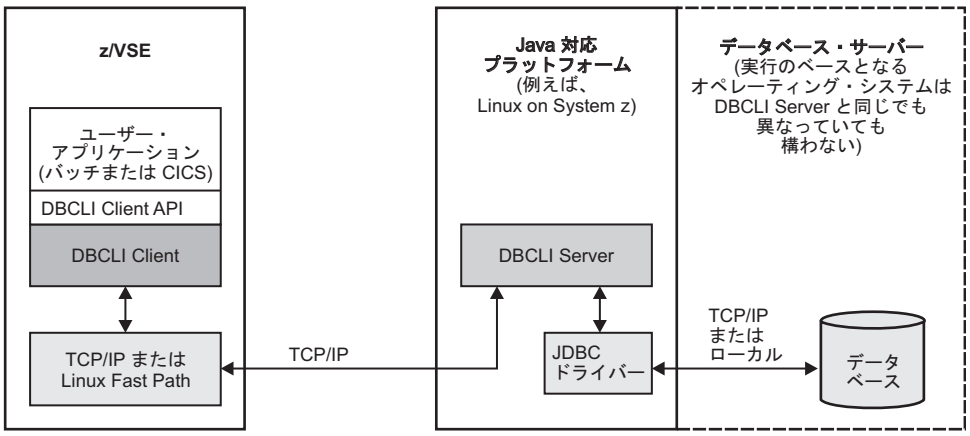
データベース呼び出しレベル・インターフェース の概要

z/VSE データベース呼び出しレベル・インターフェース (*DBCLI*) は、2 つの主要パートで構成されています。

- z/VSE で実行される *DBCLI* クライアント。
- Java プラットフォームで実行される *DBCLI* サーバー (プログラム `DBCliServer`)。

DBCLI クライアントは、アプリケーション・プログラム用のプログラミング API を提供します。

1. DBCLI クライアントは、TCP/IP 接続を介して DBCLI サーバーに接続します。
2. DBCLI クライアントは、アプリケーション・プログラムからの呼び出しを、DBCLI サーバーへの要求に変換します。
3. DBCLI サーバーは、アプリケーションから要求を受け取り、それを、ベンダー・データベースが提供する JDBC ドライバーに渡します。
4. DBCLI サーバーは、z/VSE で実行されているアプリケーション・プログラムへの呼び出しの結果を、DBCLI クライアント経由で返します。



DBCLI クライアントも DBCLI サーバーも、アプリケーション・プログラムで使用されている SQL ステートメントを検査しないため、ベンダーの JDBC ドライバー (IBM Db2、Oracle、Microsoft SQL Server、MySQL など) がサポートする SQL ステートメントや SQL ダイアレクトならどんなタイプでも 使用できます。ただし、データベース・プロバイダーが提供する JDBC ドライバーが適切であることが条件です。

接続プーリングの概要

接続プーリング では、CICS Transaction Server for z/VSE V1.1 下で実行されている DBCLI アプリケーション用の既存の データベース接続が保持および再利用されます。これは、頻繁に実行される、存続期間が短い CICS DBCLI アプリケーションに特に役立つ可能性があります。

接続プーリングを使用しないと、ネットワーク接続の確立と、データベース・サーバーへのアクセスの初期化に必要なオーバーヘッドのために、データベース接続の作成は比較的時間がかかる可能性があります。

接続プーリングの使用は、DBCLI アプリケーションには認識されません。デフォルトでは、接続プーリングは使用されません。したがって既存の DBCLI アプリケーションは、そのままの状態で (つまり接続プーリングを使用せずに) 作業を続行します。

z/VSE のインストール中、または高速サービス・アップグレード中に、ご使用の *DBDCCICS* CICS システムに必要な CICS/BSM テーブル更新が自動的に実行されます。

- 接続プーリングを別の CICS システムに実装したい場合は、そのシステム用に CICS/BSM テーブルを自分で更新する必要があります (92 ページの『別の CICS システム用に CICS/BSM テーブルを更新』を参照)。

注**:** バッチ・アプリケーションは、接続プーリングを使用できません。接続プーリングの使用を試行しても拒否されます。

関連トピック**:**

- 91 ページの『接続プーリング・マネージャーの構成と開始/停止』
- 325 ページの『接続プーリング』 (プログラミングの概念)。

データベース呼び出しレベル・インターフェースを使用する際の前提条件

データベース呼び出しレベル・インターフェース (DBCLI) の使用に関する前提条件は、以下のとおりです。

- データベース呼び出しレベル・インターフェースで使用するベンダー・データベースは独自に用意する必要があります。
- そのベンダー・データベース (IBM Db2、Oracle、Microsoft SQL Server、MySQL など) は、*JDBC V3.0* 以降をサポートする JDBC ドライバーを提供する必要があります。
- DBCLI サーバーには、Java Runtime (JRE) または Java Developer Kit (JDK) のバージョン 1.5 以降をサポートする Java 対応のプラットフォームが必要です。

DBCLI で接続プーリング を実装する場合は、DBCLI の接続プーリング・マネージャー を構成し、開始する必要があります。詳細については、 91 ページの『接続プーリング・マネージャーの構成と開始/停止』を参照してください。

DBCLI サーバーのインストール

このトピックでは、2 層または 3 層環境の物理層/論理層に DBCLI サーバーをインストールする方法について説明します。

DBCLI サーバーは VSE 中央機能に組み込まれていて、1 ファイル (**IESDBSRV.w**) で構成されています。

DBCLI サーバー のコピーの取得

DBCLI サーバー のコピーを取得するには、以下のいずれから取得するかを決定する必要があります。

- インターネットから
- Extended Base Tape から VSE コネクター・ワークステーション・コードをインストールすることによって

インターネットから DBCLI サーバー を取得するには、以下を行ってください。

1. Web ブラウザーを起動して次の URL に進みます。

   ```
   http://www.ibm.com/systems/z/os/zvse/downloads/
   ```

2. **Database Connector** セクションから、DBCLI サーバーをインストールするディレクトリーに、ファイル **DatabaseCliServer*nnn*.zip** をダウンロードします。注**:** *nnn* は現行 VSE バージョン (**DatabaseCliServer620.zip**) です。

DBCLI サーバー を *VSE* コネクター・ワークステーション・コード・コンポーネントをインストールすることによって 取得するには、以下を行ってください。

1. Extended Base Tape から VSE コネクター・ワークステーション・コード・コンポーネントをインストールします。このコンポーネントのインストール後、DBCLI サーバー の VSE コネクター・ワークステーション・コード **IESDBSRV.w** が、z/VSE サブライブラリー PRD2.PROD に保管されます。

2. TCP/IP for z/VSE の FTP (ファイル転送プログラム) ユーティリティーを使用して、DBCLI サーバー をインストールするディレクトリーに **IESDBSRV.w** をダウンロードします。次に、**IESDBSRV.w** を **DatabaseCliServer.zip** に名前変更します。

注**:**

1. **IESDBSRV.w** はバイナリー でダウンロードする必要があります。
2. UNIX モードがオフ になっていることを確認してください。 オフにしないと、バイナリー を指定しても、**IESDBSRV.w** は ASCII モードでダウンロードされます。 *Unix* モード は VSE FTP デーモンのパラメーターの 1 つです。 FTP クライアントの中には、UNIX モードを強制的に オンにするものがあります。 以下の例に、バッチ FTP クライアントを使用して正常に **IESDBSRV.w** を転送する方法を示します。UNIX モードが設定される場所は太字で示しています。

```
c:¥temp>ftp 9.164.155.2
Connected to 9.164.155.2.
220-TCP/IP for VSE -- Version 01.05.F  -- FTP Daemon
    Copyright (c) 1995,2006 Connectivity Systems Incorporated
220 Service ready for new user.
User (9.164.155.2:(none)): sysa
331 User name okay, need password.
Password:
230 User logged in, proceed.
ftp> cd prd2
250 Requested file action okay, completed.
ftp> cd prod
250 Requested file action okay, completed.
ftp> binary
200 Command okay.
ftp> get iesdbsrv.w
200 Command okay.
150-File: PRD2.PROD.IESDBSRV.W
    Type: Binary Recfm: FB Lrecl:    80 Blksize:      80
    CC=ON  UNIX=OFF RECLF=OFF TRCC=OFF CRLF=ON  NAT=NO
150 File status okay; about to open data connection
226-Bytes sent:       n,nnn,nnn
    Records sent:        nn,nnn
    Transfer Seconds:        nn.nn (    nnnK/Sec)
    File I/O Seconds:         n.nn (  n,nnnK/Sec)
226 Closing data connection.
nnnnnnn bytes received in nn,nn seconds (nnn,nn Kbytes/sec)
ftp> bye
221 Service closing control connection.
c:¥temp>ren IESDBSRV.w DatabaseCliServer.zip
```

DBCLI サーバー のインストールの実行

DBCLI サーバーのインストールを実行するには、以下の作業を行う必要があります。

1. ファイル **DatabaseCliServer.zip** を unzip します。これには以下のファイルが入っています。
   - setup.jar (DBCLI サーバー・コードを含む)
   - setup.bat または setup.cmd (Windows 用インストール・バッチ・ファイル)
   - setup.sh (Linux/Unix 用インストール・スクリプト)
2. (ファイルをダブルクリックして) オペレーティング・システム・プラットフォームに適用可能なバッチ・ファイルを開始します。

3. インストール・プロセスが開始され、さまざまなインストール・メニューによるガイドが表示されます。

DBCLI サーバー のアンインストール

DBCLI サーバー のアンインストールには、以下の 2 つの方法があります。

- 通常の *Windows* プログラムの追加/削除機能を使用。これは、Windows の「コントロール パネル」から選択します。
- DBCLI サーバー アンインストール・プログラムを手動で実行。 このプログラムは、**_uninst** サブディレクトリーにあります。このサブディレクトリーは、DBCLI サーバー をインストールしたディレクトリー内にあります。 このオプションは、すべてのプラットフォームで使用できます。

DBCLI サーバーのセットアップと構成

DBCLI サーバーは、Java 対応プラットフォームにインストールされます。インストールは、Java ベースのインストーラーで実行されます。このインストーラーはファイル setup.jar に含まれていて、以下のインストール・スクリプトのいずれかを実行して呼び出すことができます。

- setup.bat (Windows)
- setup.cmd (OS/2 および Windows)
- setup.sh (Linux、Unix)

Java インストーラーを実行すると、各種のダイアログ・ボックスが表示され、それに従ってインストールを進めることができます。

DBCLI サーバーの構成には以下のファイルを使用します。

- DatabaseCliServer.cfg
- JdbcAliases.cfg
- JdbcDriver.cfg

これらのファイルについては、これから詳しく説明します。

DatabaseCliServer.cfg

DatabaseCliServer.cfg 構成ファイルには、サーバーが listen するポート番号や、デフォルトのコード・ページなどの基本設定が含まれます。

注: プロパティー値はすべて 1 行で指定しなければなりません。下の例では、本書のページ幅の制限のために、いくつかの行は折り返されています。

```
########################################################################
# VSE Database CLI Server configuration
########################################################################

# Port where the Database CLI Server listens
listenport=16178

# Number of maximum parallel connections allowed.
maxconnections=256

# Optional. Address to bind the listening socket to.
#bindaddr=127.0.0.1
```

```
# Optional. Default EBCDIC codepage used with the VSE application.
#ebcdiccodepage=Cp1047

# Print messages (on) or do not print messages (off).
messages=on

# Print trace (on) or do not print trace (off).
trace=off

# Sets the trace level which can be:
# FLOW - trace the main application flow
# NORMAL - most messages
# FINE - most extensive tracing
tracelevel=NORMAL

# Optional. Configuration file for the JDBC alias definitions.
#jdbcaliases=JdbcAliases.cfg

# Configuration file for the JDBC drivers.
#jdbcdriver=JdbcDriver.cfg

######################################################################
#SSL specific settings (uncomment to activate):
######################################################################

# sslversion can be either SSL or TLS.
#sslversion=SSL

# If client authentication is true, the server requests the client to send its
certificate and verifies it.
# If client authentication is false or not specified, no client authentication
is done.
#clientauthentication=true

# keyringfile specifies the file name of the keyring file.
#keyringfile=keyring.pfx

# keyringpwd specifies the password for opening the keyring file.
#keyringpwd=ssltest

# ciphersuites specifies a comma separated list of cipher suites that are
accepted by the server.
# If this property is not specified, all supported cipher suites are used.
#ciphersuites=TLS_RSA_WITH_AES_256_CBC_SHA,TLS_RSA_WITH_AES_128_CBC_SHA
# For use with OpenSSL:
#ciphersuites=TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA,TLS_DHE_RSA_WITH_AES_256_CBC_SHA256,
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256,TLS_DHE_RSA_WITH_AES_256_CBC_SHA,
TLS_DHE_RSA_WITH_AES_128_CBC_SHA,SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA,
TLS_RSA_WITH_AES_256_CBC_SHA256,TLS_RSA_WITH_AES_128_CBC_SHA256,
TLS_RSA_WITH_AES_256_CBC_SHA,TLS_RSA_WITH_AES_128_CBC_SHA


#----------------------------------------------------------------------
# Available cipher suites with VSE:
#
# SSL_RSA_WITH_NULL_MD5             - deprecated
# SSL_RSA_WITH_NULL_SHA             - deprecated
# SSL_RSA_EXPORT_WITH_DES40_CBC_SHA - deprecated
# SSL_RSA_WITH_DES_CBC_SHA          - deprecated
# SSL_RSA_WITH_3DES_EDE_CBC_SHA     - deprecated
# TLS_RSA_WITH_AES_128_CBC_SHA
# TLS_RSA_WITH_AES_256_CBC_SHA
#
# The following cipher suites are only available with OpenSSL:
# TLS_RSA_WITH_NULL_SHA256          - deprecated
```

```
# TLS_RSA_WITH_AES_128_CBC_SHA256
# TLS_RSA_WITH_AES_256_CBC_SHA256
# SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
# TLS_DHE_RSA_WITH_AES_128_CBC_SHA
# TLS_DHE_RSA_WITH_AES_256_CBC_SHA
# TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
# TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
# TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
# TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
# TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
# TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
#------------------------------------------------------------------
```

JdbcAliases.cfg

JdbcAliases.cfg 構成ファイルには、接続先データベースの別名定義が含まれています。それぞれの別名は、JDBC ドライバーが定義する JDBC URL を使用して、データベースを指します。 z/VSE 上のアプリケーション・プログラムがデータベースへの接続を確立しようとする際には、DBCLI プログラミング・インターフェースの CONNECT 関数で別名を指定する必要があります。オプションで、別名ごとにユーザー名とパスワードを定義することができます。その場合、z/VSE で実行されるアプリケーション・プログラムは、CONNECT 関数でユーザー名とパスワードを指定する必要はありません。

DBCliServer は、JdbcAliases.cfg 構成ファイルを動的に再ロードできます。ファイルの再ロードには、「reload jdbcalias」コマンドを使用します。

```
SAMPLE.jdbcurl = jdbc:db2:SAMPLE


SAMPLE2.jdbcurl = jdbc:db2://ifranzki/SAMPLE
SAMPLE2.user = HUGO
SAMPLE2.password = PASSWORD
```

JdbcDriver.cfg

JdbcDriver.cfg 構成ファイルには、DBCliServer がロードする必要があるすべての JDBC ドライバーのクラス名が含まれています。また、その JDBC ドライバーでデータベースに接続する際に使用する JDBC URL の形式に関するヒントも含まれています。

JDBC ドライバーのクラスまたは JAR ファイルが CLASSPATH 環境変数に含まれていることを確認してください。 JDBC ドライバーが見つからない場合は、DBCliServer の始動時に該当するメッセージが表示されます。

接続プーリング・マネージャーの構成と開始/停止

このトピックでは、接続プーリング・マネージャー を構成、開始/停止、および照会する方法について説明します。

(DBCLI) 接続プーリング・マネージャーは、接続プーリングを管理する長期トランザクションです。

- 接続プーリング・マネージャーは、CICS 領域ごとに 1 つあります。
- 接続プーリング・マネージャーは、異なる DBCLI サーバーおよびデータベースに接続を「プール」できます。

関連トピック:
- 86 ページの『接続プーリングの概要』
- 325 ページの『接続プーリング』 (プログラミングの概念)。

別の CICS システム用に CICS/BSM テーブルを更新

DBDCCICS CICS システムの接続プーリング CICS/BSM テーブル項目は、z/VSE のインストール中、または高速サービス・アップグレード中に、自動的に作成されます。ただし、接続プーリングを別の CICS システムに実装したい場合は、その CICS システム用に CICS/BSM テーブルを更新する必要があります。

以下のプログラムを CICS に定義する必要があります。

```
IESDBCPM Language=Assembler, EXECKey=CICS, DataLoc=Any
IESDBCPP Language=Assembler, EXECKey=User, DataLoc=Any
IESDBCPQ Language=Assembler, EXECKey=User, DataLoc=Any
IESDBCPS Language=Assembler, EXECKey=User, DataLoc=Any
```

以下のトランザクションを CICS と基本セキュリティー・マネージャー (BSM) の両方に定義する必要があります。

```
IDBM Program=IESDBCPM, TaskDataKey=User, TaskdataLoc=ANY
IDBP Program=IESDBCPP, TaskDataKey=User, TaskdataLoc=ANY
IDBQ Program=IESDBCPQ, TaskDataKey=User, TaskdataLoc=ANY
IDBS Program=IESDBCPS, TaskDataKey=User, TaskdataLoc=ANY
```

上記の定義を実行できるように DBCLIDEF.JOB を取得するには、ご使用の Web ブラウザーを開始して、次の URL にアクセスする必要があります。

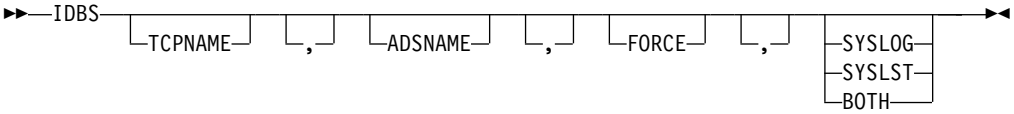
```
http://www.ibm.com/systems/z/os/zvse/downloads/
```

接続プーリング・マネージャーの開始

DBCLI 接続プーリング・マネージャーは、トランザクション IDBS を使用して手動で開始するか、CICS 始動時に自動で開始できます。DBCLI 接続プーリング・マネージャーを自動で開始する場合、EZATRUE プログラムを同時に開始できます。

接続プーリング・マネージャーを手動で開始 するには、トランザクション **IDBS** を使用します。このトランザクションは、システム・コンソールまたは他の端末から呼び出すことができます。

注: 接続プーリング・マネージャーを開始する前に、EZA タスク関連ユーザー出口 (EZATRUE) を開始しておく必要があります。EZATRUE の開始方法の詳細については、「*IBM z/VSE TCP/IP* サポート」の『CICS Considerations for the EZA Interfaces』を参照してください。



各部の説明:

TCPNAME

DBCLI アプリケーションで使用されるローカルの TCP/IP スタックを指定します。このパラメーターを SOCKET*nn* に設定できます。また、単に *nn*

(左または右寄せにして 6 個のブランクを埋め込む) に設定することもできます。*nn* の値は TCP/IP 始動ジョブ内の ID パラメーターで指定されるため、この値によって、選択された TCP/IP スタックの ID が決まります。TCPNAME が指定されていない場合、`// OPTION SYSPARM='nn'` ステートメントの ID が使用されます。上記のどれも指定されていない場合、ID のデフォルトは `'00'` になります。

ADSNAME
EZA 処理環境で使用される TCP/IP インターフェース・ルーチンの名前を指定します。デフォルトでは、IBM 提供の TCP/IP インターフェース・ルーチン EZASOH99 が使用されます。ステートメント `// SETPARM EZA$PHA='routine-name'` を使用して、この仕様を上書きできます。

FORCE
プーリング・マネージャーが前にキャンセルされている場合は接続マネージャー・プールを強制的に開始する必要があることを指定します。

注**:** この開始オプションは、接続プーリング・マネージャーが応答しなくなった場合にのみ、注意して 使用する必要があります。この開始オプションによってアクティブな接続プーリング・マネージャーが破棄される場合があります。

SYSLOG | SYSLST | BOTH
接続プーリング・マネージャーのトレースをアクティブにする必要があることを指定します。SYSLOG の場合、トレース・メッセージはシステム・コンソールに書き込まれます。SYSLST の場合、トレース・メッセージは CICS リストに書き込まれます。BOTH の場合、トレース・メッセージはシステム・コンソールと CICS リストに書き込まれます。

CICS の始動時に EZATRUE と接続プーリング・マネージャーを自動で開始 するには、以下のステートメントを使用して、プログラム EZASTRUE と IESDBCPS を表 DFHPLTPI に追加します。

```
DFHPLT TYPE=ENTRY,PROGRAM=EZASTRUE
DFHPLT TYPE=ENTRY,PROGRAM=IESDBCPS
```

注**:** 接続プーリング・マネージャーと ESATRUE を自動で開始する場合は、前記のどの始動パラメーターも指定できません。

接続プーリング・マネージャーの停止

DBCLI 接続プーリング・マネージャーは、トランザクション IDBP を使用して手動で停止するか、CICS シャットダウン時に自動で停止できます。

接続プーリング・マネージャーを手動で停止 するには、トランザクション **IDBP** を使用します。このトランザクションは、システム・コンソールまたは他の端末から呼び出すことができます。現在プールされているすべての接続が閉じられます。必要であれば、EZATRUE を手動で停止することもできます。詳細については、「*IBM z/VSE TCP/IP* サポート」の『CICS Considerations for the EZA Interfaces』を参照してください。

CICS シャットダウン時に接続プーリング・マネージャーと EZATRUE を自動で停止 するには、以下のステートメントを使用して、プログラム IESDBCPP と EZASTRUE を表 DFHPLTSD に追加します。

```
DFHPLT TYPE=ENTRY,PROGRAM=IESDBCPP
DFHPLT TYPE=ENTRY,PROGRAM=EZASTRUE
```

接続プーリングの照会

DBCLI 接続プーリングを照会するには、トランザクション **IDBQ** を使用します。このトランザクションは、システム・コンソールまたは他の端末から呼び出すことができます。

トランザクション IDBQ がシステム・コンソールから使用された場合、現在プールされている接続のリストがシステム・コンソールに表示されます。

トランザクション IDBQ が端末から使用された場合、現在プールされている接続のリストが 3270 画面に表示されます。その後、このリストをスクロールできます。

DBCLI サーバー コマンド

DBCLI サーバー (`DBCliServer`) が稼働中の場合、以下のコマンドを入力してサーバーを操作できます。

- **status**: すべての接続の状況を表示します
- **stop all** : すべてのアクティブな接続を切断します
- **stop <n>**: 指定された接続を切断します
- **quit [force]**: サーバーを終了します
- **trace <on|off>**: トレースのオンとオフを切り替えます
- **show jdbcalias**: JDBC 別名構成を表示します
- **update jdbcalias**: JDBC 別名構成を再ロードします
- **help**: ヘルプ情報を表示します

第 10 章 Db2 ベース・コネクターのカスタマイズ

このトピックでは、Db2 ベース・コネクターを使用する前に必要なカスタマイズのアクティビティーについて説明します。

以下の項目があります。

- 『Db2 ベース・コネクター の概要』
- 96 ページの『完了しておくべきホストのインストール・アクティビティー』
- 97 ページの『ステップ 1: CICS TS をカスタマイズする』
- 97 ページの『ステップ 2: TCP/IP をカスタマイズする』
- 97 ページの『ステップ 3: Db2 をカスタマイズしてサンプル・データベースを定義する』
- 106 ページの『ステップ 4: DRDA サポートのセットアップ』
- 106 ページの『ステップ 5: ストアード・プロシージャー・サーバーのセットアップと Db2 への定義』
- 107 ページの『ステップ 6: ストアード・プロシージャーをセットアップする』
- 108 ページの『ステップ 7: VSAM データ・アクセス用に Db2 ベース・コネクターをカスタマイズする』
- 108 ページの『ステップ 8: DL/I データ・アクセス用に Db2 ベース・コネクター をカスタマイズする』
- 109 ページの『ステップ 9: Db2 の開始と、ストアード・プロシージャー・サーバーの始動』
- 110 ページの『ステップ 10: Db2 Connect のインストールとクライアント/ホスト接続の確立』

Db2 ベース・コネクター の概要

Db2 ベース・コネクター は、分散リレーショナル・データベース体系 (DRDA) を使用して、VSE/VSAM および DL/I データなどの非リレーショナル・データにアクセスできるようにするオプション・フィーチャーです。 お客様のアプリケーション・プログラムは、JDBC、ODBC、またはコール・レベル・インターフェース (CLI) などの標準インターフェースを使用してデータを要求します。

Db2 ベース・コネクター のインプリメンテーションは、基本として *Db2* ストアード・プロシージャー を使用します。これは、Db2 Server for VSE & VM、バージョン 6 以降で使用できます。Db2 ストアード・プロシージャーは、お客様が作成してコンパイルし、z/VSE ホストに保管するアプリケーション・プログラムです。

Db2 ベース・コネクターを使用するには、*Db2 Server for VSE* が z/VSE 上で実行されている必要があります。

注: Db2 Server for VSE Client Edition は、Db2 ストアード・プロシージャーを z/VSE 上で実行できないため、Db2 Server for VSE Client Editionは、Db2 ベース・コネクターと使用するには十分ではありません。

Db2 ストアード・プロシージャーは、任意の LE (言語処理環境) 対応言語 (COBOL、C、または PL/I) で作成できます。それにより、ローカルまたはリモート DRDA アプリケーションはこれらの Db2 ストアード・プロシージャーを呼び出すことができます。

Db2 ベース・コネクターでは、以下のようにして、Db2 データへのアクセスに使用しているものと同じ Db2 ストアード・プロシージャー内から VSE/VSAM および DL/I データにアクセスできます。

- VSE/VSAM データにアクセスするには、*VSAM* コール・レベル・インターフェース を使用します。 詳細については、 451 ページの『Db2 ストアード・プロシージャーによる VSAM データへのアクセス』を参照してください。
- DL/I データにアクセスするには、*AIBTDLI* インターフェース を使用します。詳細については、 459 ページの『Db2 ストアード・プロシージャーによる DL/I データへのアクセス』を参照してください。

3 層環境内で Db2 ベース・コネクターが使用される場面の概説については、 11 ページの図 3を参照してください。

完了しておくべきホストのインストール・アクティビティー

このトピックでは、Db2 ベース・コネクターのカスタマイズを開始する前に z/VSE ホストで完了しておく必要があるインストール・アクティビティーについて簡単に説明します。

- Db2 Server for VSE は、z/VSE Extended Base Tape からサブライブラリー PRD2.DB2750 にリストアする必要があります。Db2 Server for VSE のスタートアップ・ジョブは、クラス S の動的区画に定義されます。次のいずれかでインストールされます。
  - z/VSE の初期インストール時。
  - FSU (高速サービス・アップグレード) の後。
- Db2 ベース・コネクターを使用するには、*Db2 Server for VSE* が z/VSE 上で実行されている必要があります。

  注: Db2 Server for VSE Client Edition は、Db2 ストアード・プロシージャーを z/VSE 上で実行できないため、Db2 Server for VSE Client Editionは、Db2 ベース・コネクターと使用するには十分ではありません。
- AIBTDLI インターフェースは、Db2 ストアード・プロシージャーを介した DL/I データのアクセス用にインストールする必要があります。AIBTDLI インターフェースを使用するには、以下の作業を行います。
  - DL/I VSE をインストールする必要があります。
  - CICS/DLI システムでは、AIBTDLI インターフェースと合わせて、使用するすべてのデータベース (DBD) を CICS FCT に定義しておく必要があります。
  - CICS/DLI システムには、以下が必要です。

- DL/I オンライン中核 DLZNUCxx にすべての PSB が定義済みである。
- アクティブな MPS システム。

– DL/I タスク終了出口 *DLZBSEOT* (469 ページの『タスクの終了および異常終了の処理』で説明) を SVA 内に常駐させておく必要があります。

• Db2 Server for VSE で使用する CICS TS システムを 1 つ以上カスタマイズする必要があります。

ステップ 1: CICS TS をカスタマイズする

複数の CICS TS を実行する場合、*Db2 Server for VSE* にアクセスする CICS システム (1 つ以上) を先に決定する必要があります。選択した CICS TS をカスタマイズするには、以下の作業を行う必要があります。

• ジャーナリングをアクティブ にします。 対応する DFHSITxx スケルトンを使用して JCT=**SP** (DFHJCT**SP** の場合) または別のサフィックスを設定します。
• 変更したスケルトン DFHJCTSP を使用して JCT をコンパイルします。
• ジャーナル・ファイルを定義します。 CICSICCF の場合は付属のスケルトン SKJOURN を使用します。 PRODCICS の場合は付属のスケルトン SKJOUR2 を使用します。 いずれも VSE/ICCF ライブラリー 59 から入手できます。

ステップ 2: TCP/IP をカスタマイズする

TCP/IP スタートアップ・ジョブに以下の 2 つの値が設定されていることを確認します。

```
SET WINDOW              = 8192
SET MAX_SEGMENT         = 700
```

編集には、クライアント・ワークステーションで TCP/IP ダイアログを使用するか、VSE ライブラリー内の該当するスタートアップ・メンバーを変更します。

詳しくは、「*IBM z/VSE TCP/IP* サポート」を参照してください。

ステップ 3: Db2 をカスタマイズしてサンプル・データベースを定義する

ステップ 3 では、**SKDB2VAR** (区画 *BG* 内) を実行します。これは、Db2 ベース・コネクターのカスタマイズに使用するメイン・スケルトンです。 ここでは、このスケルトンに含まれるジョブについて説明します。

注**:**

1. このステップを始める前に、Db2 を使用するためのライセンス・キーを有効にしておく必要があります。有効ではない場合は、スケルトン SKUSERBG (ICCF ライブラリー 59 にある) を使用してこのキーを有効にする方法について、「*IBM z/VSE* 計画」を参照してください。
2. スケルトン SKDB2VAR では、ストレージ割り振りに IBM 3380 ディスク装置を使用することを想定しています。 別のディスク装置タイプを使用する場合は、それに応じてこれらの割り振りを変更する必要があります (特に FBA 装置の場合)。

スケルトン SKDB2VAR が実行するジョブは、以下のとおりです。

ステップ 3.1: ユーザー・カタログを定義する

ジョブ **DB2DEFCT** は、別のボリュームにユーザー・カタログおよびスペースを定義します。

DB2UCAT カタログの場合、割り振りスペースは 150 シリンダーで構成されます。

新規カタログに **DB2UCAT** の名前の標準ラベルが挿入されます。 以下の変数に独自の値を入力する必要があります。

- -V001- -
: ご使用のボリュームの ID。 この VOLID はジョブ DB2CTVAR でも使用されます。

- -V002- -
: カタログ・スペースに割り振られるトラックの数。 割り振りは、150 シリンダーです。

```
$$ JOB JNM=DB2DEFCT,CLASS=0,DISP=D,NTFY=YES
$$ LST CLASS=Q,DISP=H
// JOB DB2DEFCT DEFINE USER CATALOG DB2UCAT
* THIS JOB WILL TERMINATE IN CASE THE DB2UCAT IS ALREADY DEFINED.
// EXEC IDCAMS,SIZE=AUTO
  LISTCAT ALL CATALOG(DB2.USER.CATALOG)
  IF LASTCC EQ 8 THEN DO
      SET LASTCC = 0
      SET MAXCC = 0
   DEFINE USERCATALOG(NAME(DB2.USER.CATALOG) -
      VOL(--V001--)                          -
      NOTRECOVERABLE                         -
      TRK(15))
   DEFINE SPACE(VOLUMES(--V001--)            -
      CYL(--V002--))                         -
      CATALOG(DB2.USER.CATALOG)
    END
   ELSE DO
      SET LASTCC = 0
      SET MAXCC = 0
   END
/*
// OPTION STDLABEL=DELETE
DB2UCAT
/*
// OPTION STDLABEL=ADD
// DLBL DB2UCAT,'DB2.USER.CATALOG',0,VSAM
/*
// EXEC IESVCLUP,SIZE=AUTO             ADD LABEL TO STDLABUP PROC
```

```
D                                                DB2UCAT
D DB2.SQLGLOB.MASTER                             SQLGLOB
A DB2.USER.CATALOG                               DB2UCAT
A DB2.SQLGLOB.MASTER                             SQLGLOB DB2UCAT OLD KEEP
/*
/&
$$ EOJ
```

ステップ 3.2: 新規 ARISIVAR.Z をカタログする

このステップでは、ジョブ DB2CTVAR は、まず Db2 提供のオリジナルの ARISIVAR.Z を ARISIVAR.ORIG に名前変更し、グローバル変数メンバー ARISIVAR.Z をカタログします。次に、新規 ARISIVAR.Z を使用して Db2 ベース・コネクターのインストールおよびサンプル・データベースがテストされます。

ARISIVAR.Z の処理が **Db2** ジョブ・マネージャーによって制御されることに注意してください。

(スケルトン SKDB2VAR によって配置された) VSE/POWER 読み取り待ち行列内のジョブ **DB2JMGR** をリリースしてジョブ・マネージャーを開始します。 準備、インストール (またはマイグレーション) のステップごとにジョブ・マネージャーをリリースする必要があります。

Db2 Server for VSE のプログラム・ディレクトリー の詳細説明にあるように、ARISIVAR.Z は多くのパラメーター、グローバル、および変数を処理し、Db2 特性およびリソースを定義します。

主要な定義には、例えば、以下が含まれます。

- Db2 サンプル・データベース **SQLDS**。ボリューム **- -V001- -** (スケルトン SKDB2VAR 内の変数) で定義されます。
- **Db2 Server for VSE Help** コンポーネント。以下の変数によって制御されるインストール。

  ```
  ARIS75JZ HELP       YES
  ```

  Db2 Server for VSE Help のインストールを推奨しています (**- -V003- -** をテープ・ドライブのアドレスに置き換える必要があります)。 この変数が処理されるときに、対応するテープ (Db2 ヘルプ・ファイルが含まれるもの) を取り付けるように要求が出されます。これは、お客様が受け取っている基本配布テープの 3 番目のテープ (追加テープ) になります。
- **BINDFILE**、**BINDWFILE**、および **SQLGLOB** などの Db2 (作業) ファイルの作成。
- Db2 環境に必要な **CICS TS** パラメーターの設定。

以下に、DB2CTVAR の内容を示します。

```
$$ JOB JNM=DB2CTVAR,CLASS=0,DISP=D,NTFY=YES
$$ LST CLASS=Q,DISP=H
// JOB DB2CTVAR CATALOG GLOBAL VARIABLE MEMBER FOR DB2
* ORIGINAL ARISIVAR.Z RENAMED TO ARISIVAR.ORIG
// EXEC  LIBR,PARM='MSHP'
    ACCESS  SUBLIB = PRD2.DB2750
    RENAME           ARISIVAR.Z:=.ORIG
CATALOG ARISIVAR.Z        EOD=&&          REPLACE=YES
```

```
⋮

 (for further details, refer to the sample SKDB2VAR in Library 59)
```

ステップ 3.3: 準備/インストール・ステップ用のジョブ・マネージャー

```
$$ JOB JNM=DB2JMGR,CLASS=R,DISP=L,NTFY=YES
$$ LST CLASS=Q,DISP=H
// JOB DB2JMGR DB2 JOB MANAGER
// LIBDEF *,SEARCH=(PRD2.DB2750)
// EXEC REXX=ARISIMGR
/*
/&
$$ EOJ
```

ジョブ DB2JMGR は 101 ページの『ステップ 3.6: Db2 サンプル・データベースの準備』および 102 ページの『ステップ 3.7: Db2 サンプル・データベースのインストール』で使用されます。

ステップ 3.4: DRDA サーバー・サポートをアクティブにする

このジョブは、以降のステップ 4 で使用されます。

```
$$ JOB JNM=DB2DRDA,CLASS=R,DISP=L,NTFY=YES
$$ LST CLASS=Q,DISP=H
// JOB DB2DRDA ACTIVATE DRDA SERVER SUPPORT
* $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
*  LINK EDIT RDS WITH DRDA SERVER SUPPORT
* $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
// LIBDEF *,SEARCH=PRD2.DB2750
// LIBDEF PHASE,CATALOG=PRD2.DB2750
// OPTION CATAL
 INCLUDE ARISLKRA
// EXEC  PGM=LNKEDT,PARM='MSHP,AMODE=31,RMODE=ANY'
/*
/&
$$ EOJ
```

ステップ 3.5: ストアード・プロシージャー・サーバー用のスタートアップ・ジョブ

このステップでは、スケルトン SKDB2VAR は、ストアード・プロシージャーのサーバーを始動するジョブを VSE/POWER 読み取り待ち行列にロードします。

```
$$ JOB JNM=PSERVER,CLASS=0,DISP=L
$$ LST CLASS=W,DISP=H
// JOB PSERVER
// LIBDEF PROC,SEARCH=(PRD2.DB2750)
// DLBL SQLGLOB,'DB2.SQLGLOB.MASTER',,VSAM,CAT=DB2UCAT,DISP=(OLD,KEEP)
// EXEC PROC=ARIS75PL        *-- DB2 PROD. LIBRARY ID PROC
// EXEC PROC=ARIS75DB        *-- DB2 DATABASE ID PROC
// EXEC  ARIDBS,SIZE=AUTO
CONNECT SQLDBA IDENTIFIED BY SQLDBAPW;
CREATE PSERVER SPSERV01 AUTOSTART YES;
/*
/&
$$ EOJ
```

ステップ 3.6: Db2 サンプル・データベースの準備

このジョブは、ARISIVAR.Z 内に含まれる準備関連のグローバル定義 (ステップ 3.2 でカタログ済み) を使用します。 準備ステップを実行するには、DB2 ジョブ・マネージャー (DB2JMGR) をリリースする必要があります。 このステップの実行方法について詳しくは、下記のコンソール・リストを参照してください。

注: 準備 ジョブを実行するには、**0** (区画 **BG**) を入力する必要があります。下記のサンプルでは、DB2 ジョブ・マネージャーは区画 4 で実行され、Db2 Server for VSE バージョン 7.5 を使用します。

```
r rdr,db2jmgr
AR 0015 1C39I COMMAND PASSED TO VSE/POWER
F1 0001 1R88I  OK
F4 0001 1Q47I F4 DB2JMGR 00249 FROM (HEHA) , TIME= 9:28:39, TKN=000001A7
F4 0004 // JOB DB2JMGR DB2 JOB MANAGER
        DATE 07/07/2001, CLOCK 09/28/39
F4 0004 ************************************************************
F4 0004       PREPARE FOR INSTALLATION/MIGRATION PROCESS
F4 0004 ************************************************************
F4 0004 ENTER INSTALLATION LIBRARY NAME (PRD2.DB2750   default)
F4-0004
4
F4 0004 YOU HAVE SELECTED  PRD2.DB2750
F4 0004 PRESS ENTER TO CONTINUE OR ENTER ANY OTHER KEY TO
F4 0004 MODIFY YOUR SELECTION:
F4-0004
4
F4 0004 WHICH CLASS WILL YOU USE TO RUN THE PROCESS ?  (4  default)
F4-0004
4 0
F4 0004 YOU HAVE SELECTED  0
F4 0004 PRESS ENTER TO CONTINUE OR  ENTER ANY OTHER KEY TO
F4 0004 MODIFY YOUR SELECTION:
F4-0004
4
F4 0004 PLEASE SELECT ONE OF THE FOLLOWING :
F4 0004 FOR PREPARATION....  ENTER (P)
F4 0004 FOR INSTALLATION...  ENTER (I)
F4 0004 FOR MIGRATION......  ENTER (M)
F4-0004
F4-0004
4 p
F4 0004 IF PREPARATION FOR:
F4 0004 INSTALLATION... PLEASE ENTER (I)
F4 0004 MIGRATION...... PLEASE ENTER (M)
F4-0004
4 i
F4 0004 DO YOU WANT TO EXECUTE ALL JOBS? {Y|N-default}
F4-0004
4 y
F4 0004
F4 0004 ************ JOB ARIS75JD ********************
F4 0004 *   DEFINE DB2750 PROGRAMS AND TRANSACTIONS
F4 0004 ***********************************************
F4 0004 JOB ARIS75JD IS OPTIONAL.
F4 0004 WOULD YOU LIKE TO RUN IT? {Y|N-Default}
F4-0004
4 y
BG 0001 1Q47I   BG ARIS75JD 00281 FROM (HEHA) , TIME=11:12:37, TKN=000001A8
BG 0000 * ** JOB JNM=ARIS75JD,CLASS=0,DISP=D
BG 0000 * ** LST CLASS=V,DISP=D,DEST=(,XXXXXXX)
⋮
F4 0004 JOB ARIS759D IS OPTIONAL.
F4 0004 WOULD YOU LIKE TO RUN IT? {Y|N-Default}
F4-0004
4 y
BG 0001 1Q47I   BG ARIS759D 00285 FROM (HEHA) , TIME=11:13:10, TKN=000001A9
```

```
BG 0000 // JOB ARIS759D
BG 0000 * ************************************************************
BG 0000 * ARIS759D: DEFINE VSAM CLUSTER FOR THE BINDWKF FILE
BG 0000 * ************************************************************
BG 0000 EOJ ARIS759D  MAX.RETURN CODE=0000
BG 0000 EOJ NO NAME
BG 0001 1Q34I   BG WAITING FOR WORK
F4 0004 *********************************************
F4 0004 * Job  ARIS759D  executed successfully
F4 0004 *********************************************
F4 0004
F4 0004 ************ JOB ARISIQBD ********************
F4 0004 *   ISQL BIND FILE CONVERSION
F4 0004 *********************************************
F4 0004 JOB ARISIQBD IS OPTIONAL.
F4 0004 WOULD YOU LIKE TO RUN IT? {Y|N-Default}
F4-0004
4 y
BG 0001 1Q47I   BG ARISIQBD 00286 FROM (HEHA) , TIME=11:13:20, TKN=000001AA
BG 0000 // JOB ARISIQBD -- ISQL BIND FILE CONVERSION
BG-0000 // PAUSE
0
BG 0000 EOJ ARISIQBD  MAX.RETURN CODE=0000
BG 0000 EOJ NO NAME
BG 0001 1Q34I   BG WAITING FOR WORK
F4 0004 *********************************************
F4 0004 * Job  ARISIQBD  executed successfully
F4 0004 *********************************************
F4 0004
F4 0004 ************ JOB ARIS75CD ********************
F4 0004 *   DB2 SERVER STARTER DATABASE VSAM DEFINITIONS
F4 0004 *********************************************
BG 0001 1Q47I   BG ARIS75CD 00294 FROM (HEHA) , TIME=11:28:08, TKN=000001AB
BG 0000 // JOB ARIS75CD              DB2 FOR VSE STARTER DB VSAM DEFINITIONS
BG 0000 EOJ ARIS75CD  MAX.RETURN CODE=0
F4 0004
F4 0004 ************ JOB ARISSTDL ********************
F4 0004 *   ADD NEW LABELS TO STANDARD LABELS
F4 0004 *********************************************
F4 0004 JOB ARISSTDL IS OPTIONAL.
F4 0004 WOULD YOU LIKE TO RUN IT? {Y|N-Default}
F4-0004
4 y
BG 0001 1Q47I   BG ARISSTDL 00297 FROM (HEHA) , TIME=11:30:06, TKN=000001AC
BG 0000 // JOB ARISSTDL
BG 0000 EOJ ARISSTDL
BG 0000 EOJ NO NAME
BG 0001 1Q34I   BG WAITING FOR WORK
F4 0004 READ CONSOLE FOR DETAILS.
F4 0004 EOJ DB2JMGR   MAX.RETURN CODE=0000
```

ステップ 3.7: Db2 サンプル・データベースのインストール

このジョブは、ARISIVAR.Z 内に含まれるインストール関連のグローバル定義 (ステップ 3.2 でカタログ済み) を使用します。 インストール・ステップを実行するには、Db2 ジョブ・マネージャー (DB2JMGR) をリリースする必要があります。このステップの実行方法について詳しくは、下記のコンソール・リストを参照してください。

注: このインストール・ステップは、静的 区画で実行する必要があります (下記の例は、静的区画 **F4** を使用します)。下記のサンプルでは、Db2 ジョブ・マネージャーは区画 F7 で実行されます。

```
r rdr,db2jmgr
AR 0015 1C39I COMMAND PASSED TO VSE/POWER
F1 0001 1R88I  OK
F7 0007 // JOB DB2JMGR DB2 JOB MANAGER
```

```
F7 0007 ************************************************************
F7 0007       PREPARE FOR INSTALLATION/MIGRATION PROCESS
F7 0007 ************************************************************
F7 0007 ENTER INSTALLATION LIBRARY NAME (PRD2.DB2750   default)
F7-0007
7
F7 0007 YOU HAVE SELECTED  PRD2.DB2750
F7 0007 PRESS ENTER TO CONTINUE OR ENTER ANY OTHER KEY TO
F7 0007 MODIFY YOUR SELECTION:
F7-0007
7
F7 0007 WHICH CLASS WILL YOU USE TO RUN THE PROCESS ?  (4  default)
F7-0007
7 4
F7 0007 YOU HAVE SELECTED  4
F7 0007 PRESS ENTER TO CONTINUE OR  ENTER ANY OTHER KEY TO
F7 0007 MODIFY YOUR SELECTION:
F7-0007
7
F7 0007 PLEASE SELECT ONE OF THE FOLLOWING :
F7 0007 FOR PREPARATION....  ENTER (P)
F7 0007 FOR INSTALLATION...  ENTER (I)
F7 0007 FOR MIGRATION......  ENTER (M)
F7-0007
7 i
F7 0007 DO YOU WANT TO EXECUTE ALL JOBS? {Y|N-default}
F7-0007
7 y
F7 0007
F7 0007 ************ JOB ARISBDID ********************
F7 0007 * SETUP THE DBNAME DIRECTORY
F7 0007 *******************************************
F7 0007 JOB ARISBDID IS OPTIONAL.
F7 0007 WOULD YOU LIKE TO RUN IT? {Y|N-Default}
F7-0007
7 y
F4 0001 1Q47I   F4 ARISBDID 00300 FROM (HEHA) , TIME=11:36:54, TKN=000001AD
F4 0004 // JOB ARISBDID -- DBNAME DIRECTORY SERVICE GENERATION
F4 0004 * ********************************************************************
F4 0004 *                                                                  *
F4 0004 *  THIS JCL EXECUTES STEPS TO GENERATE THE DBNAME DIRECTORY SERVICE *
F4 0004 *  ROUTINE (ARICDIRD.PHASE).                                       *
F4 0004 *                                                                  *
F4 0004 *  STEP 1 EXECUTES THE PROCEDURE ARICCDID FOR MIGRATION            *
F4 0004 *  OR ARICBDID FOR INSTALLATION TO READ THE DBNAME                 *
F4 0004 *  DIRECTORY SOURCE MEMBER ARISDIRD.A FROM THE PRODUCTION LIBRARY,  *
F4 0004 *  AND GENERATES THE ASSEMBLER VERSION OF ARISDIRD ON SYSPCH.      *
F4 0004 *                                                                  *
F4 0004 * ********************************************************************
F4 0004 * ********************************************************************
F4 0004 * ARISBDID  STEP 1 -- BUILD ASSEMBLER VERSION OF ARISDIRD           *
F4 0004 * ********************************************************************
F4 0004 * SQL/DS DBNAME DIRECTORY BUILT SUCCESSFULLY
F4 0004 EOJ ARISBDID  MAX.RETURN CODE=0000
F7 0007 *********************************************
F7 0007 * Job  ARISBDID  executed successfully
F7 0007 *********************************************
F7 0007
F7 0007 ************ JOB ARIS75BD ********************
F7 0007 * LINK EDIT DB2 SERVER ONLINE SUPPORT COMPONENTS
F7 0007 *********************************************
F7 0007 JOB ARIS75BD IS OPTIONAL.
F7 0007 WOULD YOU LIKE TO RUN IT? {Y|N-Default}
F7-0007
7 y
F4 0001 1Q47I   F4 ARIS75BD 00301 FROM (HEHA) , TIME=11:37:07, TKN=000001AE
F4 0004 // JOB ARIS75BD     LINK EDIT DB2 FOR VSE ONLINE SUPPORT COMPONENTS
F4 0004 * ****************************************************
F4 0004 * ARIS090D: LINK EDIT SQL/DS ONLINE RESOURCE ADAPTER CONTROL
F4 0004 * ****************************************************
F4 0004 * ****************************************************
```

```
F4 0004 * ARIS140D: LINK EDIT ISQL
F4 0004 * ************************************************************
F4 0004 * ************************************************************
F4 0004 * ARIS150D: LINK EDIT ISQL ITRM TERMINAL TRANSACTION
F4 0004 * ************************************************************
F4 0004 * ************************************************************
F4 0004 * ARIS160D: LINK EDIT ISQL ITRM TERMINAL EXTENSION PROGRAM
F4 0004 * ************************************************************
F4 0004 EOJ ARIS75BD  MAX.RETURN CODE=0000
F4 0004 EOJ NO NAME
F4 0001 1Q34I   F4 WAITING FOR WORK
F7 0007 *********************************************
F7 0007 * Job  ARIS75BD  executed successfully
F7 0007 *********************************************
F7 0007
F7 0007 ************ JOB ARIS75DD ********************
F7 0007 * DATABASE DBGEN AND SET UP
F7 0007 *********************************************
F4 0001 1Q47I   F4 ARIS75DD 00302 FROM (HEHA) , TIME=11:37:20, TKN=000001AF
F4 0004 // JOB ARIS75DD               DATABASE DBGEN AND SET UP
F4 0004 * ************************************************************
F4 0004 * ARIS75SL: DB2 SERVICE/PRODUCTION LIBRARY DEFINITION
F4 0004 * ************************************************************
F4 0004 * ************************************************************
F4 0004 * ARIS75DB: DB2 STARTER DATABASE IDENTIFICATION
F4 0004 * ************************************************************
F4 0004 * ************************************************************
F4 0004 * ARIS030D: GENERATE THE STARTER DATABASE
F4 0004 * ************************************************************
F4 0004 ARI0025I The program ARISQLDS is loaded at 400078.
F4 0004 ARI0025I The program ARICMOD is loaded at 564D80.
F4 0004 ARI0025I The program ARIXSXR is loaded at 581C00.
F4-0004 ARI0919D Database generation invoked.
                 The database will be formatted and the original
                 database destroyed.

                 Enter either:
                   DBGEN   to continue, or
                   CANCEL  to cancel.
4 dbgen
F4 0004 System identification at DB generation = DB2 VSE & VM 7.5
F4 0004
⋮
F7 0007 *********************************************
F7 0007 * Job  ARIS75DD  executed successfully
F7 0007 *********************************************
F7 0007
F7 0007 ************ JOB ARIS75ED ********************
F7 0007 * INSTALL DATABASE COMPONENTS (ISQL, FIPS FLAGGER)
F7 0007 *********************************************
F7 0007 JOB ARIS75ED IS OPTIONAL.
F7 0007 WOULD YOU LIKE TO RUN IT? {Y|N-Default}
F7-0007
7 y
F4 0001 1Q47I   F4 ARIS75ED 00303 FROM (HEHA) , TIME=11:45:41, TKN=000001B0
F4 0004 // JOB ARIS75ED     INSTALL DATABASE COMPONENTS
F4 0004 * ************************************************************
F4 0004 * ARIS75SL: DB2 SERVICE/PRODUCTION LIBRARY DEFINITION
F4 0004 * ************************************************************
F4 0004 * ************************************************************
F4 0004 * ARIS75DB: DB2 STARTER DATABASE IDENTIFICATION
F4 0004 * ************************************************************
F4 0004 * ************************************************************
F4 0004 * ARIS080D: GRANT SCHEDULE AUTHORITY TO DBDCCICS
F4 0004 * ************************************************************
⋮
F4 0001 1Q34I   F4 WAITING FOR WORK
F7 0007 *********************************************
F7 0007 * Job  ARIS75WD  executed successfully
F7 0007 *********************************************
F7 0007 ************ JOB ARIS75HZ ********************
```

```
F7 0007 * ENLARGE HELP TEXT DBSPACE
F7 0007 *********************************************
F7 0007 JOB ARIS75HZ IS OPTIONAL.
F7 0007 WOULD YOU LIKE TO RUN IT? {Y|N-Default}
F7-0007
7 n
F7 0007
F7 0007 ************ JOB ARIS75JZ ********************
F7 0007 * INSTALL LANGUAGE
F7 0007 *********************************************
F7 0007 JOB ARIS75JZ IS OPTIONAL.
F7 0007 WOULD YOU LIKE TO RUN IT? {Y|N-Default}
F7-0007
7 n
F7 0007
F7 0007 ************ JOB ARIS75FD ********************
F7 0007 * GRANT SCHEDULE AUTHORITY
F7 0007 *********************************************
F7 0007 JOB ARIS75FD IS OPTIONAL.
F7 0007 WOULD YOU LIKE TO RUN IT? {Y|N-Default}
F7-0007
7 y
F4 0001 1Q47I   F4 ARIS75FD 00305 FROM (HEHA) , TIME=11:47:09, TKN=000001B1
F4 0004 // JOB ARIS75FD     GRANT SCHEDULE FOR DFHSIT APPLID
F4 0004 * ************************************************************
F4 0004 * ARIS75PL: DB2 PRODUCTION LIBRARY DEFINITION
F4 0004 * ************************************************************
F4 0004 * ************************************************************
F4 0004 * ARIS75DB: DB2 STARTER DATABASE IDENTIFICATION
F4 0004 * ************************************************************
F4 0004 * ************************************************************
F4 0004 * ARISDBSD: EXECUTE THE DBS UTILITY IN SQL/DS SINGLE USER MODE
F4 0004 * ************************************************************
⋮
F7 0007
F7 0007 ************ JOB ARIS6ASD ********************
F7 0007 * SQL ASSEMBLER SAMPLE PROGRAM
F7 0007 *********************************************
F7 0007 JOB ARIS6ASD IS OPTIONAL.
F7 0007 WOULD YOU LIKE TO RUN IT? {Y|N-Default}
F7-0007
7 y
F7 0001 1Q47I   F7 ARIS6ASD 00318 FROM (HEHA) , TIME=12:33:17, TKN=000001B2
F7 0007 // JOB ARIS6ASD SQL ASSEMBLER SAMPLE PROGRAM
F7 0007 * STEP 1 - PREP
F7 0007 1T20I  SYS079 HAS BEEN ASSIGNED TO X'FED' (PERM)
F7 0007 1T20I  SYSPCH HAS BEEN ASSIGNED TO X'FED' (PERM)
F7 0007 * STEP 2 - ASSEMBLE
F7 0007 1T20I  SYSIPT HAS BEEN ASSIGNED TO X'FEC' (PERM)
F7 0007 * STEP 3 - LINK EDIT
F7 0007 * STEP 4 - EXECUTE THE SAMPLE PROGRAM
F7 0007 EOJ ARIS6ASD  MAX.RETURN CODE=0000
F7 0007 ************ JOB ARIS6CD ********************
F7 0007 * SQL C/370 SAMPLE PROGRAM
F7 0007 *********************************************
F7 0007 JOB ARIS6CD IS OPTIONAL.
F7 0007 WOULD YOU LIKE TO RUN IT? {Y|N-Default}
F7-0007
7 n
F7 0007
F7 0007 ************ JOB ARIS6FTD ********************
F7 0007 * SQL FORTRAN SAMPLE PROGRAM
F7 0007 *********************************************
F7 0007 ************ JOB ARIS6PLD ********************
F7 0007 * SQL PL/I SAMPLE PROGRAM
F7 0007 *********************************************
F7 0007 JOB ARIS6PLD IS OPTIONAL.
F7 0007 WOULD YOU LIKE TO RUN IT? {Y|N-Default}
F7-0007
7 n
```

```
F7 0007 INSTALLATION STEP HAS ENDED SUCCESSFULLY....
F7 0007 PLEASE READ THE PROGRAM DIRECTORY AND PERFORM NECESSARY
F7 0007 STEPS MANUALLY.
F7 0007 EOJ DB2JMGR   MAX.RETURN CODE=0000
```

ステップ 4: DRDA サポートのセットアップ

このステップと、ステップ 5 および 6 を合わせて、 Db2 ベース・コネクターで使用するように z/VSE 環境を準備します。

DRDA のセットアップに必要な定義を作成するには、スケルトン SKDB2VAR によって VSE/POWER 読み取り待ち行列に配置されたジョブ **DB2DRDA** をリリースします。

```
* $$ JOB JNM=DB2DRDA,CLASS=R,DISP=L,NTFY=YES
* $$ LST CLASS=Q,DISP=H
// JOB DB2DRDA ACTIVATE DRDA SERVER SUPPORT
* ********************************************************
*  LINK EDIT RDS WITH DRDA SERVER SUPPORT
* ********************************************************
// LIBDEF *,SEARCH=PRD2.DB2750
// LIBDEF PHASE,CATALOG=PRD2.DB2750
// OPTION CATAL
 INCLUDE ARISLKRA
// EXEC  PGM=LNKEDT,PARM='MSHP,AMODE=31,RMODE=ANY'
/*
/&
* $$ EOJ
```

ステップ 5: ストアード・プロシージャー・サーバーのセットアップと Db2 への定義

ステップ 5 は、以下のように分かれています。

- 『ステップ 5.1: ストアード・プロシージャー・サーバーをセットアップする』
- 107 ページの『ステップ 5.2: Db2 にストアード・プロシージャー・サーバーを定義する』

ステップ 5.1: ストアード・プロシージャー・サーバーをセットアップする

ストアード・プロシージャー・サーバー は、デフォルトでは区画サイズ 8MB のクラス R の動的区画で実行されます。 しかし、 (推奨最小値の 8MB) より大きなサイズの区画を持つ別の区画で実行するようにジョブを構成できます。

スケルトン **SKDB2SPS** (VSE/ICCF ライブラリー 59 にある) を使用して、ストアード・プロシージャー・サーバー のスタートアップ・ジョブ **SPSERV01** を PRD2.DB2750 にカタログします。以下に、このジョブを示します。

```
$$ JOB JNM=DB2SPSCA,CLASS=0,DISP=D,NTFY=YES
$$ LST CLASS=Q,DISP=H
// JOB DB2SPSCA CATALOG STORED PROCEDURE SERVER JOB
* ********************************************************************  C
* *                                                                  *  C
* *  THIS JOB WILL CATALOG THE JOB THAT WILL RUN IN THE              *  C
* *  STORED PROCEDURE SERVER PARTITION.                              *  C
* *  SERVER PARTITION SHOULD BE A DYNAMIC PARTITION OF AT LEAST      *  C
```

```
* *  8MB IN SIZE, DEFAULT IS CLASS R.                              *  C
* *  MODIFY JCL AS FOR YOUR NEEDS. CLASS IN THE POWER PUN CARD     *  C
* *  IS THE PARTITION WHERE THE SERVER RUNS.                       *  C
* *  THE STORED PROCEDURE LIBRARY PRD2.DB2STP SHOULD BE IN THE     *  C
* *  SEARCH CHAIN.                                                 *  C
* ********************************************************************  C
                                                                       C
         AFTER YOU HAVE MODIFIED THE SKELETON ENTER '§DTRSEXIT'        C
         FROM THE EDITOR'S COMMAND LINE.                               C
         THIS MACRO WILL DELETE ALL DESCRIPTIVE TEXT FROM THIS FILE,   C
         BY DELETING ALL LINES WHICH ARE MARKED WITH THE CHARACTER C   C
         IN COLUMN 71.                                                 C
                                                                       C
// EXEC  LIBR,PARM='MSHP'
   ACCESS  SUBLIB = PRD2.DB2750
CATALOG SPSERV01.A        EOD=&&          REPLACE=YES
. $$ PUN JNM=SPSERV01,DISP=I,CLASS=R
// JOB SPSERV01    START DB2 STORED PROCEDURE SERVER 01
// OPTION NODUMP,NOSYSDUMP
* // EXEC PROC=ARIS75SL
// ASSGN SYS098,SYSPCH
// LIBDEF *,SEARCH=(PRD2.DB2STP,PRD2.DB2750,PRD2.SCEEBASE,PRD1.BASE)
   ON $RC > 0 GOTO END
// EXEC  PGM=ARISPRC,SIZE=1M
/.END
/*
/&
&&
/*
/&
$$ EOJ
```

ストアード・プロシージャー・サーバー は、関連付けられているストアード・プロシージャー・サーバー をシステム・スタートアップ時にスタートする *Db2 Server for VSE* 専用です。

ステップ 5.2: Db2 にストアード・プロシージャー・サーバーを定義する

以下を使用してストアード・プロシージャー・サーバー SPSERV01 を定義できます。

- Db2 コマンド:

  ```
  CREATE PSERVER SPSERV01 AUTOSTART YES
  ```

- バッチ・ジョブ。インストール中のこの時点で実行できます。

  ```
  * $$ JOB JNM=PSERVER,CLASS=0,DISP=D
  * $$ LST CLASS=W,DISP=H,DEST=(,xxxxx)
  // JOB PSERVER
  // LIBDEF PROC,SEARCH=(PRD2.DB2750)
  // DLBL SQLGLOB,'DB2.SQLGLOB.MASTER',,VSAM,CAT=DB2UCAT,DISP=(OLD,KEEP)
  // EXEC PROC=ARIS75PL        *-- DB2 PROD. LIBRARY ID PROC
  // EXEC PROC=ARIS75DB        *-- DB2 DATABASE ID PROC
  // EXEC  ARIDBS,SIZE=AUTO
  CONNECT SQLDBA IDENTIFIED BY SQLDBAPW;
  CREATE PSERVER SPSERV01 AUTOSTART YES;
  /*
  /&
  * $$ EOJ
  ```

ステップ 6: ストアード・プロシージャーをセットアップする

ストアード・プロシージャーは、システム・ライブラリー PRD2.DB2STP にカタログする必要があります。

ストアード・プロシージャーは再入可能 パラメーターを使用してコンパイルする必要があります。 COBOL プログラムでのコンパイル方法の例については、スケルトン SKDLICMP (VSE/ICCF ライブラリー 59 にある) で使用される RENT オプションを参照してください。

ストアード・プロシージャーの定義を作成するには、『ステップ 7: VSAM データ・アクセス用に Db2 ベース・コネクターをカスタマイズする』および 『ステップ 8: DL/I データ・アクセス用に Db2 ベース・コネクター をカスタマイズする』の下にリストされているスケルトンを使用します。

ストアード・プロシージャーは、特定のストアード・プロシージャー・サーバー 専用です。

ステップ 7: VSAM データ・アクセス用に Db2 ベース・コネクターをカスタマイズする

Db2 ベース・コネクターを介して VSAM データにアクセスする場合は、下記のステップに従ってストアード・プロシージャーを作成および定義します。 これらのスケルトンは VSE/ICCF ライブラリー 59 で提供されます。

- ステップ 7.1: スケルトン SKCRESTP を使用して、VSAM データ・アクセス用のストアード・プロシージャーを作成し、Db2 に定義する。
- ステップ 7.2: スケルトン SKCPSTP を使用して、VSAM データ・アクセス用の C で記述したストアード・プロシージャーをコンパイルし、リンク・エディットする。
- ステップ 7.3: スケルトン SKVSSAMP を使用して、VSE/VSAM クラスターを定義し、そこにサンプル・データをロードする。

ステップ 8: DL/I データ・アクセス用に Db2 ベース・コネクター をカスタマイズする

Db2 ベース・コネクターを介して DL/I データにアクセスする場合は、まず z/VSE オプショナル・プログラム DL/I VSE をインストールする必要があります。 その後で、下記のステップを実行できます。 これらのスケルトンは VSE/ICCF ライブラリー 59 にあります。

- ステップ 8.1: スケルトン SKDLISMP を使用して、DL/I サンプル・データベースを定義し、ロードする。
- ステップ 8.2: スケルトン SKDLISTP を使用して、DL/I サンプル・データベースのアクセスに使用される Db2 ストアード・プロシージャーを作成する。
- ステップ 8.3: スケルトン SKDLICMP を使用して、DL/I サンプル・データベースのアクセスに使用される COBOL Db2 ストアード・プロシージャーをコンパイルし、リンク・エディットする。
- ステップ 8.4: CICS TS をカスタマイズする。 Db2 ベース・コネクターを介して DL/I データにアクセスするには、CICS TS - DL/I オンライン・システムをカスタマイズする必要があります。
  1. 以下の説明に従って、CICS/DLI オンライン・システムを構成します。
     – 「*DL/I Resource Definition and Utilities*」の『Part 6』。

- 「*DL/I Resource Definition and Utilities*」の『CICS - DL/I Tables - Requirements』のトピック。
- 「*DL/I Release Guide*」の『Migrating to DL/I VSE 1.11 and the CICS Transaction Server for z/VSE 1.1』トピック。

2. サンプル・データベース STDIDBP およびアクセスしたいその他のデータベースを CICS に (CICS FCT またはトランザクション CEDA のいずれかを使用して) 定義します。
3. サンプル・データベース STDIDBP (// DLBL STDIDBC ...) およびアクセスしたいその他のデータベースにラベルを指定します。
4. 使用するすべての DL/I オンライン・プログラムおよび PSB を組み込むことによって、新規 DL/I オンライン中核 (DLZACT 生成) を作成します。CICS/DLI ミラー・プログラム DLZBPC00 は、DL/I サンプル・データベースにアクセスするために PSB STBICLG に対して許可されている必要があります。 CICS/DLI ミラー・プログラム DLZBPC00 は、他の DL/I データベースにアクセスするその他の PSB に対しても許可しておく必要があります。
5. CICS/DLI オンライン・システム内の並行 DLZBPC00 ミラー・タスクの増分数を計算します。結果に応じて、DLZACT 生成時の MAXTASK および CMAXTSK パラメーターを調整する必要があります。
6. DL/I 出口ルーチン DLZBSEOT を SVA にロードします。
7. MPS システムを始動します。

DLZMPX00 は SVA 適格で、*AIBTDLI* インターフェース 経由の DL/I データへのアクセスに使用されます (DLZMPX00 および AIBTDLI インターフェースの説明については、 460 ページの『AIBTDLI インターフェースの概要』を参照してください)。 AIBTDLI インターフェースは、 DLZMPX00 がすでにあれば SVA から使用するか、DLZMPX00 を区画スペースにロードしてそこから使用します。

ステップ 9: Db2 の開始と、ストアード・プロシージャー・サーバーの始動

スケルトン **SKDB2STR** を実行して、スタートアップ・ジョブ **DB2START** を VSE/POWER 読み取り待ち行列に配置します。 以下に、このスケルトンを示します。

```
* $$ JOB JNM=DB2START,CLASS=S,DISP=L,NTFY=YES
* $$ LST CLASS=Q,DISP=H
// JOB DB2START DB2 SERVER STARTUP JOB
// LIBDEF PROC,SEARCH=(PRD2.DB2750,PRD2.DB2STP)
// LIBDEF PHASE,SEARCH=(PRD1.BASE,PRD2.SCEEBASE,PRD2.DB2750,          X
               PRD2.DB2STP)
// EXEC  PROC=ARIS75SL    *-- DB2 PRODUCTION LIBRARY ID PROC
// EXEC  PROC=ARIS75DB    *-- DB2 DATABASE ID PROC
// ASSGN SYS098,SYSPCH    *-- DB2 ENABLE POWER FOR STORED PROC HANDLER
// EXEC  ARISQLDS,SIZE=AUTO,PARM='TCPPORT=446,NCUSERS=05,DBNAME=SQLDS, X
               DSPLYDEV=B,RMTUSERS=10'
/*
/&
* $$ EOJ
```

DB2START をリリースすることで *Db2 Server for VSE* を始動します。 それにより、*Db2 Server for VSE* は PRD2.DB2750 から取り出したスタートアップ・ジョブ SPSERV01 を通じてストアード・プロシージャー・サーバー をスタートアップします。自動スタートアップ処理を行う場合は、SKJCL1 に DB2START を入れることができます (97 ページの『ステップ 1: CICS TS をカスタマイズする』も参照してください)。

DB2START では、TCP/IP が実行されている必要があります。 TCP/IP が実行されていることを確認するには、 DB2START で EXEC ARISQLDS ステートメントの前に以下のジョブ・ステップを挿入します。

```
// EXEC REXX=IESWAITR,PARM='TCPIP00'
 /*
```

ここで、*TCPIP00* は TCP/IP スタートアップ・ジョブの名前です。

さらにスタートアップするには、以下に示すように、CICS TS スタートアップ・ジョブに CIRB トランザクションを挿入することでサンプル・データベースをオープンすることもできます。

```
// EXEC DFHSIP,SIZE=DFHSIP,PARM='SIT=C3,START=COLD,SEC=NO,STATRCD=OFF,S*
               VA=NO,NEWSIT=YES,DSALIM=8M,EDSALIM=30M,SI',DSPACE=2M,OS3*
               90
/*
CIRB PASSWORD,8,XXX03,1,GER,PRODDBI
/*
```

ステップ 10: Db2 Connect のインストールとクライアント/ホスト接続の確立

クライアントから z/VSE ホストへの接続を最終的に確立するには、3 層環境の物理/論理中間層 (10 ページの『3 層環境の概要』で説明) に Db2 Connect バージョン 6 リリース 1 以降をインストールする必要があります (まだインストールされていない場合)。次に、Db2 Connect を構成 して、z/VSE ホストに保管された Db2 データにアクセスできるようにしなければなりません。これを行うには、クライアント構成アシスタント (CCA)、または以下に記載する Db2 コマンド行インターフェースのいずれかを使用することができます。

サンプル・データベース **sqlds** は、z/VSE ホストに保管され、Db2 ベース・コネクターと合わせて使用されます。 **sqlds** データベースを別名 **db2vsewm** で物理/論理中間層の Db2 Connect に定義するには、以下の作業を行う必要があります。

1. 物理/論理中間層の Db2 Connect と、Db2 Server for VSE にあるデータベース (**sqlds**) の間の通信プロトコルを定義します。 これを行うには、以下のように *node* を定義します。

   ```
   db2 catalog protocol node nodename remote ip-addr server port-nr
   ```

   Db2 ベース・コネクターのサンプル (DL/I アプレットおよび VSAM アプレット) では、次のようなコマンドを入力します。

   ```
   db2 catalog tcpip node tcpvse remote 9.111.122.33 server 446
   ```

2. データベース接続サービス (DCS) の項目を定義します。 これを行うには、次のコマンドを使用します。

   ```
   db2 catalog dcs database dcs-name as vse-dbname
   ```

Db2 ベース・コネクターのサンプル (DL/I アプレットおよび VSAM アプレット) では、次のようなコマンドを入力します。

```
db2 catalog dcs database dcsdb as sqlds
```

3. Db2 ベース・コネクターのサンプルが使用する別名 を定義します。これを行うには、次のコマンドを使用します。

   ```
   db2 catalog dcs-name  as vse-alias-dbname at node nodename authentication dcs
   ```

   Db2 ベース・コネクターのサンプル (DL/I アプレットおよび VSAM アプレット) では、次のようなコマンドを入力します。

   ```
   db2 catalog dcsdb as db2vsewm at node tcpvse authentication dcs
   ```

Db2 Connect へのデータベース **sqlds** の定義が完了したら、次の *bind* ステップを実行する必要があります。

```
db2 bind path@ddcsvse.lst blocking all sqlerror continue messages msg-file
grant public
```

例えば、Windows では次のように入力します。

```
db2 bind db2¥bnd¥@ddcsvse.lst blocking all sqlerror continue messages log.msg
grant public
```

Db2 Connect のインストールおよびカスタマイズ方法について詳しくは、以下の箇所を参照してください。

- *Db2 Connect* 使用者の手引き (SD88-7259)。
- Db2 ホーム・ページ: `http://www.ibm.com/db2` (英語のみ)。ここには、Db2 に関するほとんどの資料へのリンクが記載されています。「ライブラリー **(Library)**」を選択して「**Db2** 資料 **(Db2 Publications)**」を選択することで、Db2 Server for VSE および Db2 Connect に関するほとんどの資料を (PDF、ポストスクリプト、または HTML 形式 (あるいはすべての形式) で) 参照できます。

第 11 章 VSAM-Via-CICS サービスの構成

VSE/ESA 2.6 まで、VSAM クラスターが CICS による更新用にオープンされた場合、VSE コネクター・サーバー および Db2 ストアード・プロシージャーは、VSAM 共用オプション 4 が使用されていない限り 同じ VSAM クラスターにのみ読み取り専用モードでアクセスできました。VSAM 共用オプション 4 を使用すると、パフォーマンスの低下を招く結果になりました。

VSE/ESA 2.6 以降では、*VSAM-via-CICS* サービス を使用することで、VSE コネクター・サーバー および Db2 ストアード・プロシージャーは、VSAM 共用オプション 4 を使用するという制約なし で VSAM データにアクセスできます。

VSAM クラスターは CICS によって一度だけ オープンされなければなりません。共用問題がなくなったため、VSAM 共用オプション 4 を使用する必要はありません。

このトピックでは、このパフォーマンス改善方法について説明します。

以下の項目があります。

- 『IBM 提供の CICS システムの構成』
- 115 ページの『VSAM-Via-CICS 用の CICS システムの詳細な構成』
- 115 ページの『VSAM-Via-CICS サービスの動作方法』
- 116 ページの『VSAM-Via-CICS で使用される CICS トランザクション』

IBM 提供の CICS システムの構成

CICS を介した VSAM アクセスには以下の CICS プログラムが含まれます。

- IESCVSRV (サーバー・タスク)
- IESCVMIR (ミラー・タスク)
- IESCVSTA (開始トランザクション)
- IESCVSTI (内部開始トランザクション)
- IESCVSTP (停止トランザクション)

VSAM-via-CICS サービスでサポートされるファイル・タイプは、以下のとおりです。

- ESDS
- KSDS
- RRDS
- KSDS-PATH
- ESDS-PATH

VSAM-via-CICS サービスを使用するには、以下の作業を行う必要があります。

1. CICS にアクセスする VSAM クラスターを定義します。 このためには、`CEDA DEFINE` トランザクションを使用します。 以下のように VSAM クラスターが (`CEDA DEFINE` を使用して) 定義されていることを確認します。
   - すべての VSAM クラスターが使用可能である。

- 読み取り専用アクセスの場合、VSAM クラスターが readable および browsable である。
- 書き込みアクセスの場合、VSAM クラスターが addable、updateable、および deleteable である。

2. 以下の項目を開始する前に、モジュール IESCVSVA.PHASE が SVA にロードされていることを確認します。
   - VSAM-via-CICS サービス
   - CICS
3. VSAM-via-CICS サービスがアクティブであることを確認します。 IBM 提供の CICS システムは、このサービスを自動的に 開始するように構成されています。このサービスをお客様自身で 開始する場合は、トランザクション ICVA を使用して行ってください。 VSAM-via-CICS サービスを停止するには、トランザクション ICVP を使用します。
4. CICS を介して VSAM クラスターにアクセスするアプリケーション (VSE コネクター・クライアント アプリケーション、または Db2 ストアード・プロシージャーを呼び出すアプリケーション) が以下の命名規則を使用していることを確認します。
   - カタログ・ファイル ID は次のように指定する必要があります。

     ```
     #VSAM.#CICS.CICS applid
     ```

     例えば、#VSAM.#CICS.DBDCCICS
   - クラスター・ファイル ID は、CICS に使用したものと同じにする必要があります (7 文字で構成)。

   これらの命名規則の使用例を示します。 MY.TEST.CLUSTER という名前の VSAM クラスターがあり、このクラスターが VSAM カタログ MY.USER.CATALOG 内にあると想定します。 このファイルは、CICS システムに MYTEST として定義済みで、DBDCCICS が適用されています。

   ファイル MYTEST に VSE コネクター・クライアント アプリケーション、または Db2 ストアード・プロシージャーを呼び出すアプリケーションのいずれかからアクセスするには、以下のマッピング名を使用します。

   ```
   Catalog File ID: #VSAM.#CICS.DBDCCICS
   Cluster File ID: MYTEST
   ```

ここで、既存のプログラム (VSE コネクター・クライアント アプリケーション、または Db2 ストアード・プロシージャーを呼び出すアプリケーション) に加えなければならない変更は、これらのプログラムで上記の VSAM マッピングの命名規則を使用することです。

パフォーマンスを最適化するために、以下の処理が可能です。

- *read* コマンドでオリジナルの VSAM 名を使用してバッチで VSAM クラスターにアクセスする。
- *write* コマンドで VSAM-via-CICS サービスを使用して VSAM-via-CICS サービス名により VSAM クラスターにアクセスする。

VSAM-Via-CICS 用の CICS システムの詳細な構成

VSE/ESA 2.6 以降に出荷される各 CICS システムは、デフォルトで VSAM-via-CICS サービスをアクティブにするように構成されています。 したがって、VSE/ESA 2.6 以降に出荷されるその他の CICS システムに対してカスタマイズ・アクティビティーを実行する必要はありません。

ただし、*VSE/ESA 2.6* よりも前に出荷済みの CICS システムで VSAM-via-CICS サービスを使用するように構成する場合には、以下の作業を行う必要があります。

1. 以下のプログラムがまだ定義されていない場合には、IBM 提供の CICS に定義した同じ方法でご使用の CICS に定義します。
   - IESCVSRV (サーバー・タスク)
   - IESCVMIR (ミラー・タスク)
   - IESCVSTA (開始トランザクション)
   - IESCVSTI (内部開始トランザクション)
   - IESCVSTP (停止トランザクション)
2. 以下のトランザクションがまだ定義されていない場合には、IBM 提供の CICS に定義した同じ方法でご使用の CICS に定義します。
   - ICVS
   - ICVM
   - ICVA
   - ICVP

   (これらのトランザクションについて詳しくは、 116 ページの『VSAM-Via-CICS で使用される CICS トランザクション』を参照してください。)
3. VSAM-via-CICS サービスを自動的に 開始する場合は、以下の作業を行う必要があります。
   a. 次のステートメントを CICS PLT テーブル DFHPLTPI に追加します (テーブルの最後 のステートメントとして)。

      ```
      DFHPLT TYPE=ENTRY, PROGRAM=IESCVSTI
      ```

   b. 次のステートメントを CICS PLT テーブル DFHPLTSD に追加します (テーブルの最初 のステートメントとして)。

      ```
      DFHPLT TYPE=ENTRY, PROGRAM=IESCVSTP
      ```
4. ロード・リスト $SVACONN を使用して、IESCVSVA.PHASE が SVA にロードされたことを確認します (CICS を介した VSAM アクセスのバッチ・サイド・サポートは $IESCVBA.PHASE にあります)。

VSAM-Via-CICS サービスの動作方法

VSAM-via-CICS サービスの動作方法を以下に示します。

1. VSE コネクター・クライアント アプリケーションまたは Db2 ストアード・プロシージャーを呼び出すアプリケーションが、VSAM アクセス要求を出します。
2. 要求は、VSE コネクター・サーバー または Db2 ストアード・プロシージャーが実行されているバッチ区画に送信されます。
3. 要求は XPCC (区画間通信) を介して CICS に転送されます。

4. CICS Transaction Server 内で実行中の VSAM-via-CICS サービスは、要求を実行し (例えば、レコードの読み取り)、VSE コネクター・サーバー または Db2 ストアード・プロシージャーが実行されているバッチ区画にデータを渡します。
5. VSE コネクター・サーバー または Db2 ストアード・プロシージャーはアプリケーションにデータを戻します。

VSAM-Via-CICS で使用される CICS トランザクション

VSAM-via-CICS サービスで使用するように以下の CICS トランザクションが用意されています (VSE/ESA 2.6 システムに事前定義されています)。

ICVS
サーバー・トランザクション。 プログラム IESCVSRV を使用します。

ICVM
ミラー・トランザクション。 プログラム IESCVMIR を使用します。

ICVA このトランザクションを使用して VSAM-via-CICS サービスを開始できます。 プログラム IESCVSTA を使用します。

ICVP このトランザクションを使用して VSAM-via-CICS サービスを停止できます。 プログラム IESCVSTP を使用します。

第 12 章 リレーショナル構造への VSE/VSAM データのマッピング

このトピックでは、VSE/VSAM データをリレーショナル構造にマップする方法、およびそれに使用できる 4 つの方法について説明します。

以下の項目があります。

VSE/VSAM データのマッピングの概要

以下のものを使用する VSAM データにアクセスしたい場合は、VSAM レコードをリレーショナル構造にマップする必要があります。

- 451 ページの『Db2 ストアード・プロシージャーによる VSAM データへのアクセス』で説明する VSAM CLI (コール・レベル・インターフェース) を介した Db2 ストアード・プロシージャー。
- 159 ページの『VSE Java Beans クラス・ライブラリーの内容』で説明する VSE Java Beans。

これまでは、VSAM データは主に、VSAM データの内部構造を認識しているアプリケーション・プログラムを使用してアクセスされていました。レコード・レイアウトは、アプリケーション・プログラム内のデータ構造によって表されていました。 この方法の欠点としては、以下のものがあります。

- 指定のレコード・レイアウトを他のアプリケーションと共用する方法がない。
- レコード・レイアウトが変更された場合、アプリケーション・プログラムも変更しなければならない。
- データ構造がプログラム言語 (例えば、 COBOL、アセンブラー、または PL/I) に依存する。
- アプリケーション・プログラムを実行するオペレーティング・システム・プラットフォームにフォーマット済みデータ・レポートを作成しなければならない。

ただし、本書に記載する *e-business* アプリケーション を開発するには、以下の条件を満たす必要があります。

- オペレーティング・システム・プラットフォーム間のデータ表記の共用。
- 異なるアプリケーションからのデータ表記への容易なアクセス。
- データ表記およびデータ表示をオペレーティング・システム・プラットフォームおよびプログラム言語から独立させる。

これにより、z/VSE は、e-business アプリケーション内で使用できるように VSAM データをマッピングする以下の 2 つの方法を提供します。

- 指定の VSAM レコードまたはレコード・タイプにデータ・マップ (単にマップと呼ぶ) を作成する。 マップによって、VSAM レコードを列とそのデータ・フィールドに分割し、名前、長さ、データ・タイプ、およびレコード内のオフセットを付けます。
- マップ内に入れるフィールドのサブセットを含むデータ・ビュー (単にビュー と呼ぶ) を作成する。 ビューは常に、指定の VSAM マップのデータ・フィールドのサブセットを指します。 そのため、VSAM マップを変更すると、そのマップを使用するビューも影響を受けます。 例えば、マップを削除すると、そのマップのすべてのビューも削除されます。 ビューを使用して、異なるユーザー・グループに同じデータの別個のビューを提供できます (例えば、一部の情報を特定のユーザーから隠す場合)。

VSAM マップの構造化方法

マップピング定義は、以下の階層 構造になっています。

```
CATALOG       (MY.USER.CATALOG)
 CLUSTER      (MY.DATA.CLUSTER)
  MAP         (Map One)
   COLUMN     (Name,      Ofs=0,Len=10,Type=String,Description=xxxxx)
   COLUMN     (Street,    Ofs=10,Len=20,Type=String,Description=xxxxx)
   ...
   VIEW       (View One)
     COLUMN   (PersonInfo, Ref=Name,Description=xxxxx)
     COLUMN   (Address,    Ref=Street,Description=xxxxx)
     ...
   VIEW       (View Two)
   ...
  MAP         (Map Two)
  ...
```

図 *20. VSAM* マップの階層構造

1 つのデータ・フィールドは、VSAM レコードの特定の 1 つの列を表します。 これは常に、指定のマップまたはビューのパーツになります。 VSAM レコードは、以下により構成されます。

- 名前 (列名)
- 長さ
- データ・タイプ
- レコード内のオフセット
- 説明

アプリケーションがマップを使用して VSAM データにアクセスするときには、マップ のフィールドのみが使用されます。

マップを使用して VSAM データにアクセスする際にフィルターを指定することもできます。 例えば、指定の列 (データ・フィールド) が指定のフィルター文字列と一致するレコードのみを表示できます。 任意の Java プログラム (Java アプリケーション、Java サーブレット、Java アプレット、または Enterprise Java Beans) でフィルターを使用できます。

注: クラスターにマップを定義したときに、クラスターが存在するかどうかの検査は行われません。 そのため、マッピング定義が常に VSAM クラスターと同期していることを確認するステップを入れる必要があります。

z/VSE ホストにおけるマップの保管方法

ホストでは、すべての VSAM クラスターのマップは 1 つの VSAM ファイル VSE.VSAM.RECORD.MAPPING.DEFS に保管されます。 このファイルは、*z/VSE* の初期インストール時に自動的に作成されます。

この VSAM ファイルのショート・ネームは IESMAPD で、このファイルのクラスターは VSESP.USER.CATALOG に保管されます。

図 21 は、ファイル VSE.VSAM.RECORD.MAPPING.DEFS へのクラスターの定義に使用されるジョブの例です。 このサンプル・ジョブは、ライブラリー IJSYSRS.SYSLIB のメンバー VSAMDEFS.Z にあります。

注:

1. 通常はこのジョブを実行する必要はありません。z/VSE のインストール時に自動的に実行されます。
2. このファイルを削除および移動しないでください。

```
* $$ JOB JNM=DEFINE,CLASS=0,DISP=D
// JOB  DEFINE FILE
// EXEC IDCAMS,SIZE=AUTO
          DEFINE CLUSTER (NAME (VSE.VSAM.RECORD.MAPPING.DEFS) -
          RECORDS(2000,1000) -
          SHAREOPTIONS (2) -
          RECORDSIZE (284    284   ) -
          VOLUMES (DOSRES SYSWK1 ) -
          NOREUSE -
          INDEXED -
          FREESPACE (15 7) -
          KEYS (222 0     ) -
          NOCOMPRESSED -
          TO (99366  )) -
          DATA (NAME (VSE.VSAM.RECORD.MAPPING.DEFS.@D@) -
          CONTROLINTERVALSIZE (4096  )) -
          INDEX (NAME (VSE.VSAM.RECORD.MAPPING.DEFS.@I@)) -
          CATALOG (VSESP.USER.CATALOG)
IF LASTCC NE 0 THEN CANCEL JOB
/*
/&
* $$ EOJ
```

図 *21. VSE.VSAM.RECORD.MAPPING.DEFS* にクラスターを定義するジョブ

RECMAP を使用したマップの定義

VSAM クラスターにマップを定義するこの方法では、IDCAMS **RECMAP** コマンドを使用します。 RECMAP を使用して、マップまたはビューの内容を削除、変更、またはリストすることもできます。

関連トピック:

- RECMAP コマンドの構文は、「*IBM z/VSE VSE* 中央機能 *VSE/VSAM* コマンド」(SC43-2946) に説明されています。
- RECMAP を使用する方法の実際的な例の説明は 226 ページの『4. VSAM データ・クラスターの定義』にあります。

サンプル・アプレットを使用したマップの定義

VSAM クラスターにマップを定義するこの方法では、アプレット を使用して*Web* ブラウザー で実行します。

マップの定義に使用されるサンプル・アプレットは、z/VSE e-business コネクターのオンライン・ドキュメンテーションの **VsamMappingApplet.html** にあり、212 ページの『データ・マッピング・アプレットのサンプルの実行』で説明しています。

Java アプリケーションを使用したマップの定義

VSAM クラスターにマップを定義するこの方法では、*Java* アプリケーション を使用してワークステーション で実行します。

ここに記載する例は、指定の VSAM クラスターにマップを作成し、2 つのビューも定義する単純な Java アプリケーション用です。 この例は、*z/VSE e-business* コネクター のオンライン・ドキュメンテーションから引用しています。

この例のオンライン版が必要な場合には、*VsamDataMapping.html* を参照してください。 ソース・コードは、コネクター・クライアント・インストール (28 ページの『VSE コネクター・クライアントのインストール』で説明) のサンプル・ディレクトリーの下にある **com.ibm.vse.samples** ディレクトリー内の Java ソース・ファイル *VsamDataMapping.java* に入っています。

以下のステップでは、VSAM クラスターにマップを定義する Java アプリケーションをコーディングする方法について説明します。

1. *VSAM* クラスターにマップを作成する

   マップを作成するには、まず *VSEVsamMap* クラスのローカル・オブジェクトのインスタンスを生成する必要があります。 ホスト上にマップ定義を作成するには、*create()* メソッドへの呼び出しも必要です。

   既存のマップを使用するには、クラス *VSEVsamMap* のローカル・オブジェクトのインスタンスを生成してから、*isExistent()* メソッドを使用してこのマップがホスト上に存在するかどうかを調べる必要があります。

   注**:** 分散オブジェクトは使用中であることに注意してください。 そのため、クラス *VSEVsamMap* および *VSEVsamView* のローカル・オブジェクトが作成されても、それは関連する定義がホスト上に作成されることを意味するものではありません。

   ローカル・マップ・オブジェクトを作成して、ホスト上に存在するかどうかを調べます。 存在しない場合には、*create()* メソッドを使用してホスト上に作成してください。 *create()* および *isExistent()* メソッドは例外エラーを作成できるた

め、try/catch ブロックにコードを配置します。 次に、ローカル VSAM マップ・オブジェクトの作成例を示します。

```
...
public static void main(String argv[ ]) throws IOException
{
   VSESystem system;
   VSEVsamMap map;
   VSEVsamView employeeView=null, managerView=null;
   VSEVsamField dataField;
   ...
   // Create VSESystem and connect
   ...

   // Create map ...
   String catalogID = "VSESP.USER.CATALOG";
   String clusterID = "VSAM.DISPLAY.DEMO.CLUSTER";
   try
   {
     System.out.println("Creating map MYMAP ...");
     map = new VSEVsamMap(system, catalogID, clusterID, "MYMAP");
     if (!(map.isExistent()))
     {
        System.out.println("  Map does not exist on host. Create it ...");
        map.create();
     }
     else
        System.out.println("  Map already exists on host. Continue ...");
   }
   catch (Exception e)
   {
      System.out.println("  Exception when creating map: ");
      System.out.println(e);
      return;
   }
   ... (Continued)
```

図 *22.* ローカル *VSAM* マップ・オブジェクトの作成例

2. マップにデータ・フィールドを作成する

   マップの指定のビューのデータ・フィールドは、単にマップのフィールドを参照しています。 しかし、同じフィールドが別のビューで別の名前を持つことができます。 例えば、マップのフィールドを削除すると、すべてのビューからそのフィールドを使用できなくなります。マップ全体を削除すると、そのマップのすべてのビューも削除されます。 しかし、ローカル・ビュー・オブジェクトにはマップが削除されたことは通知されません。 ビューに属するフィールドへの次回アクセス時には、例外が生成されます。

   そのため、レコードの内部構造を記述するフィールドをマップに追加する必要があります。 これらのフィールド名を大文字 にしなければならないことに注意してください (そうしなかった場合、ホストへ要求を送信する時に大文字に変換されます)。 *VSEVsamField* オブジェクトの作成はローカル専用操作であるのに対して、*addField()* メソッドはこの要求をホストに送信します。 次に、マップのデータ・フィールドの作成例を示します。

```
System.out.println("   ");
System.out.println("Adding fields to the map...");
String[ ] fields = {"LAST NAME", "FIRST NAME", "DEPARTMENT", "AGE"};
int[ ] types = {VSEVsamMap.TYPE_STRING, VSEVsamMap.TYPE_STRING,
                VSEVsamMap.TYPE_UNSIGNED, VSEVsamMap.TYPE_UNSIGNED};
int[ ] lengths = {20, 10, 4, 2};
int[ ] offsets = {0, 20, 30, 34};

for (int i=0;i<fields.length;i++)
{
   try
   {
      dataField = new VSEVsamField(system, fields[i], types[i],
                                   lengths[i], offsets[i]);
      map.addField(dataField);
   }
   catch (Exception e)
   {
      System.out.println("Exception when adding " + fields[i] + "
                                      to the map: ");
   }
}
... (Continued)
```

図 *23.* マップのデータ・フィールドの作成例

3. マップのプロパティーを表示する

   次に、作成したマップのプロパティーの一部を表示する例を示します。

```
...
/* Now display some properties of the map */
System.out.println("Map properties");
System.out.println("  Map name         : " + map.getName());
System.out.println("  Catalog name     : " + map.getCatalog());
System.out.println("  Cluster name     : " + map.getCluster());
System.out.println("  System name      : " + map.getVSESystem());
System.out.println("  Number of fields : " + new Integer(
                                  (map.getNoOfFields())).toString());
for (int i=0;i<map.getNoOfFields();i++)
{
   System.out.println("  Field " + new Integer(i).toString() + " : " +
                                                 map.getFieldName(i));
   System.out.println("   type   = " + new Integer(
                                    (map.getFieldType(i))).toString());
   System.out.println("   length = " + new Integer(
                                   (map.getFieldLength(i))).toString());
   System.out.println("   offset = " + new Integer(
                                   (map.getFieldOffset(i))).toString());
}
... (Continued)
```

図 *24.* マップのプロパティーの表示例

4. マップに従業員ビューを作成する

   z/VSE e-business コネクター のオンライン・ドキュメンテーションに記載する例では、管理者のビュー (ここには記載しません)、および従業員のビューの 2 つのビューをこのマップ内に作成します。 また、ビュー・オブジェクトの作成はローカル専用ですが、*create()* メソッドはホスト上にビューを作成します。

```
...
System.out.println("   ");
System.out.println("Creating employee's view on the map ...");
try
{
   employeeView = new VSEVsamView(system, catalogID, clusterID,
                                  "MYMAP", "EMPLOYEEVIEW");
   if (!(employeeView.isExistent()))
   {
      System.out.println("Employee view does not exist. Create it ...");
      employeeView.create();
   }
   else
      System.out.println("Employee already exists. Continue ...");
}
catch (Exception e)
{
   System.out.println("Exception when creating employeeView: ");
   return;
}
...
...
```

図 *25.* マップのビューの作成例

5. 従業員ビューにデータ・フィールドを追加する

   この例で、次に、従業員ビューにフィールドを追加します。 これには、マップへのフィールドの追加と同じメソッドを使用します。 さらに、VSAM レコードの 3 つのフィールド (姓、名、および部門番号) のみを従業員が参照できるとします。 次に、このようなシナリオのコードを示します。

```
String[ ] empFields = {"LAST NAME", "FIRST NAME", "DEPARTMENT"};
System.out.println("Adding fields to employeeView ...");
for (int i=0;i<emp.Fields.length;i++)
{
     try
     {
        employeeView.addField(empFields[i]);
     }
     catch (Exception e)
     {
        System.out.println("Exception when adding " + empFields[i] + "
                                          to employeeView: ");
     }
}
  ... (Continued)
```

図 *26.* ビューへのデータ・フィールドの追加例

6. 従業員ビューのプロパティーを表示する

   次に、従業員ビューのプロパティーを表示する例を示します。

```
/* Now display some properties of the employeeView */
   System.out.println("employeeView properties");
   System.out.println("  View name        : " + employeeView.getName());
   System.out.println("  Catalog name     : " + employeeView.getCatalog());
   System.out.println("  Cluster name     : " + employeeView.getCluster());
   System.out.println("  System name      : " + employeeView.getVSESystem());
   System.out.println("  Number of fields : " + new Integer(
                                        (employeeView.getNoOfFields())).toString());
   for (int i=0;i<employeeView.getNoOfFields();i++)
   {
   System.out.println("  Field " + new Integer(i).toString() + " : " +
                                                    employeeView.getFieldName(i));
      System.out.println("   type   = " + new
                                      Integer((employeeView.getFieldType(i))).toString());
      System.out.println("   length = " + new
                                     Integer((employeeView.getFieldLength(i))).toString());
      System.out.println("   offset = " + new
                                     Integer((employeeView.getFieldOffset(i))).toString());
   }
   ... (Continued)
```

図 *27.* ビューのプロパティーの表示例

7. マップを削除する

   ここでは、マップと、そのビューおよびフィールドを一緒に削除しています。

```
 /* Now delete the map including all views and fields */
    try
    {
       System.out.println("Deleting map MYMAP ...");
       map.delete();
    }
    catch (Exception e)
    {
       System.out.println("Exception when deleting map:");
       System.out.println(e);
    }
    ...
 }
```

図 *28.* マップの削除方法の例

サンプルによって作成される **Java** コンソール出力

IBM テスト・システムでこのサンプルを Web ブラウザーから (ファイル *RunExamples.html* を使用して) 実行すると、以下の出力が Java コンソールに示されます。

```
C:¥vsecon¥samples>java  com.ibm.vse.samples.VsamDataMapping
Please enter your VSE IP address:
9.164.155.95
Please enter your VSE user ID:
sysa
Please enter password:
*****  (password will display in clear)
Creating map MYMAP ...
  Map does not exist on host. Create it ...

Adding fields to the map...
Map properties
  Map name         : MYMAP
  Catalog name     : VSESP.USER.CATALOG
  Cluster name     : VSAM.DISPLAY.DEMO.CLUSTER
  System name      : 9.164.155.95
  Number of fields : 4
  Field 0 : LAST NAME
```

```
   type   = 2
   length = 20
   offset = 0
  Field 1 : FIRST NAME
   type   = 2
   length = 10
   offset = 20
  Field 2 : DEPARTMENT
   type   = 3
   length = 4
   offset = 30
  Field 3 : AGE
   type   = 3
   length = 2
   offset = 34

Creating employee's view on the map ...
Employee view does not exist. Create it ...
Adding fields to employeeView ...
employeeView properties
  View name        : EMPLOYEEVIEW
  Catalog name     : VSESP.USER.CATALOG
  Cluster name     : VSAM.DISPLAY.DEMO.CLUSTER
  System name      : 9.164.155.95
  Number of fields : 3
  Field 0 : LAST NAME
   type   = 2
   length = 20
   offset = 0
  Field 1 : FIRST NAME
   type   = 2
   length = 10
   offset = 20
  Field 2 : DEPARTMENT
   type   = 3
   length = 4
   offset = 30

Creating manager's view on the map ...
Manager view does not exist. Create it ...
Adding fields to managerView ...
Exception when adding MONTHLY SALARY to managerView:
managerView properties
  View name        : MANAGERVIEW
  Catalog name     : VSESP.USER.CATALOG
  Cluster name     : VSAM.DISPLAY.DEMO.CLUSTER
  System name      : 9.164.155.95
  Number of fields : 4
  Field 0 : LAST NAME
   type   = 2
   length = 20
   offset = 0
  Field 1 : FIRST NAME
   type   = 2
   length = 10
   offset = 20
  Field 2 : DEPARTMENT
   type   = 3
   length = 4
   offset = 30
  Field 3 : AGE
   type   = 3
   length = 2
   offset = 34
Deleting map MYMAP ...
Finished.

C:¥vsecon¥samples>pause
Press any key to continue . . .
```

VSAM クラスター (VSAM レコードの内部構造を記述) に 1 つ以上のマップを定義したら、以下のものを使用してマップ済みの VSAM データを表示できます。

- VSAM CLI インターフェースを介した Db2 ストアード・プロシージャー (451 ページの『Db2 ストアード・プロシージャーによる VSAM データへのアクセス』で説明します)。
- VSE Java Beans (175 ページの『VSE Java Beans を使用した VSAM データへのアクセスの例』で説明します)。

VSAM MapTool を使用したマップの定義

VSAM クラスターにマップを定義するこの方法では、*VSAM MapTool* を使用して COBOL サンプル集を構文解析することでマップを作成します。

さらに、VSAM MapTool は、以下の作業にも使用できます。

- 指定の z/VSE システムから指定のマップをインポートする (受け取る)。
- z/VSE システムへマップをエクスポートする (すなわち、VSE に送信する)。
- XML ファイルからマップをインポートする。
- XML ファイルへマップをエクスポートする。
- 指定のマップから Java ソース・ファイルを作成する。 Java プログラムは、このマップを使用して、関連する VSAM ファイルからすべてのレコードを取得できます。

図 29 に、VSAM MapTool が表示するウィンドウを示します。

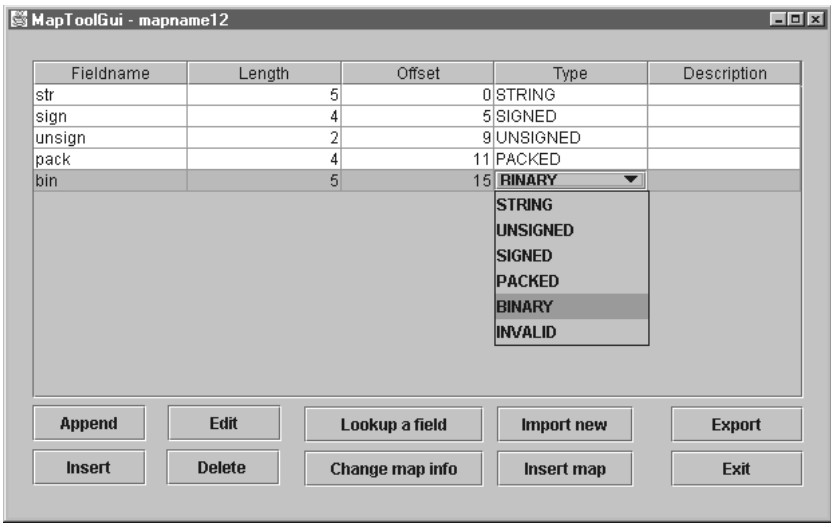


図 *29. VSAM MapTool* ウィンドウの例

VSAM MapTool をインストールするには、以下の作業を行う必要があります。

1. 次のインターネット・アドレスからファイル **maptool.zip** をダウンロードします。

   ```
   http://www.ibm.com/systems/z/os/zvse/downloads/
   ```

2. 上記のアドレスの Web サイトに記載されたインストール手順を実行します。

インストール・パッケージには、VSAM MapTool の使用方法の多くの例が入っています。

第 3 部 システム・モニター

第 13 章 VSE モニター・エージェントを使用してデータを収集

このトピックでは、*VSE Monitoring Agent* を使用してデータを z/VSE システムから収集する方法を説明します。

以下の項目があります。

- 『VSE データ収集方法の概要』
- 134 ページの『VSE Monitoring Agent の構成』
- 135 ページの『SNMP 通信のセキュリティー最大化』
- 136 ページの『IBM 提供のモニター・プラグインの構成』
- 137 ページの『VSE Monitoring Agent で使用可能なコマンド』
- 138 ページの『データ収集方法の例』
- 139 ページの『バッチ・ジョブでの SNMP トラップ・クライアントの使用』
- 141 ページの『LE/C プログラムでのトラップ・クライアント API の使用』
- 142 ページの『トラップ LE/C インターフェースで提供される関数』
- 142 ページの『COBOL プログラムおよび PL/I プログラムでのトラップ・クライアント API の使用』
- 143 ページの『CICS プログラムでの トラップ・クライアント API の使用』
- 144 ページの『トラップ COBOL および PL/I バッチ・インターフェースに使用されるコピーブック』
- 145 ページの『トラップ・クライアント API で生成される戻りコード』
- 146 ページの『VSE Monitoring Agent 用の独自プラグインの作成』

注**:** このトピックでは、各 SNMP メンバーを、「SNMP」という語を含まない 簡易書式で表しています。例えば、SNMP トラップ・クライアント API は単に トラップ・クライアント *API* と表記されます。また、SNMP トラップ CICS インターフェースは トラップ *CICS* インターフェース と表記されます。ただし、簡易書式のない *SNMP* トラップ・クライアント は例外です。

VSE データ収集方法の概要

IBM が開発したアプリケーション・フレームワーク によって、お客様は、標準モニター・インターフェース を使用して z/VSE システムをモニターできるようになります。このアプリケーション・フレームワークは、一般的な規格 TCP/IP と SNMP をベースにしています。ここにはオープン・インターフェース が組み込まれているため、独自のプログラムを使用してその他のデータを収集できます。

通常、z/VSE ホストで稼働する主要アプリケーション (CICS、COBOL、VSAM、など) は、企業の業務にとって重要なため、z/VSE システムをモニターするのは重要です。加えて、これらの主要アプリケーションは、過去のリソースという膨大な投資を表すのが常です。

アプリケーション・フレームワークは、以下の *3* つの論理層で構成されます。

- クライアント。ワークステーション、サーバー、携帯電話、または携帯情報端末 (PDA) である可能性があり、ここにモニター・ソフトウェアがインストールされます。
- *VSE Monitoring Agent*。アプリケーション・フレームワークの中枢にあり、以下を行います。
  - クライアントからの要求を処理
  - ビジネス・ロジックおよびデータへのアクセスを制御
- プラグイン。以下へのアクセスを提供します。
  - z/VSE データ
  - その他データ (独自プログラム使用による)

133 ページの図 30 に示してあるように、z/VSE には以下が備わっています。

- VSE Monitoring Agent。これは、SNMP バージョン 1 のすべての要求に応答します。
- SNMP トラップ・クライアント。これは、SNMP バージョン 1 「トラップ」をバッチ・ジョブから SNMP モニターに送信することによってモニター目的で使用できます。
- トラップ・クライアント API (IESMTRPA.PHASE)。これは、SNMP バージョン 1 「トラップ」を COBOL、C、および CICS プログラムから SNMP モニターに送信することによってモニター目的で使用できます。

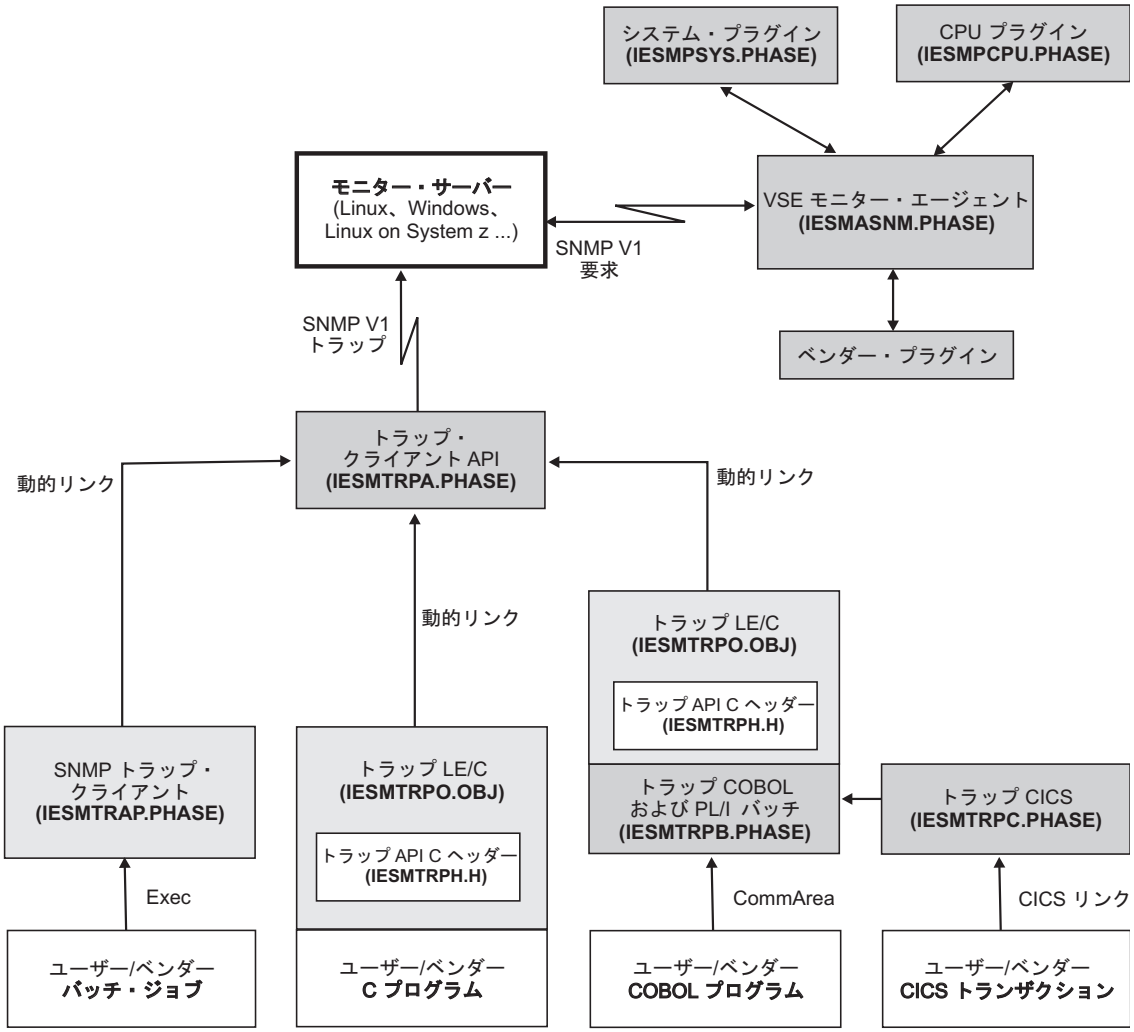


図 *30. VSE Monitoring Agent* への *SNMP* トラップの送信

注:

1. トラップ・クライアント API (IESMTRPA.PHASE) は、z/VSE 5.1 以降で使用できます。
2. z/VSE 5.1 より前 のバッチ・ジョブで SNMP トラップ・クライアント を実装した場合は、そのバッチ・ジョブを変更する必要はありません。 SNMP トラップ・クライアントがインターフェースやパラメーターを変更せずに トラップ・クライアント API も使用するためです。
3. トラップ・クライアント API (IESMTRPA.PHASE) は、以下の IBM 提供インターフェースとともに使用できます。
   - トラップ COBOL および PL/I バッチ・インターフェース (IESMTRPB.PHASE)
   - トラップ CICS インターフェース (IESMTRPC.PHASE)
   - トラップ LE/C インターフェース (IESMTRPO.OBJ) (トラップ API C ヘッダー (IESMTRPH.H) を含む)

4. 133 ページの図 30 に示されているメンバー (IESMTRAP.PHASE、IESMTRPA.PHASE、IESMTRPB.PHASE、IESMTRPC.PHASE、IESMTRPO.OBJ、および IESMTRPH.H) は、サブライブラリー PRD1.BASE 内にあります。

VSE Monitoring Agent の構成

VSE Monitoring Agent は、z/VSE (静的または動的) 区画で稼働するバッチ・アプリケーションです。以下を行います。

- SNMP エージェントの実装
- SNMP クライアントからの SNMP バージョン 1 要求に応答

注: SNMP プロトコルについて詳しくは、 SNMP RFC (RFC 1157) を参照してください。

図 31 は、VSE Monitoring Agent の開始に使用できるサンプル・ジョブ (SKSTMAS) です。これは、 VSE/ICCF ライブラリー 59 にあります。

```
* $$ JOB JNM=STARTMAS,DISP=L,CLASS=R
// JOB STARTMAS STARTS THE SNMP MONITORING AGENT
* ************************************************** *
* This Job starts the SNMP MONITORING AGENT.          *
* Please change the ID and the SYSPARM card if necessary *
* ************************************************** *
// ID USER=VCSRV,PWD=VCSRV
// LIBDEF *,SEARCH=(PRD2.TCPIPC,PRD2.CONFIG,PRD1.BASE,PRD2.SCEEBASE)
// OPTION SYSPARM='00'
// EXEC IESMASNM,PARM='DD:PRD2.CONFIG(IESMASCF.Z)'
/*
/&
* $$ EOJ
```

図 *31. VSE Monitoring Agent* を開始するためのジョブ

図 31 の説明は以下のとおりです。

- VSE Monitoring Agent は、プログラム IESMASNM を実行することにより開始されます。
- `PARM='DD:PRD2.CONFIG(IESMASCF.Z)'` ステートメントは、構成ファイルがある場所を指定します。サンプルの構成ファイル (135 ページの図 32 に表示) は、PRD2.CONFIG ライブラリーで提供されます。

```
* ****************************************** *
* CONFIG FILE FOR z/VSE SNMP MONITORING AGENT   *
* ****************************************** *
* SNMP COMMUNITY NAME:
COMMUNITYNAME = 'public'
* PORT (default SNMP Port 161):
PORT = '161'
* SYSTEM PLUGIN
PLUGIN = 'IESMPSYS'
PARM = 'DD:PRD2.CONFIG(IESMPSCF.Z)'
* CPU PLUGIN
PLUGIN = 'IESMPCPU'
* SAMPLE PLUGIN
* THE SAMPLE PLUGIN IS SHIPPED AS SOURCE CODE, YOU
* HAVE TO COMPILE IT, IF YOU WANT TO USE IT
* PLUGIN = 'IESMPSMP'
```

図 *32. VSE Monitoring Agent* で使用されるサンプル構成ファイル

- セキュリティー上の理由から、IBM は、構成ファイル内の COMMUNITYNAME を変更することをお勧めします。 COMMUNITYNAME など、設定のいずれかを変更するには、サンプル構成ファイル SKMASCFG (図 32 に表示) およびサンプル・ジョブ IESMASCFG (図 33 に表示) を使用して、PRD2.CONFIG 内で新規構成ファイルまたはカスタマイズした構成ファイルを作成できます。サンプル・ジョブ IESMASCF は、VSE/ICCF ライブラリー 59 で提供されており、自動的に SKMASCFG が含まれます。

```
* $$ JOB JNM=IESMASCF,DISP=D,CLASS=0
// JOB IESMASCF CATALOG SNMP MONITORING AGENT CONFIGURATION MEMBER
// EXEC LIBR,PARM='MSHP'
ACCESS S=PRD2.CONFIG
CATALOG IESMASCF.Z REPLACE=Y
* $$ SLI ICCF=(SKMASCFG),LIB=(YY)
/+
/*
/&
* $$ EOJ
```

図 *33.* 構成ファイルを置換するために *VSE Monitoring Agent* で使用されるジョブ

SNMP 通信のセキュリティー最大化

VSE Monitoring Agent は、SNMP バージョン *1* とのみ一緒に使用できます。SNMP バージョン 1 には、単一のセキュリティー・メカニズムがあり、コミュニティー名と呼ばれています。

- コミュニティー名は、各 SNMP 要求に含まれている必要があります。これは、SNMP プロトコルには必須です。
- コミュニティー名が、z/VSE システムの構成ファイル (図 32 に表示) 内のコミュニティー名と一致しない場合、VSE Monitoring Agent は要求を無視して、それ以上処理を実行しません (通常、構成ファイルの名は IESMASCF.Z であり、デフォルトでライブラリー PRD2.CONFIG に保管されます)。

  注: コミュニティー名には大/小文字の区別があります。
- VSE Monitoring Agent 構成ファイルで、デフォルトのコミュニティー名は、public です。ただし、これは多くのオペレーティング・システムでデフォルト

値です。同様に、コミュニティー名 private も広く使用されています。したがって、デフォルト値の変更を強くお勧めします。

- 結果として、SNMP 内のコミュニティー名にはパスワードと同様の機能があります。
- SNMP の要求、応答、およびトラップは、必ず暗号化されずに送信されることに注意してください。つまり、ハッカーがネットワーク・トラフィックを「listen」(覗き見) している場合、コミュニティー名を取得するのは、比較的容易です。

SNMP を使用すると、通常、構成値をターゲット・システムまたはターゲット・サーバーに設定することもできます。ただし、セキュリティー上の理由から、これらの SNMP 要求は VSE Monitoring Agent により無視されます。したがって、クライアントのみが、モニター・エージェントおよびそのプラグインにより提供されたデータを読み取ることができます。

IBM 提供のモニター・プラグインの構成

VSE Monitoring Agent は、容易に拡張できるように設計されています。IBM は、以下をモニターするためにプラグイン・インターフェースを作成しています。

- z/VSE 固有データ
- 独自プログラムで収集したユーザー独自データ

VSE Monitoring Agent に使用可能なプラグインを構成できます。以下の z/VSE モニター・プラグインが現在使用可能です。

- システム・プラグイン。システム・データを収集します。
- CPU プラグイン。CPU データを収集します。
- サンプル・プラグイン。独自プラグインの作成方法を示します。

CPU プラグインおよびサンプル・プラグインはすぐに使用できます。ただし、システム・プラグインの構成は、使用する前に変更する必要があります。システム・プラグイン (図 34 に表示) の構成ファイルを変更するには、 VSE/ICCF ライブラリー 59 でも提供されるファイル IESMPSCF (137 ページの図 35 に表示) を使用します。このジョブは、自動的に構成ファイル (SKMPSCFG) を含みます。

```
* ****************************************** *
* CONFIG FILE FOR MONITORING PLUGIN IESMPSYS   *
* ****************************************** *
* ENTER CONTACT INFORMATION AND LOCATION HERE
CONTACT = 'Please change contact in IESMPSCF.Z config
file'
LOCATION = 'Please change location in IESMPSCF.Z config
file'
* THE SYSTEM NAME AND DESCRIPTION ARE OPTIONAL
*DESC = 'z/VSE TEST SYSTEM'
*SYSNAME = 'VSETestSystem'
```

図 *34.* システム・プラグインに属する構成ファイル

```
* $$ JOB JNM=IESMPSCF,DISP=D,CLASS=0
// JOB IESMPSCF CATALOG MONITORING SYSTEM PLUGIN
CONFIGURATION MEMBER
// EXEC LIBR,PARM='MSHP'
ACCESS S=PRD2.CONFIG
CATALOG IESMPSCF.Z REPLACE=Y
* $$ SLI ICCF=(SKMPSCFG),LIB=(YY)
/*
/&
* $$ EOJ
```

図 *35.* システム・プラグインで使用される構成ファイルを置き換えるジョブ

- プラグインで処理されるデータの概要を知りたい場合は、サブライブラリー PRD1.BASE 付属の IESMPMIB.Z として提供される MIB (管理情報ベース) を単に表示するのみでわかります。「MIB」の概念は、 RFC 2578 で説明されています。
- 接頭部 1.3.6.1.4.1.2.6.221 が付いた OID は、 IBM が z/VSE 製品に提供するプラグイン用に予約済みです。

ユーザー独自のデータを収集するために、独自のプラグインを作成する場合は、サンプル・プラグイン (IESMPSMP.C) の C ソース・コードに加えて、プラグイン API がライブラリー PRD1.BASE 内の C ヘッダー・ファイル (IESMPLGH.H) として出荷されています。詳しくは、 146 ページの『VSE Monitoring Agent 用の独自プラグインの作成』を参照してください。

VSE Monitoring Agent で使用可能なコマンド

VSE Monitoring Agent のコマンドを入力するには、 z/VSE コンソールで次のコマンドを入力します。

```
msg jobname,data=command
```

ここで、

- *jobname* は、VSE Monitoring Agent の名前です。
- *command* は、 138 ページの図 36 にリストされたコマンドのいずれかです。

VSE Monitoring Agent で使用できるすべてのコマンドをリストするには、 z/VSE コンソールで次のように入力します。

```
msg jobname,data=help
```

138 ページの図 36 は、コマンド入力で表示されるリストを示しています。

```
msg startmas,data=help
AR 0015 1I40I  READY
R1 0045 IESMA105I PLEASE USE "MSG XX,DATA=COMMAND" WITH THE
R1 0045 FOLLOWING COMMANDS:
R1 0045 HELP         DISPLAYS THIS INFORMATION
R1 0045 STATUS       DISPLAYS SERVER STATUS
R1 0045 RESETSTAT    RESETS STATISTICS
R1 0045 LISTOIDS     LISTS HANDLED OIDS
R1 0045 LISTOIDSDET  LISTS HANDLED OIDS (DETAILED)
R1 0045 LISTPLUGINS  LISTS ACTIVE PLUGINS
R1 0045 SHUT         ENDS THE SERVER
R1 0045 SHUTDOWN     ENDS THE SERVER
```

図 *36. VSE Monitoring Agent* で使用できるコマンドのリスト

VSE Monitoring Agent の状況を把握するには、次のように入力します。

```
msg jobname,data=status
```

図 37 は、コマンド入力で表示される情報の種類を示しています。

```
msg startmas,data=status
AR 0015 1I40I  READY
R1 0045 IESMA118I AGENT STATUS:
R1 0045 AGENT VERSION:              0004.3000
R1 0045 CONFIG MEMBER:              DD:PRD2.CONFIG(IESMASCF.Z)
R1 0045 PORT:                       161
R1 0045 COMMUNITY STRING:           public
R1 0045 RECEIVED REQUESTS:          5869313
R1 0045 WRONG COMMUNITY STRING:     0
R1 0045 WRONG SNMP VERSION:         0
R1 0045 ANSWERED REQUESTS:          5869313
R1 0045 IESMM002I MONITORING PLUGIN MANAGER STATUS:
R1 0045 MANAGER VERSION:            0004.3000
R1 0045 INSTALLED PLUGINS:          2
R1 0045 HANDLED OIDS:               34
R1 0045 HANDLED OID GROUPS:         1
```

図 *37. VSE Monitoring Agent* の状況把握

データ収集方法の例

JMibBrowser を使用したデータの取得**:** オープン・ソース・プログラムの **JMibBrowser** は Java で作成されています。MIB を使用して、SNMP 要求を送信できます。このプログラムは、次のサイトにあります。

```
http://sourceforge.net/projects/jmibbrowser
```

JMibBrowser を z/VSE で使用するには、z/VSE MIB (PRD1.BASE のメンバー IESMPMIB.Z) を JMibBrowser の mibs ディレクトリーにダウンロードする必要があります。そうすると、新しいサブツリーを、z/VSE OID を含む JMibBrowser MIB ツリーで見ることができます。

SNMP コマンド行ツールを使用したデータの取得**:** コマンド行から SNMP 要求を送信できるツールが、さまざまなオペレーティング・システム向けに多数あります。ツールは、多くの場合、SNMP エージェントに、特定の OID から開始してすべての SNMP エージェントの OID を要求する「walk」を実行できます。通常、

これらのコマンド行ツールを使用するには、(ツールが MIB を処理できないため) サポート対象の OID を把握する必要があります。OID を取得するには、以下のいずれかにより可能です。

- z/VSE MIB (PRD1.BASE 内の IESMPMIB.Z) を確認する。
- z/VSE コンソールで、`msg jobname,data=LISTOIDS` または `msg jobname,data=LISTOIDSDET` と入力します。詳細については、137 ページの『VSE Monitoring Agent で使用可能なコマンド』を参照してください。

バッチ・ジョブでの SNMP トラップ・クライアントの使用

133 ページの図 30 に示されているように、バッチ・ジョブ で *SNMP トラップ・クライアント* を使用して SNMP バージョン 1 トラップを送信すれば、1 つ以上のモニター端末またはモニター・サーバーに以下のような時期を通知できます。

- ジョブ・ストリームの終了
- ジョブ・ストリーム中のエラー発生

SNMP トラップ・クライアントは、バッチ・ジョブで以下のようにパラメーターを受け入れます。

- EXEC パラメーター (`EXEC IESMTRAP,PARM='...'`) として。
- SYSIPT 使用による。

z/VSE 5.1 以降では、バッチ・ジョブで SNMP トラップ・クライアントを使用するときに、シンボリック・パラメーター を SYSIPT 入力に組み込むことができます。

- シンボリック・パラメーターが実行時に解決され、パラメーターが値に置き換えられます。
- シンボリック・パラメーターが JCL で使用されるときと同じ規則が、SYSIPT 入力内のシンボリック・パラメーターに適用されます。

シンボリック・パラメーターの指定方法について詳しくは、「*z/VSE System Control Statements*」(SC34-2679) の章『Job Control and Attention Routine』を参照してください。

140 ページの図 38 は、バッチ・ジョブで SNMP トラップ・クライアントを使用する方法の例を示すサンプル・ジョブ (SKSTTRAP) です。このジョブは VSE/ICCF ライブラリー 59 に登録されていて、このジョブによってプログラム **IESMTRAP.PHASE** が実行されます。

```
* $$ JOB JNM=SNMPTRAP,DISP=L,CLASS=R
// JOB SNMPTRAP STARTS THE VSE SNMP VERSION 1 TRAP CLIENT
* ******************************************************* *
* This Job starts the VSE SNMP VERSION 1 TRAP CLIENT.    *
* Please change the ID and the SYSPARM card if necessary *
* ******************************************************* *
// ID USER=VCSRV,PWD=VCSRV
// LIBDEF *,SEARCH=(PRD2.TCPIPC,PRD2.CONFIG,PRD1.BASE,PRD2.SCEEBASE)
// OPTION SYSPARM='00'
* ******************************************************* *
* This is a sample job, normally you would add the       *
* SNMP TRAP CLIENT at the end of a job stream to         *
* notify you about the results.                          *
* You can add one or more destinations.                  *
* The ADDSYSINF parameter adds system information to     *
* trap packet.                                           *
* If you specify the HELP parameter you will find a      *
* detailed help and a list of all supported parameters   *
* in the job listing.                                    *
* A '*' marks lines as comments                          *
* ******************************************************* *
// EXEC IESMTRAP
DEST=192.168.1.55
DEST=myserver1:162
* DEST=myserver2
OID=1.2.3.4
MSG=This is a test
* MSGINT=99
ADDSYSINF
* This is a comment
* HELP
/*
/&
* $$ EOJ
```

図 *38.* バッチ・ジョブで *SNMP* トラップ・クライアントを開始するジョブ

使用できる SNMP トラップ・クライアント・パラメーターのリストを取得するには、Help パラメーター (EXEC IESMTRAP,PARM='HELP') を指定します。 141 ページの図 39 は、Help パラメーターの指定方法および指定によりリストされる情報のタイプを表示します。

```
// LIBDEF PHASE,SEARCH=(PRD2.TCPIPC,PRD1.BASE)
// EXEC IESMTRAP,PARM='HELP'
1S54I  PHASE IESMTRAP IS TO BE FETCHED FROM PRD1.BASE
IESMTRAP 2.0.0
IESMA908I IESMTRAP 2.0.0 HELP:
IESMTRAP SENDS A VERSION 1 SNMP TRAP PACKET TO ONE OR MORE DESTINATIONS.
PARAMETERS:
 DEST:      DESTINATION OF THE SNMP TRAP PACKET
            (MULTIPLE ARE POSSIBLE)
            E.G.: DEST=192.168.1.5
                  DEST=192.168.1.5:141
                  DEST=MYSERVER.MYDOMAIN
 OID:       TRAP OID
            E.G.: OID=1.2.3.4.5
 MSG:       MESSAGE
            E.G.: MSG=BACKUP JOB FINISHED
 MSGINT:    MESSAGE (NUMBER)
            E.G.: MSGINT=1234
 COMMUNITY: COMMUNITY STRING (OPTIONAL, DEFAULT: PUBLIC)
            E.G.: COMMUNITY=PRIVATE
 TRAPTYPE:  SPECIFIC CODE (OPTIONAL)
            E.G.: TRAPTYPE=112233
 ADDSYSINF: ADDS SYSTEM INFORMATION TO THE TRAP (OPTIONAL)
ATTENTION: ONLY ONE MSG OR ONE MSGINT IS ALLOWED !
REMARK:
 HELP WILL SHOW THIS PAGE.
 YOU CAN USE * FOR COMMENTS.
 USE - AS CONTINUATION CHARACTER.
1S55I LAST RETURN CODE WAS 0000
EOJ START MAX.RETURN CODE=0000
DATE 02/01/2013, CLOCK 12/49
```

図 *39. SNMP* トラップ・クライアントで使用できるパラメーターのリスト

LE/C プログラムでのトラップ・クライアント API の使用

このトピックでは、*LE/C* プログラム でトラップ・クライアント API を使用して SNMP バージョン 1 トラップを送信する方法について説明します。このためには、以下を実行する必要があります。

1. LE/C プログラムにトラップ API C ヘッダー (IESMTRPH.H) を組み込みます。
2. LE/C プログラムのリンク・エディット中にトラップ LE/C インターフェース (IESMTRPO.OBJ) を組み込みます。

LE/C プログラムで使用できる関数については、トラップ API C ヘッダー (IESMTRPH.H) に説明があります。

注**:** LE/C プログラムは、トラップを複数の宛先に同時に送信できます。

トラップ・クライアント APIを使用する LE/C プログラム内での呼び出しの順序は一般的に次のようになります。

1. `TRP_initPrivateData` - 専用データ構造を初期化します。
2. `TRP_ addDestination` - 宛先を追加します。この関数は、複数の宛先を追加するために何度も呼び出すことができます。
3. `TRP_setOid` -「送信者」OID を設定します。

4. TRP_setCommunity - コミュニティー・ストリングを設定します。
5. TRP_setMsgInt または TRP_setMsg - メッセージ (整数またはストリング) を追加します。
6. TRP_addSystemInfo (オプション) - システム情報をトラップに追加します。
7. TRP_checkData - すべてのデータが有効であること、およびトラップが送信可能であることを検査します。
8. TRP_sendTrap - トラップを送信します。
9. TRP_cleanupPrivateData - 専用データをクリーンアップします (また、この関数は、追加された宛先をすべて削除します)。

トラップ LE/C インターフェースで提供される関数

トラップ LE/C インターフェース (IESMTRPO.OBJ) およびトラップ API C ヘッダー (IESMTRPH.H) には、以下のプログラミング・インターフェースが用意されています。

- int TRP_initPrivateData(void** lplpPrivateData);
- int TRP_cleanupPrivateData(void* lpPrivateData);
- int TRP_sendTrap(void* lpPrivateData);
- int TRP_addDestination(destination* dest, void* lpPrivateData);
- void TRP_setDebugMode(int mode, void* lpPrivateData);
- int TRP_setTrapType(int type, void* lpPrivateData);
- int TRP_setMsgInt(int value, void* lpPrivateData);
- int TRP_setMsg(char* value, void* lpPrivateData);
- int TRP_setOid(char* oid, void* lpPrivateData);
- int TRP_setCommunity(char* community, void* lpPrivateData);
- int TRP_addSystemInfo(int addSysInfo, void* lpPrivateData);
- int TRP_checkData(void* lpPrivateData);

これらの関数について詳しくは、トラップ API C ヘッダー (IESMTRPH.H) を参照してください。このメンバーは、サブライブラリー PRD1.BASE 内にあります。

COBOL プログラムおよび PL/I プログラムでのトラップ・クライアント API の使用

このトピックでは、*COBOL* および *PL/I* プログラムでトラップ・クライアント API を使用して SNMP バージョン 1 トラップを送信する方法について説明します。

トラップ COBOL および PL/I バッチ・インターフェース (IESMTRPB.PHASE) はトラップ・クライアント API を使用するためにトラップ LE/C インターフェース (IESMTRPO.OBJ) を内部で使用します。これは、133 ページの図 30 に示します。

注: COBOL プログラムおよび PL/I プログラムは、トラップを複数の宛先に同時に送信することはできません。代わりに、COBOL プログラムおよび PL/I プログ

ラムは、トラップ COBOL および PL/I バッチ・インターフェース (IESMTRPB.PHASE) を何回か呼び出す必要があります。

トラップ COBOL および PL/I バッチ・インターフェースを使用するには、 144 ページの図 40 に示されているコピーブックを使用する必要があります。

トラップ・クライアント API を使用する COBOL プログラムは次のようになります。

```
* ------------------------------------------------
*    TRAP INTERFACE Sample Program
* ------------------------------------------------
 Identification Division.
 Program-ID.  TRPINTF.

 Data Division.

 Working-Storage Section.
 01 MTRA-AREA.
    03 AREA-LENGTH               PIC S9(9) BINARY.
    03 DEST                      PIC X(64).
    03 COMMUNITY                 PIC X(32).
    03 OID                       PIC X(256).
    03 MSGTYPE                   PIC S9(9) BINARY.
    03 MSG                       PIC X(256).
    03 MSGSTR REDEFINES MSG      PIC X(256).
    03 MSGINT REDEFINES MSG      PIC S9(9) BINARY.
    03 TRAPTYPE                  PIC S9(9) BINARY.
    03 ADDSYSINF                 PIC S9(9) BINARY.
    03 DEBUG                     PIC S9(9) BINARY.
    03 RET-CODE                  PIC S9(9) BINARY.

 01 IESMTRPB         PIC X(8) VALUE 'IESMTRPB'.
 Procedure Division.

     Move Length Of MTRA-AREA to AREA-LENGTH.
     Move '9.152.224.43' to DEST.
     Move 0 to RET-CODE.
     Move 'PUBLIC' to COMMUNITY.
     Move '1.2.3.4' to OID.
     Move 0 to DEBUG.
     Move 1 to ADDSYSINF.
     Move 6 to TRAPTYPE.
     Move 1 to MSGTYPE.
     Move 'HELLO VSE WORLD' to MSGSTR.

     DISPLAY "CALLING TRAP INTERFACE ...".
     CALL IESMTRPB USING BY REFERENCE MTRA-AREA.

     DISPLAY "RC:".
     Display RET-CODE.

     Goback.
```

CICS プログラムでの トラップ・クライアント API の使用

このトピックでは、*CICS* プログラムでトラップ・クライアント API を使用して SNMP バージョン 1 トラップを送信する方法について説明します。

CICS プログラム (C、COBOL、または PL/I で作成されたもの) は、トラップ・クライアント API を使用するためにトラップ *CICS* インターフェース (IESMTRPC.PHASE) を使用します。これは、 133 ページの図 30 に示します。

1. ご使用のプログラムで EXEC CICS LINK コマンドからトラップ CICS インターフェース (IESMTRPC) を呼び出します。トラップ COBOL および PL/I バッチ・インターフェース (IESMTRPB.PHASE) と同じレイアウトの COMMAREA が渡されます。
2. トラップ CICS インターフェース (IESMTRPC) が、トラップ・クライアント API を使用するためにトラップ *COBOL* および *PL/I* バッチ・インターフェース (IESMTRPB.PHASE) を内部で使用します。

トラップ CICS インターフェース COMMAREA の入力のレイアウトは、図 40 に示されているトラップ COBOL および PL/I バッチ・インターフェースのコピーブックと同じレイアウトでなければなりません。

EXEC CICS LINK コマンドの使用方法について COBOL の例を以下に示します。

```
EXEC CICS LINK PROGRAM('IESMTRPC')
      COMMAREA(MTRA-AREA)
      LENGTH(AREA-LENGTH)
      RESP(RC) RESP2(RC2)END-EXEC
```

注**:** CICS プログラムは、トラップを複数の宛先に同時に送信することはできません。代わりに、CICS プログラムは、トラップ CICS インターフェース (IESMTRPC) を何回か呼び出す必要があります。

トラップ **COBOL** および **PL/I** バッチ・インターフェースに使用されるコピーブック

図 40 は、トラップ COBOL および PL/I バッチ・インターフェース (IESMTRPB.PHASE) の入力として使用されるコピーブックの内容を示したものです。

```
01 MTRA-AREA.
   03 AREA-LENGTH                 PIC S9(9) BINARY.
   03 DESTINATION                 PIC X(64).
   03 COMMUNITY                   PIC X(32).
   03 OID                         PIC X(256).
   03 MSGTYPE                     PIC S9(9) BINARY.
   03 MESSAGE                     PIC X(256).
   03 MSGSTR REDEFINES MESSAGE    PIC X(256).
   03 MSGINT REDEFINES MESSAGE    PIC S9(9) BINARY.
   03 TRAPTYPE                    PIC S9(9) BINARY.
   03 ADDSYSINF                   PIC S9(9) BINARY.
   03 DEBUG                       PIC S9(9) BINARY.
```

図 *40.* トラップ *COBOL* および *PL/I* バッチ・インターフェースの入力として使用されるコピーブック

次に、図 40 に示されているフィールドがどのように使用されるのかについて説明します。注**:** RET-CODE (このフィールドの値は トラップ・クライアント API によって返される) 以外のすべてのフィールドに値を指定する必要があります。

AREA-LENGTH
データ域の長さ。

DESTINATION
トラップの宛先。例えば、192.168.1.1 や 192.168.1.2:1234 など。

COMMUNITY
トラップのコミュニティー・ストリング。

OID トラップの「送信側」の OID。

MSGTYPE
トラップのタイプ:
- ストリング = 1
- 整数 = 0

MESSAGE
トラップのメッセージ。以下のいずれかとなります。
- MSGSTR
- MSGINT

TRAPTYPE
トラップのタイプ。デフォルトのトラップでは TRAPTYPE = 6 です。

ADDSYSINF
システム情報をトラップに追加します。
- はい = 1
- いいえ = 0

DEBUG
デバッグを有効にします。
- はい = 1
- いいえ = 0

RET-CODE
トラップ・クライアント API で送信された戻りコードが含まれています。

トラップ・クライアント API で生成される戻りコード

以下の戻りコードがトラップ・クライアント API によって生成される可能性があります。

- TRPERR_NO_ERROR
- TRPERR_MEMORY
- TRPERR_INVALID_PARAMETER
- TRPERR_BER
- TRPERR_ICONV
- TRPERR_SOCKET
- TRPERR_INTERNAL
- TRPERR_SEND_ERROR
- TRPERR_SEND_ERROR_FOR_ALL

これらの戻りコードについて詳しくは、サブライブラリー PRD1.BASE 内のトラップ API C ヘッダー (IESMTRPH.H) を参照してください。

VSE Monitoring Agent 用の独自プラグインの作成

独自のデータを収集できるように VSE Monitoring Agent 用に独自のプラグインを作成しなければならない場合があります。この目的のために、ライブラリー PRD1.BASE には以下の項目が用意されています。

- C ヘッダー・ファイル (IESMPLGH.H) として出荷されるプラグイン API
- サンプル・プラグイン (IESMPSMP.C) の C ソース・コード

VSE Monitoring Agent を使用できるようにするために、OID ツリー内でデータを OID として編成する必要があります。 SNMP および MIB の RFC (RFC 1157 および RFC 2578) を参照してください。

プラグインは、VSE Monitoring Agent によって特定のシーケンスで呼び出される一連のコールバック関数で構成されています。それぞれのプラグインは、ユーザーが LE/VSE-C を使用して作成するフェーズから成り、このフェーズは VSE Monitoring Agent のスタートアップ時にロードされます。ユーザーのプラグインは、プラグインに実装された関数を VSE Monitoring Agent が呼び出せるように、IBM が定義するインターフェースを提供する必要があります。これらのコールバック関数は、以下に説明があります。

1. プラグイン・フェーズ (CDLOAD) をロードすることにより、VSE Monitoring Agent は、プラグインのエントリー・ポイント (`PluginMainEntryPoint`) を取得します。
2. プラグインのエントリー・ポイントは、他のプラグイン関数のすべてのエントリー・ポイントを取得するために複数回呼び出されます。それぞれの呼び出しごとに、パラメーター `iFuncID` によって指定された関数のエントリー・ポイントが戻されます。
3. 前の手順が完了すると、エージェントには、プラグイン・フェーズのエントリー・ポイントのリストができます。

以下は、プラグイン・ロード後のシーケンスです。

1. VSE Monitoring Agent は、`MPL_Init` 関数を呼び出します。この関数は、要求に依存しないすべてのリソース (ストレージの割り振り、データベース接続のオープンなど) を初期化するために使用されます。
2. VSE Monitoring Agent は、`MPL_Init` 関数の呼び出し直後に、`MPL_GetCountOfHandledOIDs` および `MPL_GetHandledOID` 関数を呼び出します。
   a. `MPL_GetCountOfHandledOIDs` は、プラグインに受け入れられる処理対象の OID の数を戻します。
   b. `MPL_GetHandledOID` は、あらゆる処理対象 OID について呼び出されます。
3. VSE Monitoring Agent は、特定のプラグインに要求を指示するために、上記の情報を使用できるようになりました。

したがって、OID は、すべてのプラグイン全体で一意である必要があります。OID ツリーで適切な場所を検索するため、最初に、テストの目的で OID 接頭部 1.3.6.1.3 を使用できます。

以降は、以下のいずれかを使用する必要があります。

- ユーザーの会社固有の接頭部

- 既に定義済みの標準 OID。例えば、ネットワーク・データを提供する場合、RFC 1213 に記載されるネットワーク OID を使用可能。

詳しくは、ヘッダー・ファイル IESMPLGH.H ファイルを参照してください。

注: 接頭部 1.3.6.1.4.1.2.6.221 が付いた OID は、 IBM が z/VSE 製品に提供するプラグイン用に予約済みです。

第 14 章 高可用性を実現するために GDPS サポートを使用

GDPS は、*Geographically Dispersed Parallel Sysplex*® の略語です。これは、大規模なコンピューター環境における高可用性および災害復旧のために使用できます。このトピックでは、*GDPS* クライアントにおいて、可用性データを z/VSE システムから収集して、z/OS の下で稼働する中央 *GDPS K-System* にそのデータを送信する方法について説明します。

このトピックに含まれるのは次のとおりです。

- 『GDPS クライアントがどのように使用されるのかについての概説』
- 150 ページの『GDPS クライアント の構成』
- 152 ページの『GDPS クライアントの開始』
- 152 ページの『GDPS クライアントと GDPS K-System の間の通信』
- 153 ページの『GDPS クライアントで使用可能なコマンド』

GDPS クライアントがどのように使用されるのかについての概説

IBM *GDPS K-System* は z/OS の下で稼働し、接続されているすべてのシステムをモニターおよび管理します。各種システムをモニターおよび管理するためには、GDPS K-System と、そこに接続されるシステムとの間で接続を確立できるようにするプロトコルが使用されます。 z/VSE の *GDPS* クライアント には、以下の項目で構成される GDPS K-System へのクライアント 接続が用意されています。

- コマンド受信側 (`cmdreceiver`)
- イベント送信側

z/VSE での GDPS のサポートを 150 ページの図 41 に示します。

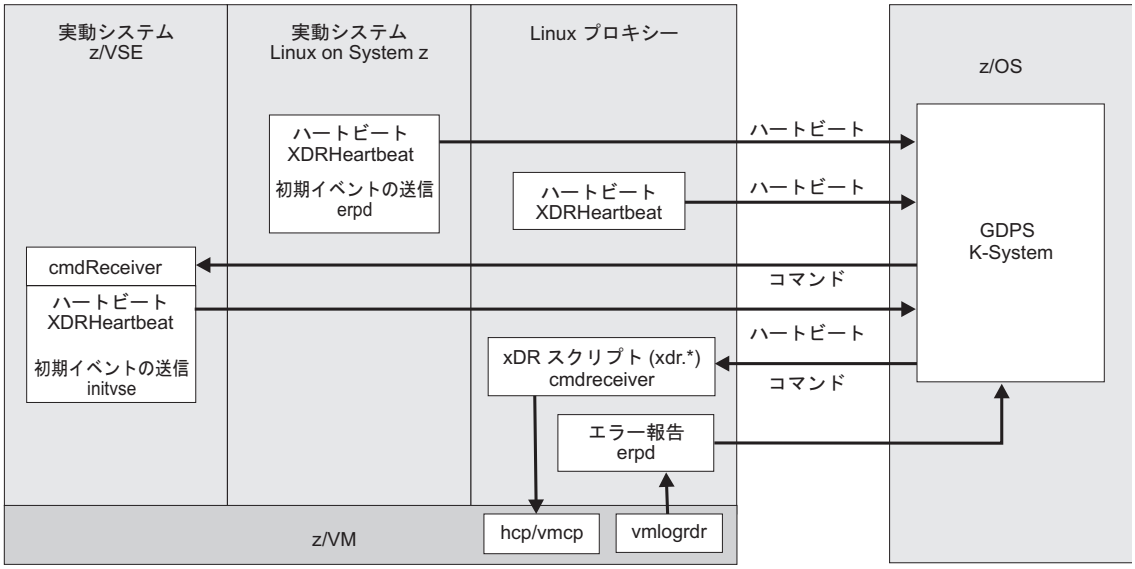


図 *41. z/VSE* での *GDPS* サポートの概説

注**:**

1. GDPS K-System は、LPAR で稼働する z/VSE システムとともに使用することはできません。併用できるのは *z/VM* の下で 稼働する z/VSE システムのみです。
2. z/VSE は、GDPS のハートビート・メカニズムのみをサポートします。
3. GDPS K-System では、z/VSE システムのモニター のみを行うことができます (管理はできません)。
4. GDPS K-System では、z/VSE システムが稼働する z/VM を管理できます。そのため、GDPS K-System では z/VSE システムを間接的に管理 できます (例えば、ディスク・スワップを開始したり、z/VSE システムが稼働している z/VM ゲストをシャットダウンしたりできます)。

GDPS クライアント の構成

GDPS クライアントを開始するには、サンプル・ジョブ SKGDPSCF (VSE/ICCF ライブラリー 59 にある) を使用して GDPS クライアントを構成しておく必要があります。

```
$$ JOB JNM=GDPSCFG,DISP=D,CLASS=0
// JOB GDPSCFG CATALOG GDPS CLIENT CONFIGURATION MEMBER
// EXEC LIBR,PARM='MSHP'
ACCESS S=PRD2.CONFIG
CATALOG IESGDPCF.Z REPLACE=Y
*----------------------------------------*
* GDPS CLIENT CONFIGURATION *
*----------------------------------------*
* NAME OF THIS GDPS CLIENT (MUST BE SAME AS YOUR HOSTNAME)
GDPSCNAME='MYVSE'
* RECEIVER (LOCAL) PORT
RECEIVERPORT='16111'
* INTERVAL (IN SECONDS) BETWEEN HEARTBEATS
HEARTBEATINTERVAL='10'
* TIMEOUT
TIMEOUT='60'
* CURRENT MAINTENANCE STATUS
MAINTENANCE='OFF'
* CURRENTLY ACTIVE SITE
SITE='1'
*----------------------*
* SITE 1 CONFIGURATION *
*----------------------*
* IP/HOSTNAME OF THE GDPS K-SYSTEM
SITE1_KSYSTEM='123.123.1.1'
* PORT OF THE GDPS K-SYSTEM
SITE1_KSYSTEMPORT='16112'
*----------------------*
* SITE 2 CONFIGURATION *
*----------------------*
* IP/HOSTNAME OF THE GDPS K-SYSTEM
SITE2_KSYSTEM='123.123.1.2'
* PORT OF THE GDPS K-SYSTEM
SITE2_KSYSTEMPORT='16112'
/+
/*
/&
$$ EOJ
```

図 *42.* ジョブ *SKGDPSCF* を使用した *GDPS* クライアントの構成

デフォルトでは、図 42 に示されているジョブにより、サブライブラリー *PRD2.CONFIG* 内に *IESGDPCF.Z* という名前のメンバーが作成されます。必要に応じて、デフォルト値を変更してから、このジョブをサブミットします。

また、図 42 に示されているジョブ内の以下のパラメーターに対して独自の値を指定することもできます。

GDPSCNAME
GDPS K-System で表示される名前/ID。注**:** GDPSCNAME は、z/VSE システムのホスト名と同じでなければなりません。

RECEIVERPORT
着信要求に使用されるローカル・ポート。

HEARTBEATINTERVAL
GDPS K-System にハートビートが送信される時間間隔 (秒)。

TIMEOUT
GDPS K-System でこの GDPS クライアントのハートビートが待機される期間 (秒)。この期間が過ぎるとタイムアウトになります。

MAINTENANCE
GDPS クライアントを保守モード で開始するのかどうかを示します。

SITE 現在アクティブなサイト。

***IP*/ホスト名**
GDPS K-System で使用される 2 つの各サイトの IP/ホスト名。

ポート
GDPS K-System で使用される 2 つの各サイトのポート名。

GDPS クライアントの開始

GDPS クライアント は、z/VSE (静的または動的) 区画で稼働するバッチ・アプリケーションです。

図 43 は、GDPS クライアントの開始に使用できるサンプル・ジョブ (SKSTGDPS) です。これは、 VSE/ICCF ライブラリー 59 にあります。このジョブでは、サブライブラリー PRD1.BASE に保管されているフェーズ IESGDPSC.PHASE が実行されます。

```
$$ JOB JNM=STRTGDPS,DISP=L,CLASS=R
// JOB STRTGDPS STARTS THE GDPS CLIENT
* ************************************************** *
* This Job starts the GDPS CLIENT.                   *
* Please change the ID and the SYSPARM card if necessary *
* ************************************************** *
// ID USER=VCSRV,PWD=VCSRV
// LIBDEF *,SEARCH=(PRD2.TCPIPC,PRD2.CONFIG,PRD1.BASE,PRD2.SCEEBASE)
// OPTION SYSPARM='00'
// EXEC IESGDPSC,PARM='DD:PRD2.CONFIG(IESGDPCF.Z)'
/*
/&
$$ EOJ
```

図 *43. GDPS* クライアントを開始するサンプル・ジョブ *SKSTGDPS*

GDPS クライアントと GDPS K-System の間の通信

このトピックでは、GDPS クライアントが GDPS K-System とどのように通信するのかについて概説します。

1. GDPS クライアントは、前回のシャットダウン時に保守モードだった 場合は、開始 時も保守モードになります。そのため、ハートビートを開始するために STOPMAINTENANCE コマンドを発行する必要があります。 GDPS クライアントは、保守モードになっていない場合は、開始された後に、構成済みの GDPS K-System に `init` パケットを送信しようとします。
2. GDPS クライアントが保守モードである 場合は、保守モードを終了するコマンド (例えば、 153 ページの『GDPS クライアントで使用可能なコマンド』で説明されている STOPMAINTENANCE コマンド) が入力されるまで、通信は行われません。
3. GDPS K-System で `init` パケットに返される応答には、GDPS クライアントを強制的に 2 番目のサイトに切り替えるコマンドが含まれている可能性があり

ます。 GDPS クライアントは、応答を処理した後に、定義されている間隔 で GDPS K-System にハートビートを送信し始めます (151 ページの図 42 の HEARTBEATINTERVAL を参照)。

4. GDPS K-System でスイッチ・コマンドが GDPS クライアントに送信されると、GDPS クライアントはサイトを切り替えて、2 番目の GDPS K-System にハートビートを送信し始めます。
5. GDPS クライアントの制御シャットダウン時には、GDPS クライアントは最初に GDPS K-System に「maintenance」パケットを送信して、GDPS K-System がハートビートをこれ以上待機しないようにします。

GDPS クライアントを保守モードに切り替えるには、STARTMAINTENANCE コマンドを使用する必要があります。保守モードを終了するには、STOPMAINTENANCE コマンドを使用する必要があります。これら両方のコマンドについては、『GDPS クライアントで使用可能なコマンド』を参照してください。

注:

1. GDPS クライアントは、以下の場合に構成メンバーを更新します。
   - GDPS クライアントがサイトまたは保守モードを変更しなければならない場合。
   - ユーザーが、パラメーターを変更することによって構成ファイル内の値をリセットする場合。
2. 重要な変更が失われないように、GDPS クライアントは同じライブラリー内に .SAVE バックアップ・ファイルを自動的に作成します。

GDPS クライアントで使用可能なコマンド

GDPS クライアント のコマンドを入力するには、 z/VSE コンソールで次のコマンドを入力します。

```
msg jobname,data=command
```

ここで、

- *jobname* は、GDPS クライアント の名前です。
- *command* は、下記のリストにあるいずれかのコマンドです。

z/VSE コンソールでは、以下のコマンドを入力できます。

HELP 有効なコマンドの概要、および各コマンドの簡略説明を表示します。

STATUS
GDPS クライアントの状況 (現在アクティブなサイト、受信された要求の数、送信されたハートビートの数) を表示します。

RESETSTAT
受信要求数および送信ハートビート数のカウンターをリセットします。

STARTMAINTENANCE
保守モードを開始します。

STOPMAINTENANCE
保守モードを停止します。

SETGDPSCNAME=*yyy*
GDPS クライアントの名前/ID (GDPS K-System で表示されるもの) を値 *yyy* に設定します。

SETSITE1KSYSTEM=*yyy*
サイト 1 の GDPS K-System の IP/ホスト名を値 *yyy* に設定します。

SETSITE1KSYSTEMPORT=*x*
サイト 1 の GDPS K-System のポートを *x* に設定します。

SETSITE2KSYSTEM=*yyy*
サイト 2 の GDPS K-System の IP/ホスト名を値 *yyy* に設定します。

SETSITE2KSYSTEMPORT=*x*
サイト 2 の GDPS K-System のポートを *x* に設定します。

SWITCHSITE
サイトを切り替えます。例えば、現在、サイト 1 がアクティブであれば、それがサイト 2 に切り替えられます。注**:** GDPS クライアントがサイト 1 で初期化された場合に、このコマンドでサイト 2 に切り替えると、サイト 1 の GDPS K-System ではこれはエラーと解釈される可能性があります。

SETRECEIVERPORT=*x*
GDPS K-System からの着信要求の (ローカル) ポートを設定します。注**:** このポートは、GDPS K-System では構成できません。 GDPS クライアントがこのポートを `init` パケットで GDPS K-System に送信するためです。

SETINTERVAL=*x*
GDPS クライアントがハートビートを送信する間隔 (秒) を設定します。

SETTIMEOUT=*x*
タイムアウトまでの期間 (秒) を設定します。 GDPS K-System では、このタイムアウトまでの期間、ハートビートが待機されます。この期間が過ぎると、システムのリカバリーが試みられます。

RELOADCONFIG
構成メンバーを再ロードします。

FORCESHUT
GDPS クライアントをシャットダウンしますが、保守パケットを GDPS K-System に送信しません。注**:** GDPS クライアントが初期化された場合に、このコマンドでシャットダウンを行うと、GDPS K-System ではこれはエラーと解釈されます。

SHUT
GDPS クライアントをシャットダウンしますが、その前に保守パケットを GDPS K-System に送信します。

SHUTDOWN
SHUT と同じです。

第 4 部 プログラミング

第 15 章 VSE Java Beans による Java プログラムの実装

このトピックでは、2 層および 3 層環境内で VSE コネクター・クライアントの *VSE Java Beans* を使用する方法について説明します。

以下の項目があります。

VSE Java Beans のインストールおよび使用場所

VSE Java Beans クラスはすべて、VSE コネクター・クライアント に組み込まれた 1 つの Java アーカイブ **VSEConnector.jar** に含まれます。

VSE Java Beans の一般的なインストール方法は以下のとおりです。

1. Java アプリケーションを開発するワークステーションに VSE コネクター・クライアント をインストールします。
2. これらの Java アプリケーションを実動システムにインプリメントする準備ができたら、以下のファイルを 2 層環境の Web クライアントまたは 3 層環境の物理/論理中間層のいずれかにコピーします。
   - VSEConnector.jar
   - cci.jar

- ibmjsse.jar
- ibmpkcs.jar

VSE Java Beans の使用については、以下を参照してください。

- 2 層環境のアプレットについては、 208 ページの図 102。
- 3 層環境のアプレットについては、 210 ページの図 103。
- 3 層環境のサーブレットについては、 253 ページの図 117。
- 3 層環境の JSP については、 272 ページの図 131。
- 3 層環境の EJB については、 282 ページの図 136。

JavaBeans および EJB と VSE Java Beans との比較方法

JavaBeans、*Enterprise Java Beans (EJBs)*、および *VSE Java Beans* の相違点を以下に示します。

- *JavaBeans* は一般的に、プッシュボタン、スライダー、およびリスト・ボックスなどのビジュアル・コンポーネントです。 *VisualAge® for Java* では、それらのコンポーネントを使用することでプログラミングせずにダイアログをアセンブルできます。 詳しくは、次の Web サイトを参照してください。

  ```
  http://www.javasoft.com/beans/docs
  ```

- Enterprise Java Beans (EJBs) は分散 Java Beans です。 分散 という用語が使用されるのは、以下の理由によります。
  - EJB の 1 つのパーツが Web アプリケーション・サーバー (IBM の WebSphere Application Server など) の JVM 内で実行される。
  - (通常) 1 つのパーツが Web ブラウザーの JVM 内で実行される。

  EJB は、データベース内の 1 つのデータ行 (*Entity Bean*)、またはリモート・データベースへの接続 (*Session Bean*) のいずれかを表します。 一般的に、Entity Bean と Session Bean を一緒に使用すると、データを標準化された方法で、かつ、リレーショナルと非リレーショナルの両方のデータを含む異種環境で表現およびアクセスすることが可能です。 EJB について詳しくは、 275 ページの『第 20 章 EJB を使用したデータの表現』を参照してください。
- z/VSE e-business コネクターで提供される *VSE Java Beans* は、ビジュアル Java Beans ではありません。Java Beans の仕様に準拠していますが、ビジュアル・コンポーネントではありません。代わりに、以下のような z/VSE ベースのオブジェクトを表します 。
  - ファイル・システム (VSE/Librarian、VSE/POWER、VSE/ICCF、VSE/VSAM)。
  - システム・コンポーネント (オペレーター・コンソールなど)。
  - データ・オブジェクト (VSE ライブラリー、POWER 待ち行列項目、および VSAM カタログなど)。

VSE Java Beans クラス・ライブラリーの内容

次の表に、VSE Java Beans のクラスを示します (パッケージ *com.ibm.vse.connector*)。 Java インターフェースはイタリックで示しています。

表 *4. VSE Java Beans* クラス・ライブラリーの内容

クラス	説明
VSECertificateEvent	このクラスは、SSL/TLS 証明書に固有のイベントです。これには、このイベントおよび証明書に関する情報のソースとして VSEConnectionSpec が含まれます。
VSECertificateListener	このインターフェースは、VSECertificateListener によってインプリメントされなければなりません。 VSECertificateListener は、VSEConnectionSpec Bean のメソッド addVSECertificateListener を使用して登録できます。 このリスナーは、SSL/TLS 接続の証明書の通知に使用されます。
VSEConnectionManager	リモート VSE システムへの接続を保持します。
VSEConnectionSpec	このクラスは、ワークステーションから z/VSE ホストへの接続の指定を表します。これは、IBM Common Connector Framework (CCF) のパーツであるインターフェース ConnectionSpec をインプリメントします。 CCF は、別の Java プログラムで再利用できる接続のプールを保持します。 これは、サーブレットまたは Enterprise Java Beans などの WebSphere ベース Java プログラムを作成する際に特に重要です。 この場合、サーブレットなどの存続期間の短いプログラムは、z/VSE ホストへの既存の接続を再利用できます。
VSEConnectorTrace	このインターフェースは、Trace-Class によってインプリメントされなければなりません。 VSE Connector Beans のスタートアップ時に、システムはデフォルト・パッケージにある Trace.class と呼ばれるクラスをロードしようとします。 このクラスは、Class.forName() を使用してロードされます。 通常、使用可能な Trace.class はありません。 この場合、トレース・メッセージは何も書き込まれません。 ユーザーは、インターフェース VSEConnectorTrace をインプリメントすることで独自の Trace.class をインプリメントできます。 VSE Connector Beans は、書き込まれる各行でメソッド writeTrace を呼び出します。 通常、Trace クラスは、トレース・テキストを System.out またはファイルに書き込みます。
VSEConsole	このクラスは、VSE オペレーター・コンソールを表します。 これにより、コンソール・コマンドの発行、メッセージ出力の取得、および指定のメッセージ番号のメッセージの説明の入手が可能です。
VSEConsoleExplanation	このクラスは、指定のコンソール・メッセージの説明テキストを表します。 メッセージ行および行数を取得するメソッドを提供します。
VSEConsoleMessage	このクラスは、コンソール・メッセージを表します。 メッセージ番号、色、属性などのプロパティーを取得するメソッドを提供します。
VSEDli	このクラスは、z/VSE 上の DL/I サブシステムを表します。 PSB のリストを取得するメソッドを提供します。
VSEDliPsb	このクラスは、DL/I PSB と対応する PSB 名を表します。 PSB をスケジュールまたは終了する、チェックポイントをとる、あるいはロールバックを行うためのメソッドを提供します。 PCB のリストも提供します。
VSEDliPcb	このクラスは、GN、GNP、GU、GHU、GHN、GHNP、DLET、ISRT、REPL のような DL/I 要求の実行に使用できる DL/I PCB を表します。
VSEIccf	このクラスは、VSESystem の VSE/ICCF コンポーネントを表します。これは、VSE/ICCF ライブラリーおよびメンバーに 読み取り専用アクセス を提供します。 VSE/ICCF ライブラリーのリストの取得、メンバーの検索、およびプロパティーの取得を行うメソッドを提供します。

表 4. *VSE Java Beans* クラス・ライブラリーの内容 (続き)

クラス	説明
VSEIccfLibrary	このクラスは、VSE/ICCF ライブラリーを表します。 メンバーのリストの取得、メンバーの検索などを行うメソッドを提供します。
VSEIccfMember	このクラスは、ICCF メンバーを表します。 メンバーのコピーとダウンロード、およびプロパティーの取得を行うメソッドを提供します。
VSELibrarian	このクラスは、VSE システムのライブラリアンを表します。 VSE ライブラリー・システムへのアクセスを取得するために必要です。 ライブラリーのリストおよびメンバーの検索を行うメソッドを提供します。 ライブラリーのリストは VSE Connector Server の構成ファイル IESLIBDF に指定されます。
VSELibrary	このクラスは、VSE ライブラリーを表します。 サブライブラリーへのアクセス、メンバーの検索、ライブラリー・プロパティーの取得などを行うメソッドを提供します。 VSESubLibrary も参照してください。
VSELibraryExtent	このクラスは、VSELibrary のエクステント情報を表します。
VSELibraryMember	このクラスは、VSE ライブラリー・メンバーを表します。 メンバーのインスタンスのコピー、削除、ダウンロード、アップロード、およびプロパティーの取得と設定などを行うメソッドを提供します。
VSEMessage	このクラスにより、別のユーザーからのメッセージを受信できるようにします。
VSEPlugin	この抽象クラスは、すべてのユーザー作成プラグイン Bean の基本クラスです。 これにより、プラグインの基本機能をインプリメントすることで独自の Bean により VSE Java Beans クラス・ライブラリーを拡張できます。 VSE Java Beans のプラグインを作成する方法については、 314 ページの『クライアント・プラグインのインプリメント』を参照してください。
VSEPower	このクラスは、指定の VSESystem の VSE/POWER コンポーネントを表します。 POWER 待ち行列へのアクセス、ジョブのサブミット、関連するジョブ出力の入手、待ち行列項目の検索、プロパティーの取得と設定などを行うメソッドを提供します。
VSEPowerEntry	このクラスは、1 つの VSE/POWER 待ち行列項目を表します。 待ち行列項目のコピー、削除、リリース、ダウンロード、プロパティーの取得と設定、待ち行列項目へのローカル・ファイルのアップロードなどを行うメソッドを提供します。
VSEPowerQueue	このクラスは、VSE/POWER 待ち行列のインスタンスを表します。リーダー、リスト、パンチ、または送信待ち行列にできます。 待ち行列項目のリストの取得、待ち行列項目の検索、待ち行列プロパティーの取得などを行うメソッドを提供します。
VSEResource	この抽象クラスは、すべての VSEResource Bean の基本クラスです。 各ソースに提供しなければならない基本機能をインプリメントします。
VSEResourceEvent	このクラスは、VSEResource に固有のイベントを表します。これには、そのイベントのソースおよびオプションとしてイベント関連データが含まれます。 例えば、VSEResourceListener をインプリメントするときに使用されます。
VSEResourceListener	このインターフェースを使用して、ホストへの送信アクションとデータ・オブジェクトの受信を同期にできるコールバック・ルーチンをインプリメントします。 VSE Java Beans の下位レベル機能からリソース・リスナーが呼び出され、受信したデータ・オブジェクトについて通知されます。 リソース・リスナーをインプリメンテーションする多くのサンプルについては、オンライン・ドキュメンテーション (VSEConnectors.html) を参照してください。
VSESubLibrary	このクラスは、指定の VSELibrary のサブライブラリーを表します。このクラスのインスタンスの作成と削除、このサブライブラリー内のメンバーのリストの取得、プロパティーの取得と設定、メンバーの検索などを行うメソッドを提供します。 VSELibrary も参照してください。

表 4. *VSE Java Beans* クラス・ライブラリーの内容 (続き)

クラス	説明
VSESystem	このクラスは、z/VSE ホスト を表します。 このクラスのインスタンスをホストに接続して、VSE ファイル・システムおよび機能にアクセスできるようにする必要があります。
VSEUser	このクラスは、指定の VSE システム上のユーザーを表します。このユーザーが現在アクティブであるかどうかの検査、プロパティーの取得などを行うメソッドを提供します。
VSEVsam	このクラスは、指定の VSESystem の VSAM コンポーネントを表します。 VSAM カタログおよびクラスターへのアクセスを可能にします。
VSEVsamCatalog	このクラスは、VSAM カタログを表します。 このカタログ内のクラスターのリストの取得、プロパティーの取得と設定を行うメソッドを提供します。
VSEVsamCluster	このクラスは、VSAM ファイルを表します。 これは、VRDS 以外のクラスターにできます。 データ・マップのリストの取得 (VSEVsamMap を参照)、プロパティーの取得と設定、クラスターからのデータ・レコードの選択などを行うメソッドを提供します。
VSEVsamField	このクラスは、指定の VSAM レコードのデータ・フィールドを表します。 1 つのデータ・フィールドは、VSAM レコードの特定の 1 つの列を表します。 これは常に、指定の VSAM マップまたはビューのパーツになります。 名前 (列名)、長さ、データ・タイプ、およびレコード内のオフセットで構成されます。
VSEVsamFilter	このクラスは、VSAM ファイルから VSAM データを取得する際のフィルターを表します。 VSEVsamField、ワイルドカードを入れることができるフィルター文字列、およびブール演算で構成されます。
VSEVsamMap	このクラスは、指定の VSAM レコードのデータ・マップを表します。マップによって、VSAM レコードを列とそれぞれのデータ・フィールドに分割し、名前、長さ、データ・タイプ、およびレコード内のオフセットを付けます。 データ・フィールドだけでなく、マップにはマップのフィールドのサブセットであるデータ・ビューを入れることができます。
VSEVsamRecord	このクラスは、指定の VSAM ファイルのデータ・レコードを表します。 レコードのデータ・フィールドにアクセスするメソッドを提供します。
VSEVsamView	このクラスは、指定の VSAM レコード上のデータ・ビューと、指定のデータ・マップを表します。 データ・ビューは常に、指定の VSAM マップのデータ・フィールドのサブセットを指します。 そのため、VSAM マップに対するアクションは必ず、関連するマップのビューに影響を与えます。 例えば、マップを削除すると、そのマップのすべてのビューも削除されます。

VSE Java Bean の Javadoc の例

162 ページの図 44 に、VSE Java Bean のソース・コードから生成された Javadoc の例を示します。 VSE Java Bean は、VSE Java Beans クラス・ライブラリーに属します。

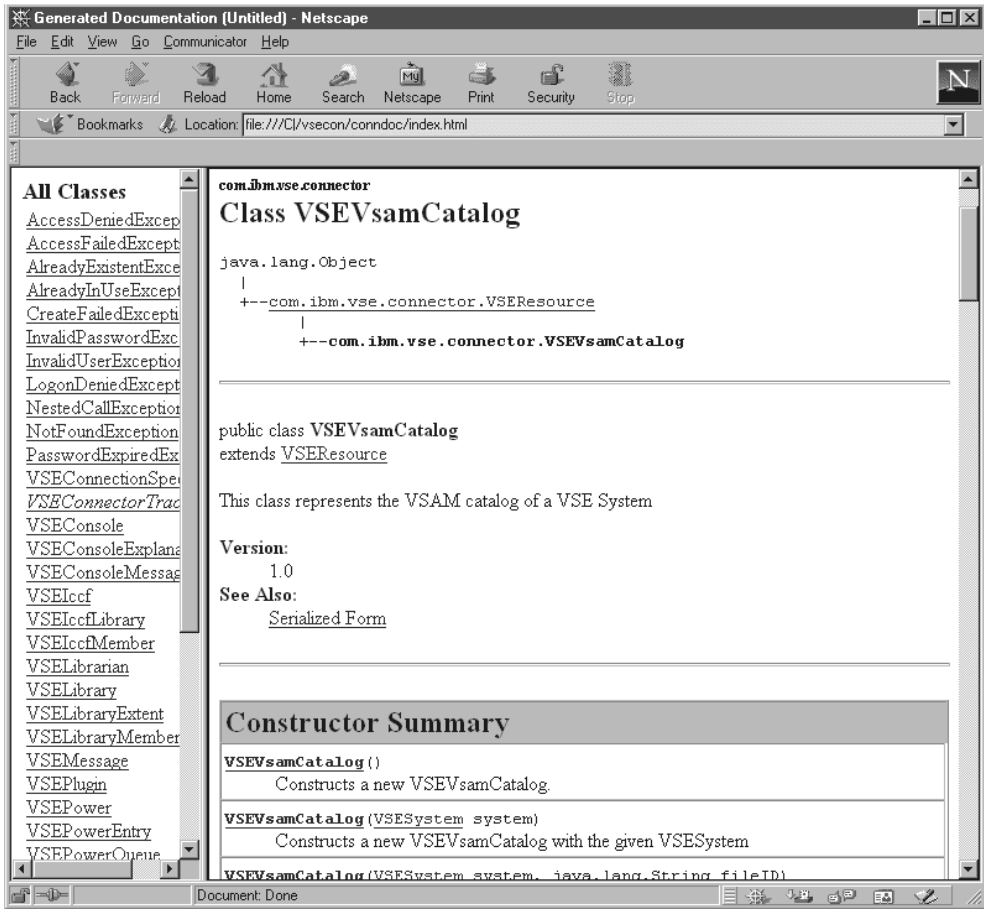


図 *44. VSE Java Beans* クラス・ライブラリーに属する *Javadoc* の例

注**:** VSE コネクター・クライアント で提供されるオンライン・ドキュメンテーションには、VSE Java Beans クラス・ライブラリーに関する詳細情報が記載されています。 詳細については、 30 ページの『オンライン・ドキュメンテーション・オプションの使用』を参照してください。

VSE Java Beans のコールバック・メカニズムの使用

VSE Java Beans のコールバック・メカニズム を使用して、以下のことを行います。

- VSE Java Beans の下位レベル関数から呼び出されるコールバック関数 (*VSEResourceListener*) をインプリメントする。これは、ホストからのデータ受信時に必ず、ホスト接続を listen するものです。
- ホストへのデータの要求を送信して、データを受け取る VSE Bean にリソース・リスナーを追加する。
- 受信したデータをインスタンス別にリソース・リスナーのインプリメンテーションに取得する。
- Bean からリソース・リスナーを除去する。
- コールバック関数から呼び出し元へデータを戻す。

このトピックでは、まずコールバック・メカニズムについて詳しく説明し、さまざまなファイル・システムおよびオペレーター・コンソールへのアクセス方法について説明します。リソース・リスナーのインプリメンテーションのその他の例については、オンライン・ドキュメンテーションも参照してください。

リモート VSE ホストからオブジェクトのリストを戻す VSE Java Beans メソッドはすべて、コールバック・ルーチンを通じてデータ・インスタンスを戻します。 つまり、要求元関数 (呼び出し元) はホストからすべてのデータ項目を受信した後に戻りますが、各データ項目はコールバック・ルーチンに受信された直後に処理できることを意味します。

注: Enterprise Java Beans (EJBs) を記述するときに、コールバック・ルーチンを入れることはできません。 EJB は、マルチスレッド対応ではなく、コールバックをサポートしません。 EJB を適切に使用したシナリオについては EJB サンプルを参照してください。

VSE Java Beans クラス・ライブラリーは *VSEResourceListener* コールバック・インターフェースを提供します。 この Java インターフェースは、 VSE リソースのリストを受信したいアプリケーションごとにインプリメントする必要があります。このリストには、VSE ライブラリー、サブライブラリー、またはメンバーのリストや、あらゆる種類の検索結果のリストなどがあります。インターフェースには、以下の 3 つのメソッドのみが存在します。

listStarted(VSEResourceEvent event)
: 最初のデータ・ブロックがホストから受信される前に呼び出されます。 初期化の目的で使用できます。 このイベントにはデータは含まれません。

listAdded(VSEResourceEvent event)
: イベントに含まれる VSE リソースの各インスタンスに対して呼び出されます。

listEnded(VSEResourceEvent event)
: 最後のデータ・ブロックがホストから受信された後に呼び出されます。 クリーンアップの目的で使用できます。 このイベントにはデータは含まれません。

次に、*VSEResourceListener* インプリメンテーションの例を示します。上記の 3 つの必須メソッドのほかに、受信データを保管してから呼び出し元へ戻る追加メソッドがインプリメントされます。

以下に示す例は、VSAM リソースを listen します。この例では、以下についてもインプリメントします。

- 一般的なリスナー。考えられるすべての VSE リソースを listen します。
- 特殊なリスナー。選択した一部のリソースのみを listen します。

例のこの部分では、2 つのベクターを定義して、受信したすべてのデータ・オブジェクトを保管して累積します。 これにより、データ・オブジェクトの完全なリストを呼び出し元に戻すことができます。

```
public class VsamListener implements VSEResourceListener
{
   public Vector catVector, fileVector;

   /**
```

```
 * constructs a new VsamListener. Two vectors are used to store
 * received resource instances.
 *
 */
public VsamListener()
{
   catVector = new Vector();
   fileVector = new Vector();
}

/**
 * allows the caller to reset the internal vectors.
 *
 */
public void clear()
{
   catVector.removeAllElements();
   fileVector.removeAllElements();
}

/**
 * returns the catalog list.
 *
 */
public Vector getCatalogVector()
{
   return catVector;
}

/**
 * returns the VSAM file list.
 *
 */
public Vector getFileVector()
{
   return fileVector;
}   /**
 * is called for the start of a list of VSEResources before
 * notifying about the list elements.
 * The event does not contain any data.
 *
 * @param event The VSEResourceEvent containing the source
 */
public void listStarted(VSEResourceEvent event)
{
   System.out.println("VsamListener: listStarted()");
}
/**
 * is called for each element of a list of VSEResources.
 * The VSEResourceEvent contains the data instance (see getData()).
 *
 * @param event The VSEResourceEvent containing the source and the data
 */
public void listAdded(VSEResourceEvent event)
{
   VSEResource resource = (VSEResource)(event.getData());
   if (resource instanceof VSEVsamCatalog)
   {
      VSEVsamCatalog cat = (VSEVsamCatalog)resource;
      catVector.addElement(cat);
   }
   else if (resource instanceof VSEVsamCluster)
   {
      VSEVsamCluster file = (VSEVsamCluster)resource;
      fileVector.addElement(file);
   }
}
 /**
 * is called for the end of a list of VSEResources after
 * notifying about the list elements. The event does not
 * contain any data.
```

```
     *
     * @param event The VSEResourceEvent containing the source
     */
    public void listEnded(VSEResourceEvent event)
    {
       System.out.println("VsamListener: listEnded()");
    }
}
```

VSAM カタログのリストを入手する標準的な流れは、次のようになります。

```
public static void main(String argv[ ]) throws IOException
{
   VSESystem system;       // the VSE host object
   VSEVsam vsam;           // the VSAM object
   VsamListener vl;        // implemented as shown above
   Vector vCatalogs;       // used to store the catalog list

   ...
   vsam = system.getVSEVsam();
   vl = new VsamListener();
   vsam.addVSEResourceListener(vl);
   vsam.getCatalogList();
   vsam.removeVSEResourceListener(vl);
   vCatalogs = vl.getCatalogVector();
   ...
}
```

図 *45. VSE Java* クラスを使用して *VSAM* カタログのリストを入手するプログラムの流れ

VSE Java Beans を使用したホストへの接続の例

このトピックでは、VSE ホスト・インスタンスの定義、および物理的な z/VSE ホストへの接続のセットアップについて説明します。

各 VSE ホストは、クラス VSESystem のオブジェクトによって表されます。 物理ホストに接続するには、接続のすべてのプロパティーを保持する接続仕様が必要です。 VSE ホスト接続の作成および再利用について詳しくは、VSE コネクター・クライアント で提供されるオンライン・ドキュメンテーションを参照してください (30 ページの『オンライン・ドキュメンテーション・オプションの使用』を参照)。

デフォルトでは、各ホスト・アクションの後で接続がこのプールに戻されます。 *VSESystem* クラスのメソッド *setConnectionMode* (true/false) があり、これによって、アプリケーションの存続期間中接続を保持できるようにします。 (つまり、各ホスト・アクセス後にプールに戻しません。) これは、実行時間の長いアプリケーションをインプリメントする場合に行う必要があります。 次に、VSESystem の定義およびホストへの接続の単純なコード例を示します。 VSE ホストに接続するには、以下の 2 つのステップが必要です。

ステップ 1: VSEConnectionSpec を作成する

```
/* Create connection specification. The connection spec */
/* holds information about the physical host connection and is */
/* stored permanently in the Common Connector Framework (CCF) */
try {
spec = new VSEConnectionSpec(InetAddress.getByName(ipAddr),
                                              2893,userID,password);
}
catch (UnknownHostException e)
{
   System.out.println("Unknown host : " + e);
   return;
}
spec.setMaxConnections(5);

/* Stay logon with this user for lifetime of this connection */
spec.setLogonMode(true);
```

図 *46. VSE Java Beans* を介したホストへの接続*: VSEConnectionSpec* の作成

ステップ 2: VSESystem を作成する

```
/* Create VSE system instance with this connection */
system = new VSESystem(spec);

/* Hold connection for the lifetime of our application */
system.setConnectionMode(true);
system.connect();
```

図 *47. VSE Java Beans* を介したホストへの接続*: VSESystem* の作成

注**:**

1. *VSESystem*との通信を可能にするために、*VSESystem* の *connect()* メソッドを呼び出す必要はありません。 その代わりに、ホスト接続を必要とするメソッドを呼び出すと、接続がオープンされます。
2. SSL/TLS 接続の構成方法について詳しくは、「*IBM z/VSE* 管理」のトピック『サーバー認証の構成』を参照してください。

VSE Java Beans を使用して SSL/TLS を介したホストへの接続の例

この例では、SSL/TLS を介して z/VSE ホストに接続するために使用できる *3* つの方法について説明します。

この例を実行するための前提条件**:**

- ジョブ SKSSLKEY (ICCF ライブラリー 59 にあります) をサブミットしてサーバー・サイドの VSE 鍵リング・ライブラリーをセットアップしておきます。 詳しくは、「*IBM z/VSE* 管理」の『SSL を使用するためのシステムの準備』の章を参照してください。
- IBM 提供のサンプルのクライアント・サイド鍵リング・ファイル **Keyring.pfx** は VSE コネクター・クライアント のインストールの **samples** ディレクトリーに配置されます。詳しくは、 30 ページの『VSE コネクター・クライアント のインストールの実行』を参照してください。

注: このトピックを開始する前に、IBM と Sun Microsystems の Java Development Kit で提供される SSL/TLS サポートの違いを理解しておく必要があります。詳しくは、「*IBM z/VSE* 管理」の『SSL を使用するためのシステムの準備』の章を参照してください。

この例は、クラス名の指定およびメイン・メソッドのインプリメントから開始されます。

```
public class SSLApiExample implements VSECertificateListener
{
   public SSLApiExample() throws IOException, ResourceException
   {
```

以下に、この例の残りの主なステップを示します。

ステップ 1: IP アドレス、ユーザー ID、パスワードのプロンプト

```
BufferedReader r = new BufferedReader(
                       new InputStreamReader(System.in));
System.out.println("Please enter your VSE IP address:");
String ipAddr = r.readLine();
System.out.println("Please enter your VSE user ID:");
String userID = r.readLine();
System.out.println("Please enter password:");
String password = r.readLine();
```

図 *48. VSE Java Beans* および *SSL* を介したホストへの接続*: IP* アドレス、ユーザー *ID*、パスワードのプロンプト

ステップ 2: VSE システムの接続仕様を作成する

このステップでは、*VSESystem* のインスタンスを使用して、z/VSE ホスト・システムにアクセスできるようにします。 次に、*VSECertificateListener* オブジェクトを使用して、SSL/TLS 接続が確立されたときにサーバー・サイドからこのクライアントに送信されるデジタル・サーバー証明書にアクセスできるようにします。

```
VSEConnectionSpec spec;
try {
   spec = new VSEConnectionSpec(InetAddress.getByName(ipAddr),
                            2893,userID,password);
}
catch (UnknownHostException e)
{
   System.out.println("Unknown host : " + e);
   return;
}
spec.setLogonMode(true);
spec.setSSL(true);
spec.addVSECertificateListener(this);
```

図 *49. SSL* および *VSE Java Beans* を介したホストへの接続*:* 接続仕様の作成

SSL/TLS プロパティーの指定: 選択 1

この選択では、SSL プロパティーを直接 指定します。 ここで定義するプロパティーは、VSE コネクター・サーバー に定義したプロパティーと一致させる必要があります。 詳しくは、ICCF ライブラリー 59 にあるジョブ・スケルトン SKVCSSSL を参照してください (37 ページの『VSE ライブラリー・メンバー SKVCSSSL - SSL/TLS の構成』でも説明しています)。

```
 Properties sslProps = new Properties();
 sslProps.put("SSLVERSION", "SSL");
 sslProps.put("CIPHERSUITES",
"TLS_RSA_WITH_AES_128_CBC_SHA," +
"TLS_RSA_WITH_AES_128_CBC_SHA"
);
 sslProps.put("KEYRINGFILE", "KeyRing.pfx");
 sslProps.put("KEYRINGPWD", "ssltest");
 spec.setSSLProperties(sslProps);

 /* Create VSE system instance with this connection */
 VSESystem system = new VSESystem(spec);
 system.connect();
```

図 *50. VSE Java Beans* および *SSL* を介したホストへの接続*: SSL* プロパティーの指定 *(*選択 *1)*

SSL/TLS プロパティーの指定: 選択 2

2 番目の選択では、Java プロパティー・ファイルを読み取って SSL パラメーターを取得します。 直前の *VSEConnectionSpec* インスタンスを再利用することができます。

```
File file = new File("ssl.prop");
FileInputStream fis = new FileInputStream(file);
DataInputStream in = new DataInputStream(fis);
sslProps = new Properties();
sslProps.load(in);
spec.setSSLProperties(sslProps);
system = new VSESystem(spec);
system.connect();
```

図 *51. VSE Java Beans* および *SSL* を介したホストへの接続*: SSL* プロパティーの指定 *(*選択 *2)*

SSL/TLS プロパティーの指定: 選択 3

3 番目の選択では、既存の SSL プロパティー・ファイル **ssl.prop** を直接使用します。 この IBM 提供のサンプル・ファイルも **samples** ディレクトリーにあります。

```
    spec.setSSLPropertiesFile("ssl.prop");
    system = new VSESystem(spec);
    system.connect();

    /* Cleanup and finish */
    spec.removeVSECertificateListener(this);
}
```

図 *52. VSE Java Beans* および *SSL* を介したホストへの接続*: SSL* プロパティーの指定 *(*選択 *3)*

この例で使用されるクラスのメイン・メソッド

次に、SSL を介して z/VSE ホストに接続するこの例に使用されるクラスのメイン・メソッドを示します。

```
public static void main(String argv[])
throws IOException, ResourceException
{
   SSLApiExample ex = new SSLApiExample();
}
```

図 *53. VSE Java Beans* および *SSL* を介したホストへの接続*:* クラスのメイン・メソッド

ConfirmCertificate メソッドのインプリメンテーション

ConfirmCertificate メソッドは、*VSECertificateListener* インターフェースをインプリメントします。 イベントには、受信されたサーバー証明書に関するさまざまなタイプのデータが含まれます。 この情報を使用して、いずれかを行うことができます。

- 証明書を受諾する (単に戻ることで)。
- 証明書を拒否する (*CertificateRejectedException* を発行して)。

このメソッドでは、エンド・ユーザーにプロンプトを出すか、あるいはエンド・ユーザーにダイアログ・ボックスを表示して、その証明書を受諾するか、拒否するかをエンド・ユーザーに質問できます。

```
    public void confirmCertificate(VSECertificateEvent event)
    throws CertificateRejectedException
    {
       System.out.println("Received server certificate:");
       System.out.println("Common name   : " + event.getCommonName());
       System.out.println("Organization  : " + event.getOrganization());
       System.out.println("Org. unit     : " + event.getOrganizationUnit());
       System.out.println("Issuer name   : " + event.getIssuerCommonName());
       System.out.println("Issuer Org.   : " + event.getIssuerOrganization());
       System.out.println("Issuer O-unit : " + event.getIssuerOrganizationUnit());
       System.out.println("Public key    : " + event.getPublicKey());
       System.out.println("Serial number : " + event.getSerialNumber());
       System.out.println("Valid from    : " + event.getValidFrom());
       System.out.println("Valid to      : " + event.getValidTo());
       System.out.println("Verified?     : " + event.isVerified());
       //throw new CertificateRejectedException("certificate rejected.");
    }
 }
```

図 *54. VSE Java Beans* および *SSL* を介したホストへの接続*: ConfirmCertificate* メソッドのインプリメンテーション

VSE Java Beans を使用したホストへのジョブのサブミットの例

この例には、*JobApiExample* からとられたコード部分が含まれます。これについては、オンライン・ドキュメンテーションに記載しています。 この例は、ジョブをサブミットして、サブミットしたジョブの状況を追跡する方法を示しています。 これを行う以下の 2 つの方法について説明します。

- 最も簡単な方法は、完全な VSE/POWER ジョブを含むローカル・ハード・ディスクにファイルを作成するものです。 このファイルはホストに送信され、ジョブ出力は別のローカル・ファイルに受信されます。
- 2 番目の方法は、ローカル・ファイル・システムにアクセスする必要がなく、メモリー内に JCL を作成し、ジョブ出力も 1 行ごとにメモリーに受信するものです。

注**:** z/VSE ホスト接続の作成および再利用について詳しくは、VSE コネクター・クライアント で提供されるオンライン・ドキュメンテーションを参照してください (30 ページの『オンライン・ドキュメンテーション・オプションの使用』を参照)。

この例は、クラス名の指定およびメイン・メソッドのインプリメントから開始されます。

```
public class JobApiExample
{
   public static void main(String argv[]) throws IOException, ResourceException
   {
```

以下に、この例の残りの主なステップを示します。

ステップ 1: IP アドレス、ユーザー ID、パスワードのプロンプト

ページ *167* ページの『ステップ *1: IP* アドレス、ユーザー *ID*、パスワードのプロンプト』 を参照してください (コードは、VSE Java Beans を介した z/VSE ホストへの接続用のものと同じです)。

ステップ 2: VSE システムの接続仕様を作成する

このステップでは、*VSESystem* のインスタンスを使用して、VSE ファイル・システム (この例では VSE/POWER) にアクセスできるようにします。

```
VSEConnectionSpec spec;
try {
   spec = new VSEConnectionSpec(InetAddress.getByName(ipAddr),
                                        2893, userID, password);
}
catch (UnknownHostException e)
{
   System.out.println("Unknown host : " + e);
   return;
}
spec.setLogonMode(true);
VSESystem system = new VSESystem(spec);
system.setConnectionMode(true);
VSEPower power = system.getVSEPower();
```

図 *55. VSE Java Beans* を介したジョブのサブミット*:* 接続仕様の作成

ステップ 3: ジョブ・ファイルをサブミットする

このステップでは、ジョブ・ファイルがサブミットされ、ローカル・ディスク上のファイル **test.job** に保管されます。 *execute()* メソッドは、ジョブ出力が POWER リスト待ち行列から現行ディレクトリーの outFile **out.txt** に転送された直後に戻ります。

```
File jobFile = new File("test.job");
File outFile = new File("out.txt");
power.executeJob(jobFile, outFile);
```

図 *56. VSE Java Beans* を介したジョブのサブミット*:* ジョブ・ファイルのサブミット

ステップ 4: ジョブ・ファイルを作成してホストへ送信する

このステップでは、ジョブ・ファイルをメモリー内に作成して、z/VSE ホストへ送信します。これには、ローカル・ファイル・システムにアクセスする必要がないという利点があります。ただし、以下のインプリメンテーションが必要です。

- *JobInputStream*。JCL が含まれます。
- *JobOutputStream*。ジョブ出力を受信します。

詳しくは、Java ソース *JobInputStream.java* および *JobOutputStream.java* を参照してください。

```
    JobInputStream job = new JobInputStream();
    JobOutputStream dest = new JobOutputStream();
    power.executeJob(job, dest);
    Vector vLines = dest.getAllLines();
    System.out.println(
       "Number of output lines: " + new Integer(vLines.size()).toString());
    for (int i=0;i<vLines.size();i++)
    {
       System.out.println((String)(vLines.elementAt(i)));
    }
 }
```

図 *57. VSE Java Beans* を介したジョブのサブミット*:* ジョブ・ファイルの作成およびホストへの送信

JobInputStream クラスの主なパーツは、以下のとおりです。

1. クラス名を指定します。

```
public class JobInputStream implements DataInput
{
   int linesRead;
   int maxLines;
```

2. 完全な VSE/POWER ジョブはローカルな String 配列に含まれます。 このため、クラスまたはファイル属性指定などのジョブ・パラメーターを以下の方法で動的に割り当てることができます。
   a. コンストラクターにこれらのパラメーターを引き渡す。
   b. String 配列を変更する。

```
protected String[] jclArray = {
  "* $$ JOB JNM=TEST2,CLASS=0,DISP=D",
  "* $$ LST CLASS=A,DISP=L,PRI=3,LST=SYSLST",
  "// JOB TEST2",
  "// EXEC LIBR",
  " ACC S=IJSYSRS.SYSLIB",
  " LD INW*.PHASE",
  "/*",
  "/&",
  "* $$ EOJ"
};
```

3. 次に、*JobInputStream* クラスのコンストラクターを示します。

```
public JobInputStream()
{
   super();
   linesRead = -1;
   maxLines = jclArray.length;
}
...
```

4. 次に、コールバック・ルーチンを示します。 これは、*VSEPower* インスタンスのジョブ・サブミット・ルーチン *executeJob()* によって呼び出されて、次の JCL 行を取得し、接続されている VSE システムにサブミットします。 このため、このルーチンは各 JCL 行ごとに 1 回呼び出されます。

```
   public String readLine()
   {
      if (linesRead < maxLines-1)
      {
         linesRead++;
         return jclArray[linesRead];
      }
```

```
        else
           return null;
     }
  }
```

VSE Java Beans を使用したオペレーター・コンソールへのアクセスの例

コンソール・コマンドの送信時にコンソール・メッセージを取得するには、次の 2 つの方法があります。

1. クラス *VSEConsole* の *execute()* メソッドの使用。メッセージの完全なリストを戻します。
2. *open()*、*setCommand()*、*getMessage()*、および *close()* の使用。他のメッセージを検索している最中に最初のメッセージを処理できるようにします。

この例には、*ConsoleApiExample* からとられたコード部分が含まれます。これについては、オンライン・ドキュメンテーションに詳しく記載しています。

注: z/VSE ホスト接続の作成および再利用について詳しくは、VSE コネクター・クライアント で提供されるオンライン・ドキュメンテーションを参照してください (30 ページの『オンライン・ドキュメンテーション・オプションの使用』を参照)。

この例は、クラス名の指定およびメイン・メソッドのインプリメントから開始されます。

```
public class ConsoleApiExample
{
   public static void main(String argv[]) throws IOException, ResourceException
   {
```

以下に、この例の残りの主なステップを示します。

ステップ 1: IP アドレス、ユーザー ID、パスワードのプロンプト

ページ *167* ページの『ステップ *1: IP* アドレス、ユーザー *ID*、パスワードのプロンプト』 を参照してください (コードは、VSE Java Beans を介した z/VSE ホストへの接続用のものと同じです)。

ステップ 2: VSE システムの接続仕様を作成する

このステップでは、*VSESystem* のインスタンスを使用して、オペレーター・コンソールにアクセスできるようにします。

```
VSEConnectionSpec spec;
try {
   spec = new VSEConnectionSpec(InetAddress.getByName(ipAddr),
                                2893, userID, password);
}
catch (UnknownHostException e)
{
   System.out.println("Unknown host : " + e);
   return;
}
spec.setLogonMode(true);
VSESystem system = new VSESystem(spec);
system.setConnectionMode(true);
```

図 *58. VSE Java Beans* を介したコンソールへのアクセス*:* 接続仕様の作成

ステップ 3: コンソール・インスタンスを作成してコマンドを送信する

このステップでは、コマンド出力はベクター *vConsOutput* に戻されます。 ここで重要な点は、*cons.execute()* の 3 番目のパラメーターとして終了文字列を指定することです。 指定しないと、関数は最後のメッセージの直後に終了状態を戻すことができません。

```
VSEConsole cons = new VSEConsole(system);
Vector vConsOutput = cons.execute("map", null, "AR 0015 1I40I  READY", 30);
for (int i=0;i<vConsOutput.size();i++)
   System.out.println(
      ((VSEConsoleMessage)(vConsOutput.elementAt(i))).getMessage());
```

図 *59. VSE Java Beans* を介したコンソールへのアクセス*:* コンソール・インスタンスの作成、コマンドの送信

ステップ 4: メッセージを一度に 1 行ずつ取得する

このステップでは、コマンド出力は個別に送信されたものが一連のメッセージとして戻されます。 これを行うには、関数 *open()* および *setCommand()* を使用します。

```
      cons.open();
      cons.setCommand("map");
      VSEConsoleMessage message = cons.getMessage();
      boolean finished = false;
      while (!finished)
      {
         if (message != null)
         {
            System.out.println(message.getMessage());
            if (message.getMessage().indexOf("AR 0015 1I40I  READY") >= 0)
               finished = true;
         }
         message = cons.getMessage();
      }
      cons.close();
   }
}
```

図 *60. VSE Java Beans* を介したコンソールへのアクセス:一度に *1* 行ずつのメッセージの取得

VSE Java Beans を使用した VSAM データへのアクセスの例

この例には、*VsamDisplayExample* からとられた部分が含まれます。これについては、オンライン・ドキュメンテーションに詳しく記載しています。 この例は、マップを使用する VSAM ファイルのデータを表示し、新規レコードをファイルに追加します。ここでは、VSAM クラスター FLIGHT.ORDERING.FLIGHTS にレコードの列 (データ・フィールド) を記述するマップ FLIGHTS_MAP が含まれることを想定しています。

この例は、以下の場合にのみ実行できます。

1. VSAM ファイル FLIGHT.ORDERING.FLIGHTS (KSDS) および FLIGHTS.ORDERING.ORDERS (RRDS) が定義されている。
2. 上記の VSAM ファイルにサンプル・データが含まれている。

これらの VSAM ファイルの定義およびサンプル・データのロード方法について詳しくは、 257 ページの『サンプル用の VSAM クラスターの作成』を参照してください。

注: z/VSE ホスト接続の作成および再利用について詳しくは、VSE コネクター・クライアント で提供されるオンライン・ドキュメンテーションを参照してください (30 ページの『オンライン・ドキュメンテーション・オプションの使用』を参照)。

この例は、クラス名の指定およびメイン・メソッドのインプリメントから開始されます。

```
public class VsamDisplayExample
{
   public static void main(String argv[]) throws IOException, ResourceException
   {
```

以下に、この例の残りの主なステップを示します。

ステップ 1: ローカル変数を定義する

```
String catName  = "VSESP.USER.CATALOG";
String fileName = "FLIGHT.ORDERING.FLIGHTS";
String mapName  = "FLIGHTS_MAP";
```

図 *61. VSE Java Beans* を介した *VSAM* データ*:* ローカル変数の定義

ステップ 2: VSE システムの IP アドレス、ユーザー ID、パスワードのプロンプト

ページ *167* ページの『ステップ *1: IP* アドレス、ユーザー *ID*、パスワードのプロンプト』 を参照してください (コードは、VSE Java Beans を介した z/VSE ホストへの接続用のものと同じです)。

ステップ 3: VSE システムの接続仕様を作成する

このステップでは、*VSESystem* のインスタンスを使用して、VSE ファイル・システム (この例では VSAM) にアクセスできるようにします。

```
VSEConnectionSpec spec;
try {
   spec = new VSEConnectionSpec(InetAddress.getByName(ipAddr),
                                2893,userID,password);
}
catch (UnknownHostException e)
{
   System.out.println("Unknown host : " + e);
   return;
}
spec.setLogonMode(true);
VSESystem system = new VSESystem(spec);
system.setConnectionMode(true);
VSEVsam vsam = system.getVSEVsam();
```

図 *62. VSE Java Beans* を介した *VSAM* データ*:* 接続仕様の作成

ステップ 4: VSEResourceListener を作成する

このステップでは、*VSEResourceListener* を作成して、z/VSE ホストからオブジェクトを検索するごとに通知を行います。 詳しくは、Java ソース・ファイル *RecordListener.java* を参照してください。

```
RecordListener rl = new RecordListener();
vsam.addVSEResourceListener(rl);
```

図 *63. VSE Java Beans* を介した *VSAM* データ*: VSEResourceListener* の作成

ステップ 5: z/VSE ホストから VSAM レコードを取得する

このステップでは、指定のマップを使用して z/VSE ホストから VSAM レコードを検索します。 *selectRecords()* メソッドは、このマップを使用してデータ・アクセ

スを行います。 これにより、ベクター *vRecords* を使用してすべてのレコードにアクセスできます。

```
VSEVsamMap map = new VSEVsamMap(system, catName, fileName, mapName);
VSEVsamCluster cluster = new VSEVsamCluster(system, catName, fileName);
cluster.addVSEResourceListener(rl);
cluster.selectRecords(map);
cluster.removeVSEResourceListener(rl);
Vector vRecords = rl.getRecords();
```

図 *64. VSE Java Beans* を介した *VSAM* データ*:* ホストからの *VSAM* レコードの取得

ステップ 6: VSAM レコードを表示する

このステップでは、次の *2* つ のループを使用します。

- VSAM レコードのリストの取得に使用される外部ループ (ベクター *vRecords* に含まれます)。
- 各レコードのデータ・フィールド (列) の取得に使用される内部ループ。

```
VSEVsamRecord record;
int numMapFields = map.getNoOfFields();
for (int k=0;k<vRecords.size();k++)
{
   System.out.println("Record " + k + ":");
   record = (VSEVsamRecord)(vRecords.elementAt(k));
   for (int i=0;i<numMapFields;i++)
   {
      try {
         System.out.println(map.getFieldName(i) + " : " + record.getField(i));
      }
      catch (Exception e)
      {
         ...
      }
   }
}
```

図 *65. VSE Java Beans* を介した *VSAM* データ*: VSAM* レコードの表示

ステップ 7: VSAM クラスターに VSAM レコードを挿入する

このステップでは、VSAM レコード (新しい飛行便) を VSAM クラスターに挿入します。その飛行便番号が FLIGHT.ORDERING.FLIGHTS という VSAM KSDS クラスターにすでに存在する場合には、例外が発生します。

```
boolean done = false;
VSEVsamRecord newRec;
while (!done)
{
   System.out.println("Please enter a new flight number:");
   System.out.println("(Enter a negative value to quit)");
   String flightNum = r.readLine();
   if ((new Integer(flightNum).intValue()) < 0)
      return;

   /* Check if new flight number already exists ... */
   try {
      newRec = new VSEVsamRecord(system, catName, fileName, mapName);
      newRec.setKeyField(0, new Integer(flightNum));
      newRec.add();
      done = true;
   }
   catch (Exception e)
   {
      System.out.println("Exception when adding new record.");
      if (e instanceof AlreadyExistentException)
         System.out.println("This flight number already exists.");
      else
         ...
   }
}
```

図 *66. VSE Java Beans* を介した *VSAM* データ*: VSAM* レコードの挿入

ステップ 8: ユーザーに列値を入力するプロンプトを出す

このステップでは、ステップ 7 で挿入した VSAM レコードの列値を入力するプロンプトをユーザーに出します。

```
      for (int i=1;i<numMapFields;i++)
      {
         System.out.println("Please enter value for field " + map.getFieldName(i) +
                            " (Length = " + map.getFieldLength(i) + ")");
         String newField = r.readLine();
         if (map.getFieldType(i) == VSEVsamField.TYPE_STRING)
            newRec.setField(i, newField);
         else
            newRec.setField(i, new Integer(newField));
      }

      /* Make changes permanent... */
      newRec.commit();
   }
}
```

図 *67. VSE Java Beans* を介した *VSAM* データ*:* ユーザーの値入力のためのプロンプトの表示

VSE Java Beans を使用した DL/I データへのアクセスの例

この例には、*DliApiExample.Java* からとられた部分が含まれます。これについては、オンライン・ドキュメンテーションに詳しく記載しています。 この例は、DL/I データベースのデータを表示します。 ここでは、 43 ページの『第 6 章 VSE Java Beans を介してアクセスするための DL/I の構成』の説明にあるように、サンプル DL/I データベースを定義してロードしていることを想定しています。

VSE/ESA 2.7 よりも前は、DL/I データには Db2 ストアード・プロシージャーを通じてのみアクセスできました。 VSE/ESA 2.7 以降は、Db2 を使用せずに DL/I データにアクセスできます。Db2 を使用する代わりに、VSE コネクター・サーバーを使用して DL/I データにアクセスします。そのとき、AIBTDLI インターフェースを使用して DL/I にアクセスします。

注: z/VSE ホスト接続の作成および再利用について詳しくは、VSE コネクター・クライアント で提供されるオンライン・ドキュメンテーションを参照してください (30 ページの『オンライン・ドキュメンテーション・オプションの使用』を参照)。

ステップ 1: VSE システム・インスタンスを作成して DL/I にアクセスできるようにする

```
public class DliApiExample
{
  protected static VSEConnectionSpec spec;
  protected static VSESystem system;
  protected static VSEDliPsb psb = null;
  protected static VSEDliPcb pcb = null;

  public static void main(String argv[])
  {
    ...
    byte[] ioarea;
    String[] ssas;

    try
    {
      ...

      /* Create VSE system ... */
      system = new VSESystem(spec);

      /* Get the DLI subsystem */
      VSEDli dli = new VSEDli(system);
```

図 *68. VSE Java Beans* を介した *DL/I* データ*: DL/I* へのアクセスの取得

ステップ 2: PSB をスケジュールする

このステップでは、PSB (プログラム仕様ブロック) をスケジュールします。

```
psb = dli.getDliPsb("STBICLG");
psb.schedule();
System.out.println("  Num PCBs: " + psb.getNumberOfPCBs());
System.out.println("  IO len:   " + psb.getMaxLengthOfIOArea());
```

図 *69. VSE Java Beans* を介した *DL/I* データ*: PSB* のスケジュール

ステップ 3: PCB を取得する

このステップでは、PCB (プログラム連絡ブロック) を取得します。この場合は、PSB の最初の PCB になります。

```
pcb  = psb.getVSEDliPcb(0);
System.out.println("  DBDName = " + pcb.getDBDName());
```

図 *70. VSE Java Beans* を介した *DL/I* データ*: PCB* の取得

ステップ 4: DL/I セグメントをリストする

このステップでは、すべての DL/I セグメントをリストします。 次に、(COBOL サンプル集に定義されている) このセグメントのデータ・レイアウトを示します。

```
/* 01  STPIITM         REDEFINES IOAREA.
       02  ITNUMB    PIC X(6).
       02  ITDESC    PIC X(25).
       02  IQOH      PIC X(6).
       02  IQOR      PIC X(6).
       02  FILLER    PIC X(6).
       02  IUNIT     PIC 9(6).
       02  FILLER    PIC X(105).  */

ssas = new String[1];
ssas[0] = "STPIITM  ";
do
{
  /* Get the next segment */
  ioarea = pcb.call("GN", null, ssas);
  System.out.println("  Status = " + pcb.getStatus());
  if (!pcb.getStatus().equals("  "))
    break;
}
while(true);
```

図 *71. VSE Java Beans* を介した *DL/I* データ*: DL/I* セグメントのリスト

ステップ 5: DL/I セグメントを挿入または更新する

このステップでは、セグメントを挿入または更新します。 最初に GHU 呼び出しが出され、そのセグメントが存在するかどうかが検査されます。 セグメントが存在する場合には更新されます。セグメントが存在しない場合は、新規セグメントが挿入されます。

```
String item = "000700";  // item number to insert/update
ssas = new String[1];
ssas[0] = "STPIITM (STQIINO = " + item + ")";

/* Get the segment */
ioarea = pcb.call("GHU", null, ssas);
if (pcb.getStatus().equals("GB") || pcb.getStatus().equals("GE"))
{
   // allocate a new ioarea
   ioarea = new byte[psb.getMaxLengthOfIOArea()];
   for (int i=0;i<ioarea.length;i++)
     ioarea[i] = 0x00;

   // set the new values into the IOArea
   VSEDliPcb.setStringToBuffer(ioarea,item,0,6);
   VSEDliPcb.setStringToBuffer(ioarea,"INSERTED ITEM",6,25);
   VSEDliPcb.setStringToBuffer(ioarea,"000001",31,6);
   VSEDliPcb.setStringToBuffer(ioarea,"000002",37,6);
   VSEDliPcb.setStringToBuffer(ioarea,"000000",43,6);
   VSEDliPcb.setStringToBuffer(ioarea,"000003",49,6)

   // do the insert
   ssas = new String[1];
   ssas[0] = "STPIITM  ";
   pcb.call("ISRT", ioarea, ssas);
}
else
{
   // segment already existing, do an update
   // set the new values into the IOArea
   VSEDliPcb.setStringToBuffer(ioarea,item,0,6);
   VSEDliPcb.setStringToBuffer(ioarea,"UPDATED ITEM",6,25);
   VSEDliPcb.setStringToBuffer(ioarea,"000004",31,6);
   VSEDliPcb.setStringToBuffer(ioarea,"000005",37,6);
   VSEDliPcb.setStringToBuffer(ioarea,"000000",43,6);
   VSEDliPcb.setStringToBuffer(ioarea,"000006",49,6);

   // do the update
   ssas = new String[1];
   ssas[0] = "STPIITM  ";
   pcb.call("REPL", ioarea, ssas);
}
```

図 *72. VSE Java Beans* を介した *DL/I* データ*: DL/I* セグメントの挿入/更新

ステップ 6: DL/I セグメントを削除する

```
item = "000700";  // item number to delete

// first verify if the segment is existing
ssas = new String[1];
ssas[0] = "STPIITM (STQIINO = " + item + ")";

/* Get the segment */
ioarea = pcb.call("GHU", null, ssas);
System.out.println("  Status = " + pcb.getStatus());

if (pcb.getStatus().equals("GB") || pcb.getStatus().equals("GE"))
{
  // segment not found
  System.out.println("  Segment not found");
}
else
{
  // segment exists, delete it
  ssas = new String[1];
  ssas[0] = "STPIITM  ";
  pcb.call("DLET", null, ssas);
}
```

図 *73. VSE Java Beans* を介した *DL/I* データ*: DL/I* セグメントの削除

ステップ 7: PSB を終了する

```
  psb.terminate();
}
...
```

図 *74. VSE Java Beans* を介した *DL/I* データ*: PSB* の終了

VSE Java Beans を使用した VSE/POWER データへのアクセスの例

この例には、*PowerApiExample* からとられたコード部分が含まれます。これについては、オンライン・ドキュメンテーション に詳しく記載しています (30 ページの『オンライン・ドキュメンテーション・オプションの使用』 ページを参照)。基本として、*VSEPower*、 *VSEPowerQueue*、および *VSEPowerEntry* のインスタンスを処理します。このサンプルでは、読み取り待ち行列からジョブをダウンロードし、指定のテキスト文字列を含むリスト待ち行列項目を検索します。

注: z/VSE ホスト接続の作成および再利用について詳しくは、VSE コネクター・クライアント で提供されるオンライン・ドキュメンテーションを参照してください (30 ページの『オンライン・ドキュメンテーション・オプションの使用』を参照)。

この例は、クラス名の指定およびメイン・メソッドのインプリメントから開始されます。

```
public class PowerApiExample
{
   public static void main(String argv[]) throws IOException, ResourceException
   {
```

以下に、この例の残りの主なステップを示します。

ステップ 1: IP アドレス、ユーザー ID、およびパスワードのプロンプト

ページ 167 ページの『ステップ 1: IP アドレス、ユーザー ID、パスワードのプロンプト』 を参照してください (コードは、VSE Java Beans を介した z/VSE ホストへの接続用のものと同じです)。

ステップ 2: VSE システムの接続仕様を作成する

このステップでは、z/VSE ホストの接続仕様を作成します。 *VSESystem* のインスタンスを使用して、VSE ファイル・システム (この例では、VSE/POWER および VSE/POWER 読み取り待ち行列) にアクセスできるようにします。

```
VSEConnectionSpec spec;
try {
   spec = new VSEConnectionSpec(InetAddress.getByName(ipAddr),
                                2893, userID, password);
}
catch (UnknownHostException e)
{
   System.out.println("Unknown host : " + e);
   return;
}
spec.setLogonMode(true);
VSESystem system = new VSESystem(spec);
system.setConnectionMode(true);
VSEPower power = system.getVSEPower();
VSEPowerQueue readerQueue = power.getReaderQueue();
```

図 *75. VSE Java Beans* を介した *VSE/POWER* データ: 接続仕様の作成

ステップ 3: VSEResourceListener を作成する

このステップでは、*VSEResourceListener* を作成して、z/VSE ホストからオブジェクトを検索するごとに通知を行います。 詳細については、Java ソース・ファイル *PowerListener.java* を参照してください。 メソッド *getEntryList()* は、読み取り待ち行列内のクラス = 0 のすべての項目を取得します。次に、PAUSEBG ジョブの検索が行われて、ローカル・ファイルにダウンロードされます。

```
PowerListener pl = new PowerListener();
readerQueue.addVSEResourceListener(pl);
readerQueue.getEntryList("*", "*", '0');
readerQueue.removeVSEResourceListener(pl);

/* Now get the entry list from the listener */
Vector vEntries = pl.getEntryVector();
VSEPowerEntry entry;
for (int i=0;i<vEntries.size();i++)
{
   entry = (VSEPowerEntry)(vEntries.elementAt(i));

   /* Look for the PAUSEBG job and download it */
   if (entry.getName().equals("PAUSEBG"))
   {
      File tempFile = new File("pausebg.job");
      entry.get(tempFile);
   }
}
```

図 *76. VSE Java Beans* を介した *VSE/POWER* データ*: VSEResourceListener* の作成

ステップ 4: コンパイル出力にエラーがあるかどうかをスキャンする

このステップでは、リスト待ち行列内のコンパイル出力にエラーがあるかどうかをスキャンします。ここでは、compjob というコンパイル・ジョブを F4 でサブミットしていることを前提としています。 次に、*search()* 関数は、compjob に属し、文字列 ==ERROR を含むクラス = 4 のリスト待ち行列項目をすべて検索します。

```
      VSEPowerQueue listQueue = power.getListQueue();
      listQueue.addVSEResourceListener(pl);
      pl.clearVector();
      listQueue.search("compjob", userID, '4', "==ERROR", true);
      listQueue.removeVSEResourceListener(pl);

      /* Get the results from the listener */
      vEntries = pl.getEntryVector();
      if (vEntries.size() == 0)
         System.out.println("No list queue entries contain the string
                           ¥"==ERROR¥".");

      for (int i=0;i<vEntries.size();i++)
      {
         entry = (VSEPowerEntry)(vEntries.elementAt(i));
         System.out.println("String found in : " + entry.getName() + "." +
                           entry.getNumber() + "[" + entry.getSuffix() + "]");
      }
   }
}
```

図 *77. VSE Java Beans* を介した *VSE/POWER* データ*:* コンパイル・エラーのスキャン

VSE Java Beans を使用したライブラリアン・データへのアクセスの例

この例には、*LibrApiExample* からとられたコード部分が含まれます。これについては、オンライン・ドキュメンテーションに詳しく記載しています。 この例は、VSE ライブラリーのリストを取得してから、PRD2 内のサブライブラリーのリストを取得し、PRD2.CONFIG 内のメンバーのリストを取得します。 最後に、PRD2.CONFIG の最初のメンバーをローカル・ハード・ディスクにダウンロードします。

注: z/VSE ホスト接続の作成および再利用について詳しくは、VSE コネクター・クライアント で提供されるオンライン・ドキュメンテーションを参照してください (30 ページの『オンライン・ドキュメンテーション・オプションの使用』を参照)。

この例は、クラス名の指定およびメイン・メソッドのインプリメントから開始されます。

```
public class PowerApiExample
{
   public static void main(String argv[]) throws IOException, ResourceException
   {
```

以下に、この例の残りの主なステップを示します。

ステップ 1: IP アドレス、ユーザー ID、およびパスワードのプロンプト

ページ *167* ページの『ステップ *1: IP* アドレス、ユーザー *ID*、パスワードのプロンプト』 を参照してください (コードは、VSE Java Beans を介した z/VSE ホストへの接続用のものと同じです)。

ステップ 2: VSE システムの接続仕様を作成する

このステップでは、z/VSE ホストの接続仕様を作成します。 *VSESystem* のインスタンスを使用して、VSE ファイル・システム (この例では、ライブラリアン) にアクセスできるようにします。

```
VSEConnectionSpec spec;
try {
spec = new VSEConnectionSpec(InetAddress.getByName(ipAddr),
                             2893,userID,password);
}
catch (UnknownHostException e)
{
   System.out.println("Unknown host : " + e);
   return;
}
spec.setLogonMode(true);
VSESystem system = new VSESystem(spec);
system.setConnectionMode(true);
VSELibrarian libr = system.getVSELibrarian();
```

図 *78. VSE Java Beans* を介したライブラリアン・データ: 接続仕様の作成

ステップ 3: VSEResourceListener を作成する

このステップでは、*VSEResourceListener* を作成して、z/VSE ホストからオブジェクトを検索するごとに通知を行います。 次に、この z/VSE ホストから VSE ライブラリーのリストを取得します。 この *getLibraryList()* メソッドは、すべてのリソースを受信した後で (つまり、リスナーの *listEnded()* メソッドが呼び出されたときに) 制御を戻します。 *getLibraryList()* 要求が制御を戻した後で、VSE リソース・オブジェクトからリソース・リスナーを除去することをお勧めします。除去しないと、後でこのリスナーを別のオブジェクト (例えば、サブライブラリーで) で使用した場合に、直前のオブジェクトにも通知されます。

```
LibrListener ll = new LibrListener();
libr.addVSEResourceListener(ll);
libr.getLibraryList();
libr.removeVSEResourceListener(ll);
```

図 *79. VSE Java Beans* を介したライブラリアン・データ: *VSEResourceListener* の作成

ステップ 4: VSEResourceListener からライブラリーのリストを取得する

このステップでは、*VSEResourceListener* オブジェクトからライブラリーのリストを取得します。 PRD2 ライブラリー・インスタンスを取得した後でループを抜けます。 そのとき、変数 *myLib* は PRD2 ライブラリーを表すオブジェクトを指しています。

```
Vector vLibs = ll.getLibVector();
int numLibs = vLibs.size();
VSELibrary myLib;
for (int i=0;i<vLibs.size();i++)
{
   myLib = (VSELibrary)(vLibs.elementAt(i));
   if (myLib.getName().equals("PRD2"))
      break;
}
```

図 *80. VSE Java Beans* を介したライブラリアン・データ: ライブラリーのリストの取得

ステップ 5: サブライブラリーのリストを取得してカウントする

このステップでは、*VSEResourceListener* オブジェクトからサブライブラリーのリストを取得して、カウントします。

```
myLib.addVSEResourceListener(ll);
myLib.getSubLibraryList();
myLib.removeVSEResourceListener(ll);
int numSublibs = myLib.getNumberOfSublibs();
```

図 *81. VSE Java Beans* を介したライブラリアン・データ: サブライブラリーのリストの取得/カウント

ステップ 6: PRD2.CONFIG サブライブラリーのインスタンスを取得する

このステップでは、ベクターから PRD2.CONFIG サブライブラリーのインスタンスを取得します。 サブライブラリー名が CONFIG のときにループを抜けます。 そのとき、変数 *mySublib* は PRD2.CONFIG サブライブラリーを表すオブジェクトを指しています。

```
VSESubLibrary mySublib;
Vector vSublibs = ll.getSublibVector();
for (int i=0;i<vSublibs.size();i++)
{
   mySublib = (VSESubLibrary)(vSublibs.elementAt(i));
   if (mySublib.getName().equals("CONFIG"))
      break;
}
```

図 *82. VSE Java Beans* を介したライブラリアン・データ*:* サブライブラリーのインスタンスの取得

ステップ 7: PRD2.CONFIG サブライブラリー内のメンバーのリストを取得する

このステップでは、PRD2.CONFIG サブライブラリー内のメンバーのリストを取得します。これは、再度、*VSEResourceListener* のインスタンスを使用して行います。

```
mySublib.addVSEResourceListener(ll);
mySublib.getMemberList();
mySublib.removeVSEResourceListener(ll);
Vector vMembers = ll.getMemberVector();
int numMembers = mySublib.getNumberOfMembers();
```

図 *83. VSE Java Beans* を介したライブラリアン・データ*: PRD2.CONFIG* 内のメンバーのリストの取得

ステップ 8: 最初のメンバーのプロパティーを取得する

このステップでは、最初のメンバーのいくつかのプロパティーを取得します。各プロパティーについて、*get...()* メソッドを入れます。 変更可能なプロパティーについては、プロパティーを変更できる *set...()* メソッドも入れます。

```
VSELibraryMember myMember
if (vMembers.size() < 0)
{
   myMember = (VSELibraryMember)(vMembers.elementAt(0));
   int num = myMember.getNumberOfRecords();
   int len = myMember.getLogicalRecordLength();
   Calendar creation = myMember.getCreation();
   Calendar update = myMember.getLastUpdate();
   ...
}
```

図 *84. VSE Java Beans* を介したライブラリアン・データ*: PRD2.CONFIG* 内のメンバーのリストの取得

ステップ 9: メンバーをローカル・ディスクにダウンロードする

この最終ステップでは、ローカル「ハード」ディスクにメンバーをダウンロードします。 *download()* メソッドは、ダウンロードが完了した後で制御を戻します。 メンバーをダウンロードするには、次のようないくつかの方法があります。

- ローカル・ファイルにダウンロードできます。
- *DataOutputStream* にダウンロードできます。これはディスクにデータを書き込みません。 代わりに、ファイル内容を 1 行ずつストレージに取得できます。

```
        File localFile = new File(myMember.getName() + "." + myMember.getType());
        myMember.download(localFile);
    }
}
```

図 *85. VSE Java Beans* を介したライブラリアン・データ: ディスクへのメンバーのダウンロード

VSE Java Beans を使用した VSE/ICCF データへのアクセスの例

ここに記載する例には、*IccfApiExample* からとられたコード部分が含まれます。これについては、オンライン・ドキュメンテーションに詳しく記載しています。 例えば、次のようにします。

1. ホストを作成して、VSE/ICCF ライブラリーのリストを取得します。
2. VSE/ICCF ライブラリー 2 からメンバー C$QCNBAT (コンパイル・スケルトン) をダウンロードします。
3. メンバーのいくつかのプロパティーを表示します。

注:

1. VSE/ICCF へのアクセスは、読み取り専用です。
2. z/VSE ホスト接続の作成および再利用について詳しくは、VSE コネクター・クライアント で提供されるオンライン・ドキュメンテーションを参照してください (30 ページの『オンライン・ドキュメンテーション・オプションの使用』を参照)。

この例は、クラス名の指定およびメイン・メソッドのインプリメントから開始されます。

```
public class IccfApiExample
{
    public static void main(String argv[]) throws IOException, ResourceException
    {
```

以下に、この例の残りの主なステップを示します。

ステップ 1: IP アドレス、ユーザー ID、およびパスワードのプロンプト

ページ 167 ページの『ステップ 1: IP アドレス、ユーザー ID、パスワードのプロンプト』 を参照してください (コードは、VSE Java Beans を介した z/VSE ホストへの接続用のものと同じです)。

ステップ 2: VSE システムの接続仕様を作成する

このステップでは、z/VSE ホストの接続仕様を作成します。 *VSESystem* のインスタンスを使用して、VSE ファイル・システム (この例では、VSE/ICCF) にアクセスできるようにします。

VSE コネクター・サーバー は DTSUTIL ジョブを使用して DTSFILE から情報を取得するため、VSE/ICCF データへのアクセスは読み取り専用です。 また、システムの他のユーザーを中断させないために、書き込みアクセス中に DTSFILE を切断しないようにします。

```
VSEConnectionSpec spec;
try {
spec = new VSEConnectionSpec(InetAddress.getByName(ipAddr),
                             2893,userID,password);
}
catch (UnknownHostException e)
{
   System.out.println("Unknown host : " + e);
   return;
}
spec.setLogonMode(true);
VSESystem system = new VSESystem(spec);
system.setConnectionMode(true);
VSEIccf iccf = system.getVSEIccf();
```

図 *86. VSE Java Beans* を介した *VSE/ICCF* データ*:* 接続仕様の作成

ステップ 3: VSEResourceListener を作成する

このステップでは、*VSEResourceListener* を作成して、z/VSE ホストからオブジェクトを検索するごとに通知を行います。 次に、この z/VSE ホストから VSE/ICCF ライブラリーのリストを取得します。 この *getLibraryList()* メソッドは、すべてのリソースを受信した後で (つまり、リスナーの *listEnded()* メソッドが呼び出されたときに) 制御を戻します。 *getLibraryList()* 要求が制御を戻した後で、VSE リソース・オブジェクトからリソース・リスナーを除去することをお勧めします。

```
IccfListener il = new IccfListener();
iccf.addVSEResourceListener(il);
iccf.getLibraryList();
iccf.removeVSEResourceListener(il);
```

図 *87. VSE Java Beans* を介した *VSE/ICCF* データ*: VSEResourceListener* の作成

ステップ 4: VSEResourceListener から ICCF ライブラリーのリストを取得する

このステップでは、*VSEResourceListener* オブジェクトから VSE/ICCF ライブラリーのリストを取得します。 次に、VSE/ICCF ライブラリー 2 からコンパイル・スケルトンがダウンロードされます。

```
Vector vLibs = il.getLibVector();
VSEIccfLibrary myLib;
VSEIccfMember myMember;
for (int i=0;i<vLibs.size();i++)
{
   myLib = (VSEIccfLibrary)(vLibs.elementAt(i));
   if (myLib.getLibrary() == 2)
   {
      /* Get memberlist of ICCF lib 2 */
      myLib.addVSEResourceListener(il);
      myLib.getMemberList();
      myLib.removeVSEResourceListener(il);
      Vector vMembers = il.getMemberVector();
      for (int j=0;j<vMembers.size();j++)
      {
         myMember = (VSEIccfMember)(vMembers.elementAt(j));
         if (myMember.getName().equals("C$QCNBAT"))
         {
            myMember.download("C$QCNBAT.SKL");
         }
      }
   }
}
```

図 *88. VSE Java Beans* を介した *VSE/ICCF* データ: *ICCF* ライブラリーのリストの取得

ステップ 5: 特定の VSE/ICCF メンバーをダウンロードする

このステップでは、特定の VSE/ICCF メンバーを高速にダウンロードする方法を記載します。メンバーを検索してから、直接ダウンロードします。

```
iccf.addVSEResourceListener(il);
il.clear();
iccf.search(2, "C$QCNBAT", "*");
iccf.removeVSEResourceListener(il);
vMembers = il.getMemberVector();
if (vMembers.size() == 1)
{
   myMember = (VSEIccfMember)(vMembers.elementAt(0));
   myMember.download("C$QCNBAT.SKL2");
}
```

図 *89. VSE Java Beans* を介した *VSE/ICCF* データ: 特定のメンバーのダウンロード

ステップ 6: 特定の VSE/ICCF メンバーをダウンロードする (非常に高速な方法)

このステップでは、特定の VSE/ICCF メンバーを非常に 高速にダウンロードする方法を記載します。 メンバー・オブジェクトを作成して、ダウンロードしようとす

るだけです。 VSE/ICCF メンバーが z/VSE ホスト上に存在しない場合には、例外を受信します。

```
      try {
         myMember = new VSEIccfMember(system, 2, "C$QCNBAT");
         myMember.download("C$QCNBAT.SKL3");
      }
      catch (Exception e)
      {
         ...
      }
   }
}
```

図 *90. VSE Java Beans* を介した *VSE/ICCF* データ*:* 特定のメンバーのダウンロード *(*非常に高速*)*

VSE ナビゲーター・アプリケーションの使用

VSE ナビゲーターは、 159 ページの表 4 に示すすべての VSE Java Beans を実質的に使用する Java アプリケーションです。 現在使用可能な多くのファイル・マネージャーと類似の外観のグラフィカル・ユーザー・インターフェース (GUI) をインプリメントします。

VSE ナビゲーターのクライアント・パーツ は、*VSE* コネクター・サーバー と通信し、さまざまな機能を提供します。 例えば、以下の作業が可能です。

- VSE ファイル・システム (POWER、Librarian、ICCF、VSAM) へのアクセス。
- ジョブの作成およびサブミット (ICCF ライブラリー 2 に保管されているスケルトンを基にしたジョブの生成を含む)。
- VSE オペレーター・コンソールでの作業。
- ファイルの比較、および VSE ベース・ファイル・システムでの全文検索。
- VSAM レコードの対話式挿入および VSAM レコードの編集。
- 以下の表示。
  - 接続装置のプロパティー・ダイアログを含む VSE ハードウェア構成
  - VSE システム・アクティビティー (CPU 使用量など)
  - 現在の VSE サービス・レベル
  - システム・ラベル
  - システム・タスク
  - 使用中および空き VSAM スペース
  - マップおよびビューを使用した VSAM データ

以下に、VSE ナビゲーターで提供する GUI の例を 2 つ示します。

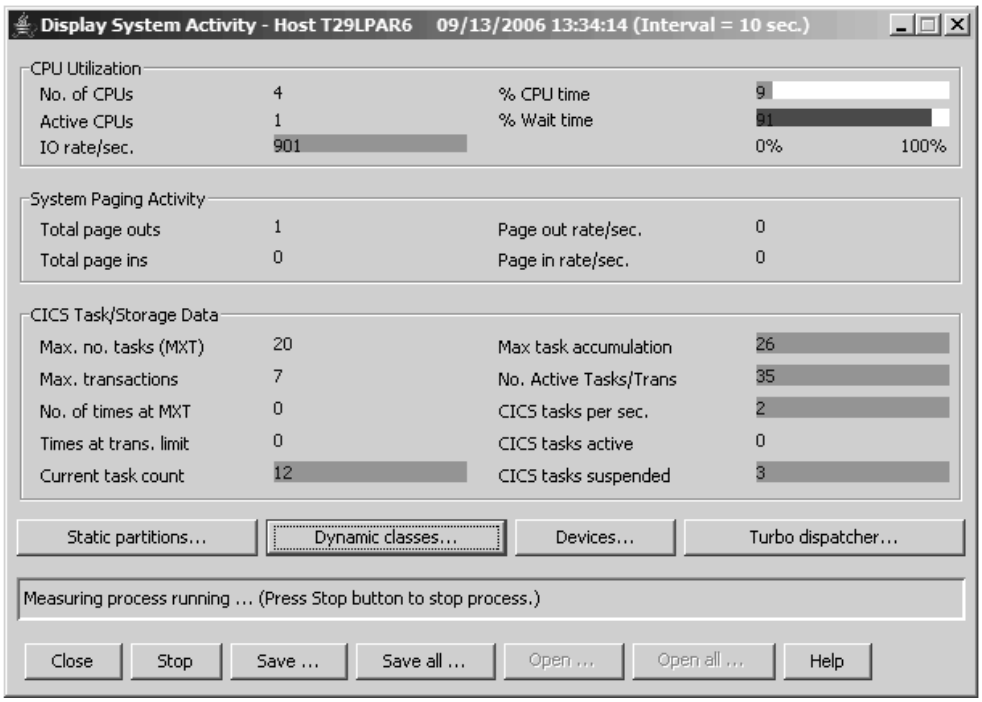


図 *91. VSE* ナビゲーターを使用したシステム・アクティビティーの表示

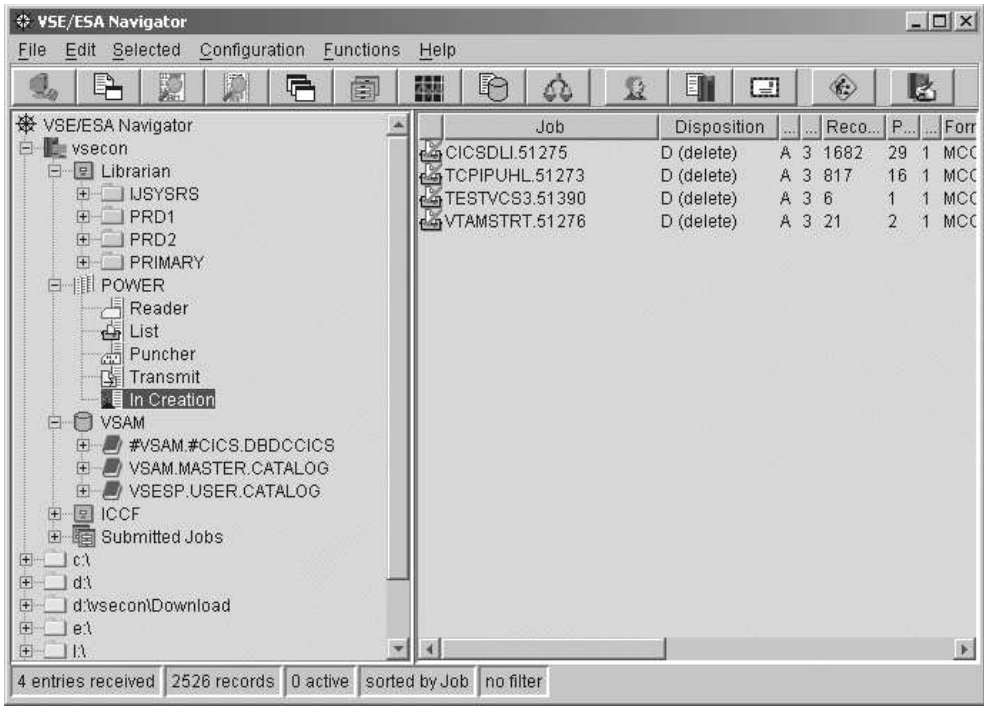


図 *92. VSE* ナビゲーターを使用した *VSAM* ファイルの表示

VSE ナビゲーターを使用する際の前提条件

VSE ナビゲーターをインストールして使用する前に、以下のインストール/構成が必要です。

- 28 ページの『VSE コネクター・クライアントのインストール』 で説明されているワークステーション上の VSE コネクター・クライアント。
- 31 ページの『VSE コネクター・サーバー の構成』 で説明されているホスト上の VSE コネクター・サーバー。
- TCP/IP for z/VSE.
- Java Development Kit (JDK) 1.5 またはそれ以降。JDK 1.5 以降をインストールしていない場合のインストール方法について詳しくは、 24 ページの『Java のインストールと構成』を参照してください。

注: 最新の APAR レベルについては、z/VSE ホーム・ページも確認してください。 URL は以下のとおりです (英語のみ)。

```
http://www.ibm.com/systems/z/os/zvse/
```

以前のバージョンからのマイグレーション

現在、VSE ナビゲーターの前のバージョンがインストールされている場合は、z/VSE の現行バージョンで稼働する最新バージョンにマイグレーションする必要があります。これを行うには、z/VSE の現行バージョンで稼働する VSE ナビゲーターを単にダウンロードして、それを既存のバージョンと置き換えます。

詳しくは、『VSE ナビゲーターのインストール』 (下記) を参照してください。

VSE ナビゲーターのインストール

VSE ナビゲーターは Java 対応プラットフォームにインストールします。

注: 開始する前に、『VSE ナビゲーターを使用する際の前提条件』に記載されたすべての前提条件を確実に満たすようにしてください。

VSE ナビゲーターは、1 つの Java インストール・クラス・ファイル**setup.jar** として提供されます VSE ナビゲーターをインストールするには、以下の作業を行う必要があります。

1. インターネットから VSE スクリプト・サーバーを入手します。Web ブラウザーを起動して次の URL にアクセスしてください (英語のみ)。

   ```
   http://www.ibm.com/systems/z/os/zvse/downloads/
   ```

   VSE ナビゲーターのセクションから、「**Details and Download (**詳細およびダウンロード**)**」を選択します。 ファイル **vsenavi*nnn*.zip** を選択して、VSE ナビゲーターのインストール先のディレクトリーに最新コードをダウンロードします。 注: *nnn* は現行 VSE バージョン (**vsenavi620.zip**) です。
2. ファイル **vsenavi*nnn*.zip** を unzip します。これには以下のファイルが入っています。
   - setup.jar (VSE ナビゲーター・コードを含む)
   - setup.bat または setup.cmd (Windows 用インストール・バッチ・ファイル)
   - setup.sh (Linux/Unix 用インストール・スクリプト)

3. (ファイルをダブルクリックして) オペレーティング・システム・プラットフォームに適用可能なバッチ・ファイルを開始します。
4. インストール・プロセスが開始され、さまざまなインストール・メニューによるガイドが表示されます。

VSE ナビゲーター・クライアントの開始

VSE ナビゲーター・クライアントを開始する方法は、以下のとおりです。

オペレーティング・システム
: 実行ファイルまたはシェル・スクリプト

Windows
: run.bat または run.cmd

Linux/Unix
: run.sh

Windows では、デスクトップ・オブジェクトが提供されます。これを使用して、VSE ナビゲーターの開始、オンライン・ドキュメンテーション (ファイル **NaviReference.html**) へのアクセスなどを行うことができます。

初めて VSE ナビゲーター・クライアントを開始するときには、以下のローカル項目を指定する必要があります。

- ユーティリティー (例えば、ヘルプ・テキストの表示に使用する Web ブラウザー)
- ディレクトリー

独自の VSE ナビゲーター・プラグインの追加

VSE ナビゲーターには、VSE ナビゲーター・クライアントに独自の「プラグイン」機能を追加できる Java プログラミング・インターフェースがあります。 VSE ナビゲーター・プラグインを作成するには、以下の作業を行う必要があります。

1. Java インターフェースのメソッドをインプリメントします。
2. クラス・ファイルを VSE ナビゲーターのプラグイン・ディレクトリーにコピーします。さらに、プラグイン・ディレクトリー内に独自のクラス用の新規ディレクトリーを作成することもできます。
3. VSE ナビゲーター・クライアントを再起動します。 ここで、プラグインが動的にロードされ、VSE ナビゲーターのツールバーおよびメニューからアクセスできるようになります。 オンラインのプログラミング解説書は HTML ベースで、VSE ナビゲーターのインストール先のディレクトリーに保管される **NaviReference.html** からアクセスできます。

VSE ナビゲーターで SSL/TLS を構成するために提供されている GUI を以下に示します。

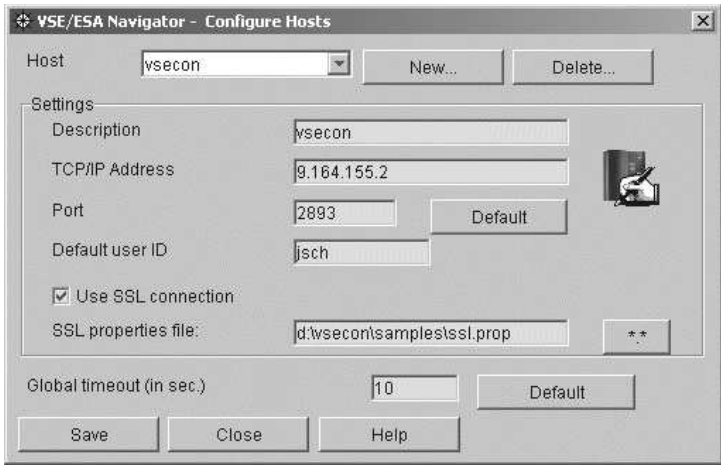


図 *93. VSE* ナビゲーターのホストの構成

次に、VSE ナビゲーターで SSL/TLS を使用する際の GUI を示します。

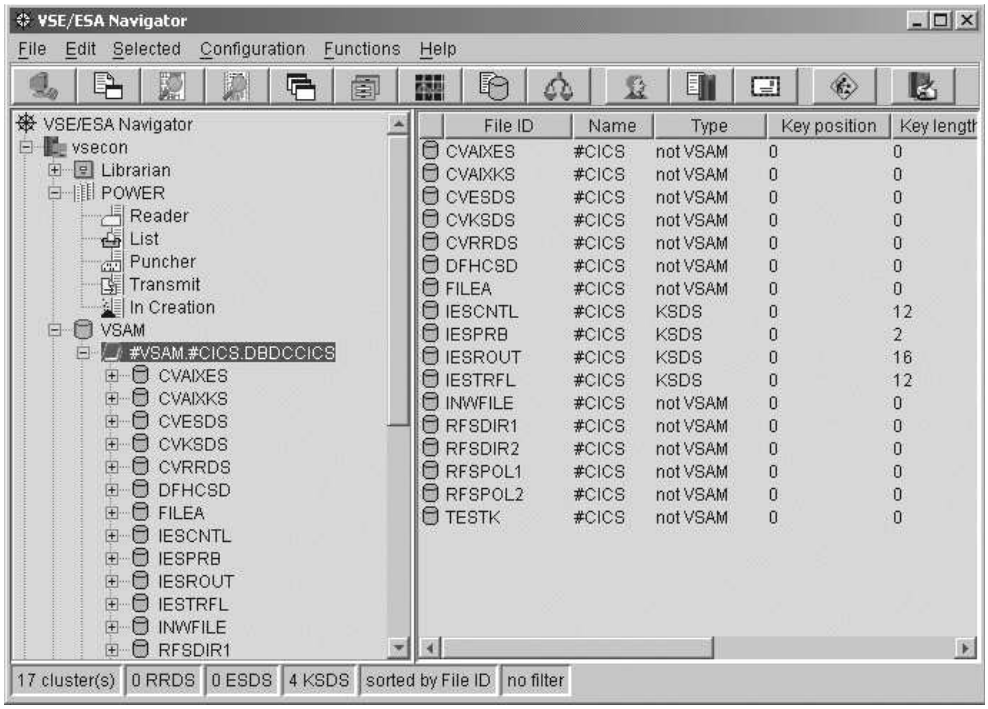


図 *94. VSE* ナビゲーターを使用した *CICS* データへのアクセス

VSE 正常性チェッカー・アプリケーションの使用

VSE 正常性チェッカーは、VSE システムからパフォーマンスに関するデータを検索したり、データの表示および分析を行えるシステム診断ユーティリティーです。

VSE 正常性チェッカーのクライアント・パーツ は、*VSE* コネクター・サーバー と通信し、さまざまな機能を提供します。 これによって、低下したパフォーマンスや

システム障害を起こす可能性のある VSE 構成問題を検出できます。 収集されたデータは、XML 形式でエクスポートおよびインポートできます。

VSE データは、以下によって取り出されます 。
- コンソール・コマンドの送信。
- VSE/POWER ジョブのサブミット。
- VSE ライブラリアン・メンバーのダウンロード。
- CICS トランザクションの呼び出し。

ベンダー・ツールに依存しません。

VSE 出力データは、ワークステーションに転送され、解析されて、さらなる分析のために GUI に表示されます。

注:

1. VSE 正常性チェッカーでは、VSE システムのシステム・パラメーターを変更できないことを理解しておくことは重要です。 VSE 正常性チェッカーが実行するすべてのアクションは、読み取り専用です。
2. VSE システムからデータを取得する処理は最大数分かかることもありますが、これは、ただの経過時間 です。結果として、VSE 正常性チェッカーによって引き起こされる CPU のオーバーヘッドはごくわずかです。

次に、VSE 正常性チェッカーでの GUI を示します。

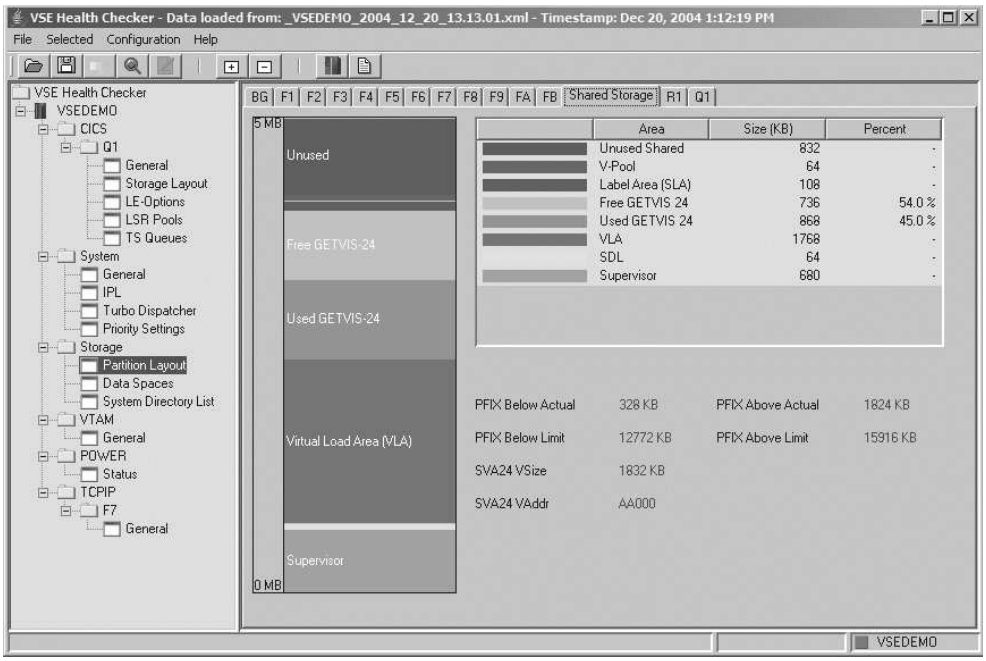


図 *95. VSE* 正常性チェッカーが提供するグラフィカル・ユーザー・インターフェース

VSE 正常性チェッカーを使用する際の前提条件

VSE 正常性チェッカーをインストールして使用する前に、以下のインストール/構成が必要です。

- 28 ページの『VSE コネクター・クライアントのインストール』 で説明されているワークステーション上の VSE コネクター・クライアント。
- 31 ページの『VSE コネクター・サーバー の構成』 で説明されているホスト上の VSE コネクター・サーバー。
- TCP/IP for z/VSE.
- Java Development Kit (JDK) 1.5 またはそれ以降。JDK 1.5 以降をインストールしていない場合のインストール方法について詳しくは、 24 ページの『Java のインストールと構成』を参照してください。

注: 最新の APAR レベルについては、z/VSE ホーム・ページも確認してください。 URL は以下のとおりです (英語のみ)。

```
http://www.ibm.com/systems/z/os/zvse/
```

以前のバージョンからのマイグレーション

現在、VSE 正常性チェッカーの前のバージョンがインストールされている場合は、z/VSE の現行バージョンで稼働する最新バージョンにマイグレーションする必要があります。これを行うには、z/VSE の現行バージョンで稼働する VSE 正常性チェッカーを単にダウンロードして、それを既存のバージョンと置き換えます。

詳しくは、『VSE 正常性チェッカーのインストール』 (下記) を参照してください。

VSE 正常性チェッカーのインストール

VSE 正常性チェッカーは Java 対応プラットフォームにインストールします。

注: 開始する前に、 196 ページの『VSE 正常性チェッカーを使用する際の前提条件』に記載されたすべての前提条件を確実に満たすようにしてください。

VSE 正常性チェッカーは、1 つの Java インストール・クラス・ファイル**setup.jar** として提供されます VSE 正常性チェッカーをインストールするには、以下の作業を行う必要があります。

1. インターネットから VSE 正常性チェッカーを入手します。Web ブラウザーを起動して次の URL にアクセスしてください (英語のみ)。

   ```
   http://www.ibm.com/systems/z/os/zvse/downloads/
   ```

   VSE 正常性チェッカーのセクションから、ファイル **vsehealth*nnn*.zip** を選択して、VSE 正常性チェッカーをインストールするディレクトリーに最新コードをダウンロードします。 注: *nnn* は現行 VSE バージョン (**vsehealth620.zip**) です。
2. ファイル **vsehealth*nnn*.zip** を unzip します。これには以下のファイルが入っています。
   - setup.jar (VSE 正常性チェッカー・コードを含む)
   - setup.bat または setup.cmd (Windows 用インストール・バッチ・ファイル)
   - setup.sh (Linux/Unix 用インストール・スクリプト)
3. (ファイルをダブルクリックして) オペレーティング・システム・プラットフォームに適用可能なバッチ・ファイルを開始します。

4. インストール・プロセスが開始され、さまざまなインストール・メニューによるガイドが表示されます。

VSE 正常性チェッカー・クライアントの開始

VSE 正常性チェッカー・クライアントを開始する方法は、以下のとおりです。

オペレーティング・システム
: 実行ファイルまたはシェル・スクリプト

Windows
: run.bat または run.cmd

Linux/Unix
: run.sh

Windows では、デスクトップ・オブジェクトが提供されます。これを使用して、VSE 正常性チェッカーの開始、オンライン・ドキュメンテーション (ファイル **vsehealth.html**) へのアクセスなどを行うことができます。

初めて VSE 正常性チェッカー・クライアントを開始するときには、ヘルプ・テキストを表示するために使用するローカルの Web ブラウザーを指定する必要があります。 VSE 正常性チェッカーを使用して開始する方法については、VSE 正常性チェッカーで提供するオンライン・ドキュメンテーションの「クイック・スタート」トピックを参照してください。

第 16 章 JDBC を使用した VSAM データへのアクセス

このトピックでは、*Java Database Connectivity* (JDBC) ドライバーを使用して、リレーショナル・データベース照会をセットアップして実行し、VSAM データに対する要求を更新する方法について説明します。これを行うために、Java ベース・コネクターには、JDBC で使用するさまざまなクラスが用意されています。

したがって、ユーザーは、VSE Java Beans インターフェースに照らしてコーディングする代わりに SQL 構成を使用して VSAM データにアクセスできます。 現在、SQL 構文のサブセットのみがサポートされています (『JDBC でサポートされている SQL ステートメント』を参照)。 ただし、これは、Db2 ストアード・プロシージャー内から VSAM SQL コール・レベル・インターフェース (CLI) で使用できるものと同じサブセットです (詳細については、 451 ページの『Db2 ストアード・プロシージャーによる VSAM データへのアクセス』を参照)。

VSE Java Beans インターフェースの代わりに JDBC ドライバーを使用すると以下の利点があります。

- JDBC および SQL は、リレーショナル・データベース内のデータにアクセスするための標準インターフェースです。
- *IBM Visual Age for Java* などの多くの製品が JDBC インターフェースをサポートしています。
- *IBM Visual Age for Java* プログラムを使用して作成されたアプリケーションに VSAM アクセスを統合できます (VSE 固有のコードを組み込む必要はありません)。

非リレーショナル VSAM データをリレーショナル構造にマップする方法の詳細については、 117 ページの『第 12 章 リレーショナル構造への VSE/VSAM データのマッピング』を参照してください。

このトピックには以下が含まれます。

JDBC でサポートされている SQL ステートメント

以下の SQL ステートメントは JDBC によってサポートされています。

表 *5. JDBC* によってサポートされている *SQL* ステートメント

SQL ステートメント
<insert statement> = "INSERT" ["INTO"] <table name> [<column name list>] "VALUES" "(" <insert values list> ")" ("," "(" <insert values list> ")")* | <select statement>

表 5. *JDBC* によってサポートされている *SQL* ステートメント (続き)

SQL ステートメント	
<table name> =	<IDENTIFIER> "¥" <IDENTIFIER> "¥" <IDENTIFIER> ["¥" <IDENTIFIER>]
<column name list> =	"(" <column name> ("," <column name>)* ")"
<column name> =	<IDENTIFIER>
<insert values list>=	"(" <insert value> ("," <insert values>)* ")"
<insert value> =	<NUMBER> | <CHAR LITERAL> | <PREPARED EXPR>
<update statement> =	"UPDATE" <table name> "SET" <column values> [<where clause>]
<column values> =	<column name> "=" <updated value> ("," <column name> "=" <updated value>)*
<updated value> =	<NUMBER> | <CHAR LITERAL> | <PREPARED EXPR>
<delete statement> =	"DELETE" ["FROM"] <table name> [<where clause>]
<query statement> =	<select statement>
<select statement> =	<select no order> [<order by clause>]
<select no order> =	"SELECT" ["ALL" | "DISTINCT"] <select list> <from clause> [<where clause>]
<select list> =	"*" | <select item> ("," <select item>)*
<select item> =	(<IDENTIFIER> "¥" [<IDENTIFIER> "¥" <IDENTIFIER> "¥" [<IDENTIFIER> "¥"]] "*") | (<sl element> ["AS"] [<IDENTIFIER> <QUOTED IDENTIFIER>])
<sl element> =	<sl mult expr> (("+" | "-") <sl mult expr>)*
<sl mult expr> =	<sl primary expr> (("*" | "/") <sl primary expr>)*

表 5. *JDBC* によってサポートされている *SQL* ステートメント (続き)

SQL ステートメント	
<sl primary expr> =	<table column> | <NUMBER> | <CHAR LITERAL> | <PREPARED EXPR> | "(" <sl element> ")"
<from clause> =	"FROM" <from item> [("," <from item>)+ | (<join clause>)+]
<from item> =	("(" <subquery> ")" | <table name>) ["AS"] [<IDENTIFIER>]
<join clause> =	<join type> <from item> [<join specification>]
<join type> =	"INNER" "JOIN" | "NATURAL" ["INNER"] "JOIN" | "LEFT" ["OUTER"] "JOIN" | "RIGHT" ["OUTER"] "JOIN" | "FULL" ["OUTER"] "JOIN" | "JOIN"
<join specification> =	"USING" "(" <IDENTIFIER> ("," <IDENTIFIER>)* ")" | "ON" <table column> "=" <table column>
<where clause> =	"WHERE" <where expr>
<where expr> =	<where and expr> ("OR" <where and expr>)*
<where and expr> =	<where rel expr> ("AND" <where rel expr>)*
<where rel expr> =	["NOT"] (<where primary expr> <relop> <where primary expr>) | "(" <where expr> ")"
<where primary expr> =	<table column> | <NUMBER> | <CHAR_LITERAL> | <PREPARED EXPR>
<order by clause> =	"ORDER" "BY" <table column> ["ASC" | "DESC"] ("," <table column> ["ASC" | "DESC"])*

表 5. *JDBC* によってサポートされている *SQL* ステートメント (続き)

SQL ステートメント	
<relop> =	"=" | "<>" | "<" | "<=" | ">" | ">="
<subquery> =	<select no order>
<table column> =	<IDENTIFIER> ["¥" <IDENTIFIER> ["¥" <IDENTIFIER> "¥" <IDENTIFIER> ["¥" <IDENTIFIER>]]]

リレーショナルおよび VSE Java Beans の用語

次の表には、リレーショナル SQL で使用される用語がどのように非リレーショナル VSE Java Beans に対応するかを理解するのに役立つガイドラインが記載されています。

表 6. リレーショナル用語および対応する *VSE* 用語

SQL 用語	**VSE Java Beans** 用語
SQL テーブル	VSEVsamMap (およびオプションとして VSEVsamView) とともに使用される VSEVSAMCatalog および VSEVsamCluster。
データベース行	列名およびデータ・フィールド・プロパティーを記述する VSEVsamMap とともに使用される VSEVsamRecord。
列名	列の以下の要素を記述する VSEVsamField: 名前 レコード内のオフセット 長さ データ・タイプ (ストリング、整数など)

テーブル名の指定

次の例で、JDBC を使用してテーブル名を指定する方法について説明します。 クラスター MY.TEST.CLUSTER が、カタログ MY.USER.CATALOG にあるとします。 このクラスターに MY.TEST.MAP という名前のマップが定義されているとします。

この場合、VSAM JDBC ドライバーで使用されるテーブル名は次のようになります。

```
MY.USER.CATALOG¥MY.TEST.CLUSTER¥MY.TEST.MAP
```

JDBC を使用した VSAM データへのアクセスの例

このトピックでは、JDBC ドライバーを使用して VSAM データにアクセスする例について説明します。

この例では、 256 ページの『サーブレットをインプリメントする方法の例』で説明されている *FlightOrderingServlet* とほとんど同じ処理を実行します。この例では、*FlightOrderingServlet* と同じ VSAM クラスターに対して操作を行いますが、これらのクラスターを作成することはしません。 したがって、これらのクラスターをまだ作成していない (作成しても、データを充てんしていない) 場合は、その方法の詳細について、 257 ページの『サンプル用の VSAM クラスターの作成』を参照してください。

この例のサーブレットは、オンライン・ドキュメンテーション (詳しくは 30 ページの『オンライン・ドキュメンテーション・オプションの使用』を参照) で提供されている Java ソース・ファイル **JdbcFlightOrderingServlet.Java** 内にインプリメントされています。

この例は、次のように、クラス名の指定で始まります。

```
public class JdbcExample
{
```

以下に、この例の残りの主なステップを示します。

ステップ 1: ローカル変数を定義する

このステップでは、ローカル変数が定義されます。 この例では、以下の VSAM ファイルがすでに定義され、サンプル・データで充てんされていると想定しています。

- FLIGHT.ORDERING.FLIGHTS (KSDS)
- FLIGHTS.ORDERING.ORDERS (RRDS)

(詳しくは 257 ページの『サンプル用の VSAM クラスターの作成』を参照)。

```
String vsamCatalog    = "VSESP.USER.CATALOG";
String flightsCluster = "FLIGHT.ORDERING.FLIGHTS";
String ordersCluster  = "FLIGHT.ORDERING.ORDERS";
String flightsMapName = "FLIGHTS_MAP";
String ordersMapName  = "ORDERS_MAP";

public static void main(String argv[]) throws IOException
{
   try
   {
```

図 *96. JDBC* を介した *VSAM* データ*:* ローカル変数の定義

ステップ 2: IP アドレス、ユーザー ID、およびパスワードの入力のためのプロンプトを出す

このステップでは、z/VSE ホスト・システムの IP アドレス、ユーザー ID、およびパスワードの入力のために、ユーザーにプロンプトが出されます。

```
BufferedReader r = new BufferedReader(
                     new InputStreamReader(System.in));
System.out.println("Please enter your VSE IP address:");
String ipAddr = r.readLine();
System.out.println("Please enter your VSE user ID:");
String userID = r.readLine();
System.out.println("Please enter password:");
String password = r.readLine();
```

図 *97. JDBC* を介した *VSAM* データ*: IP* アドレス、ユーザー *ID*、パスワードの入力のためのプロンプト

ステップ 3: z/VSE ホストへの接続を確立する

このステップで、VSAM JDBC ドライバーのインスタンスが作成され、z/VSE ホスト接続が確立されます。

```
   java.sql.Connection jdbcCon;
   java.sql.Driver jdbcDriver = (java.sql.Driver) Class.forName(
           "com.ibm.vse.jdbc.VsamJdbcDriver").newInstance();

   // Build the URL to use to connect
   String url = "jdbc:vsam:"+ipAddr;

   // Assign properties for the driver
   java.util.Properties prop = new java.util.Properties();
   prop.put("port", 2893);
   prop.put("user", userID);
   prop.put("password", password);

   // Connect to the driver
   jdbcCon = DriverManager.getConnection(url, prop);
}
catch (Throwable t)
{

⋮
}
```

図 *98. JDBC* を介した *VSAM* データ*:* ホスト接続の確立

ステップ 4: データベース内の行のリストを表示する

このステップで、SQL ステートメントが作成され、データベース行 (これは VSAM レコード) のリストが表示されます。

```
try
{
   // Get a statement
   java.sql.Statement stmt = jdbcCon.createStatement();

   // Execute the query ...
   java.sql.ResultSet rs = stmt.executeQuery(
      "SELECT * FROM "+vsamCatalog+"¥¥"+flightsCluster+"¥¥"+flightsMapName);
```

図 *99. JDBC* を介した *VSAM* データ*:* データベース行の表示

ステップ 5: JDBC から戻された結果セットを処理する

```
    while (rs.next())
    {
       int flightNumber   = rs.getInt("FLIGHT_NUMBER");
       String start       = rs.getString("START");
       String destination = rs.getString("DESTINATION");
       String departure   = rs.getString("DEPARTURE");
       String arrival     = rs.getString("ARRIVAL");
       int price          = rs.getInt("PRICE");
       String airline     = rs.getString("AIRLINE");
    }
    rs.close();
    stmt.close();
 }
 catch (SQLException t)
 {

⋮
 }
```

図 *100. JDBC* を介した *VSAM* データ*:* 結果セットの処理

ステップ 6: 新規レコードを追加する

このステップで、新規レコードが VSAM データベースに追加されます。

```
       try {
          java.sql.PreparedStatement pstmt = jdbcCon.prepareStatement(
             "INSERT INTO "+vsamCatalog+"¥¥"+flightsCluster+"¥¥"+flightsMapName+
             " (FLIGHT_NUMBER,START,DESTINATION,DEPARTURE,ARRIVAL,PRICE,AIRLINE)"+
             " VALUES(?,?,?,?,?,?,?)");

          pstmt.setInt(1, 398);
          pstmt.setString(2,"Honolulu");
          pstmt.setString(3,"Bankok");
          pstmt.setString(4,"07:30");
          pstmt.setString(5,"22:45");
          pstmt.setInt(6, 1500);
          pstmt.setString(7,"VSE Airtours");

          // Execute the query
          int num = pstmt.executeUpdate();
          pstmt.close();
       }
       catch (SQLException t)
       {

⋮
       }
    }
 }
```

図 *101. JDBC* を介した *VSAM* データ*:* 新規レコードの追加

第 17 章 Java アプレットによるデータへのアクセス

Java アプレットは、Web ブラウザーの Java 仮想マシン内で実行される Java プログラムです。 アプレットを使用すると、主に以下の利点があります。

- Java 使用可能の Web ブラウザーのいずれを使用してもアプレットにアクセスできます。
- Web クライアントに、さらにプログラムをインストールする必要がありません。
- アプレットを使用することにより、使いやすいユーザー・インターフェース (UI) を作成できます。

アプレットは、定義上はセキュア です。 アプレットは以下にはアクセスできません。

- クライアント・ワークステーションのリソース
- クライアント・ワークステーションのメモリー
- ネットワーク。

次に、アプレット・タグの例を示します。

```
<applet code="myapplet" archive="applets.jar">
</applet>
```

「applet code」タグはアプレットのメイン Java クラスの名前を指定し、オプションの「archive」タグは、クラス・ライブラリー (およびその他のすべての必要なクラス) が入る 1 つ以上のアーカイブを指定します。 アプレットに属するクラスと JAR ファイルは、物理/論理中間層のいずれのディレクトリーにも保管できます。

このトピックに含まれるのは次のとおりです。

- 『2 層環境内でのアプレットの使用方法』
- 209 ページの『3 層環境内でのアプレットの使用方法』
- 211 ページの『VSEAppletServer の使用方法』
- 212 ページの『アプレットを使用するときの欠点および制限』
- 212 ページの『データ・マッピング・アプレットのサンプルの実行』
- 222 ページの『サンプル VSAM アプレットの実行』
- 237 ページの『サンプル DL/I アプレットの実行』

2 層環境内でのアプレットの使用方法

208 ページの図 102 に、z/VSE 2 層環境 内でのアプレットの使用方法を示します。

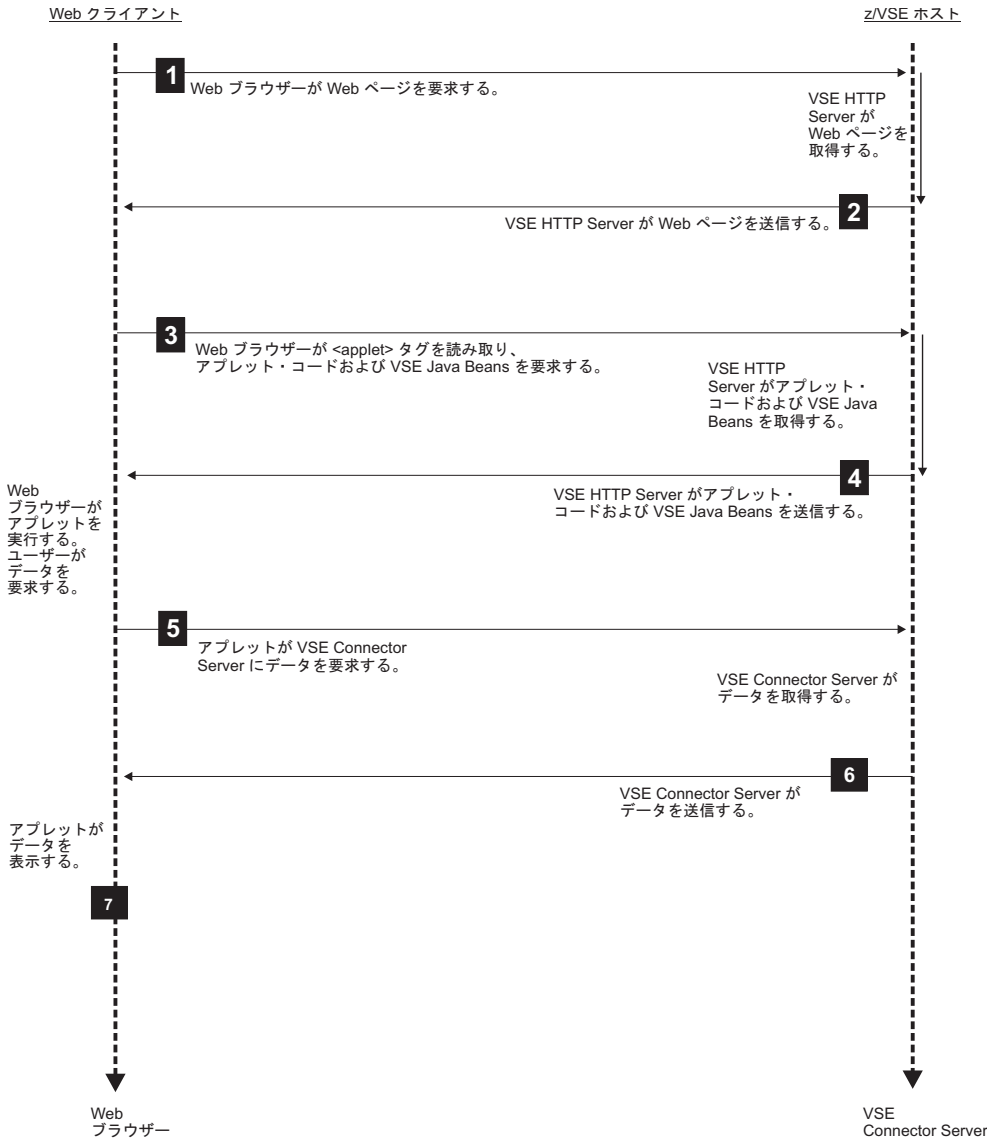


図 *102. z/VSE 2* 層環境内でのアプレットの使用方法

Web クライアントと z/VSE ホストとの間のデータの送受信には HTTP セッションが使用されます。

以下の各リスト項目の番号は、図 102 に示されているステップを説明しています。

1 クライアントの Web ブラウザーが、z/VSE ホストで実行されている VSE HTTP Server に HTML ページを要求します。

2 VSE HTTP Server は、Web ページをクライアントの Web ブラウザーに送信します。

3 クライアントの Web ブラウザーは <applet> タグを読み取り、z/VSE ホスト上の VSE HTTP Server からのアプレット・コードを要求します。アプレット・コードは 1 つ以上の JAR ファイルに保管されています。 また、Web ブラウザーは、VSE HTTP Server に VSE Java Beans クラス・ライブラリー (**VSEConnector.jar**) も要求します。

4 VSE HTTP Server は、VSE Java Beans クラス・ライブラリーとともに、アプレット・コードをクライアントの Web ブラウザーに送信します。

5 クライアントの Web ブラウザーがアプレットを実行します。 アプレットは、VSE Java Beans クラス・ライブラリー (**VSEConnector.jar**) を使用して、VSE コネクター・サーバー への接続を構築します。 エンド・ユーザーは、z/VSE ホストに保管されているデータを要求します。 2 層環境内におけるアプレットの場合、データは、VSE/POWER、VSE/VSAM、VSE/ICCF、またはライブラリアン・データのいずれでもかまいません。

注: アプレットは、2 層環境内にある Db2、DL/I、または CICS データを取得することはできません。これは、VSE コネクター・サーバー がこれらのシステムにアクセスすることができないからです。 さらに、2 層環境内では、>WebSphere MQ サーバーの使用はできません。

6 VSE コネクター・サーバー は、「ネイティブ」呼び出し (標準アクセス方式を使用する) を使用して必要なデータを取得し、次に、そのデータを TCP/IP を介してクライアントの Web ブラウザーに送信します。

7 クライアントの Web ブラウザーがアプレットの 2 回目の実行を行い、要求データと一緒に Web ページを表示します。

アプレットは主に 2 層環境内で使用されますが、*3* 層環境 内でアプレットを使用することも可能です。ただし、これには、物理/論理中間層上に、クライアントと z/VSE ホストとの間でゲートウェイとして機能する「ルーター」をインプリメントする必要があります。 詳細については、 211 ページの『VSEAppletServer の使用方法』を参照してください。

3 層環境内でのアプレットの使用方法

210 ページの図 103 に、z/VSE *3* 層環境 内でのアプレットの使用方法を示します。 以下の各リスト項目の番号は、この図に示されているステップを説明しています。

1 クライアントの Web ブラウザーが、物理/論理中間層で実行されている IBM HTTP Server にある HTML ページを要求します。

2 IBM HTTP Server は、Web ページをクライアントの Web ブラウザーに送信します。

3 クライアントの Web ブラウザーは <applet> タグを読み取り、物理/論理中間層上で実行されている IBM HTTP Server にアプレット・コードを要求します。 アプレットに属するクラスと JAR ファイルは、物理/論理中間層のいずれのディレクトリーにも保管できます。また、Web ブラウザーは、IBM HTTP Server に VSE Java Beans クラス・ライブラリー (**VSEConnector.jar**) も要求します。

4 IBM HTTP Server は、VSE Java Beans クラス・ライブラリーとともに、アプレット・コードをクライアントの Web ブラウザーに送信します。

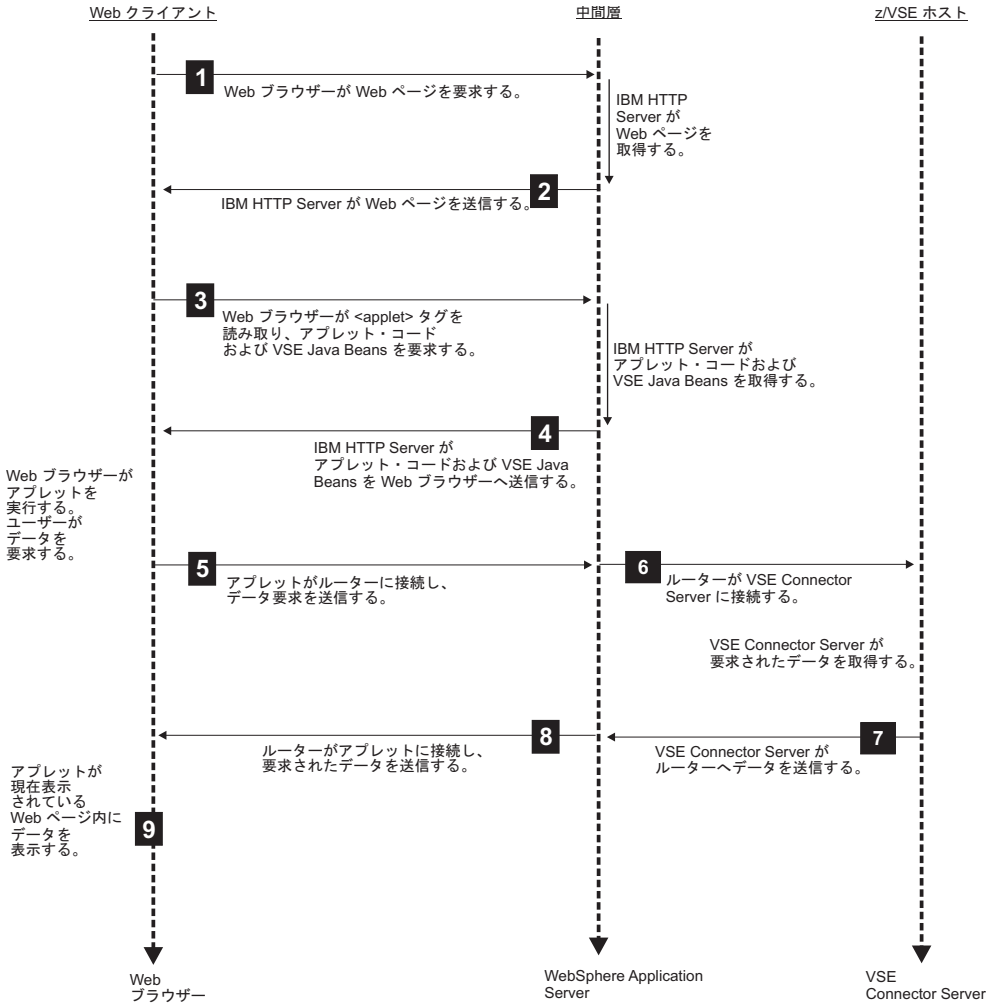


図 *103. z/VSE 3* 層環境内でのアプレットの使用方法

5 クライアントの Web ブラウザーはアプレットを実行し、エンド・ユーザーは z/VSE ホストに保管されているデータを要求します。 アプレットは物理/論理中間層上のルーターへの接続を行い、データに対する要求をそのルーターに送信します。 ルーターは、物理/論理中間層上で実行されている *VSEAppletServer* (211 ページの『VSEAppletServer の使用方法』に説明があります) です。

6 ルーターは、z/VSE ホスト上で実行されている VSE コネクター・サーバー に接続を行い、そこに、データに対する要求を転送します。

7 VSE コネクター・サーバー は、「ネイティブ」呼び出し (標準アクセス方式) を使用してデータをリトリーブし、そのデータを、物理/論理中間層上で実行されている *VSEAppletServer* ルーターに送信して戻します (TCP/IP を使用)。

注:

1. VSE コネクター・サーバー は、VSE/VSAM、VSE/POWER、VSE/ICCF、またはライブラリアン・データにアクセスするために使用できます。

2. z/VSE ホストに保管されている VSAM データにアクセスする代替メソッドは、z/VSE ホスト上の VSAM ファイル・システムと直接通信を行っている物理/論理中間層上の Db2 ストアード・プロシージャーを使用する方法です。このことについては、 451 ページの『Db2 ストアード・プロシージャーによる VSAM データへのアクセス』に説明があります。

8 ルーターは、クライアントの Web ブラウザー内で実行されているアプレットに接続を行い、そのアプレットにデータを送信します (これも TCP/IP を使用)。

9 クライアントの Web ブラウザー内で実行されているアプレットは、そのデータを、現在表示されている Web ページ内に表示します。

Web クライアントと物理/論理中間層との間のデータの送受信には HTTP セッションが使用されます。 物理/論理中間層と z/VSE ホストとの間のデータの送受信には接続セッションが使用されます。

VSEAppletServer の使用方法

アプレットには、その性質上、以下のことを含む多くの制限があります。

- アプレットが新規ネットワーク接続をオープンできるのは、アプレットがダウンロードされた元のプラットフォームに対してだけです。 3 層環境では、これは物理/論理中間層になります。
- アプレットがアクセスできるのは、アプレットがダウンロードされた元のプラットフォームのファイル・システムだけです。 3 層環境では、これは、物理/論理中間層に保管されているファイル・システムです。

これらの問題を避け、z/VSE ホストで実行されている VSE コネクター・サーバーからデータをアプレットが取得できるようにするために、単純なルーター (*VSEAppletServer*) が用意されています。 アプレットは、このルーターに接続するだけで済みます。 次に、上記の制限を持たないルーターが VSE コネクター・サーバー に接続を行って VSE ベースのデータを取得し、そのデータをアプレットに戻します。

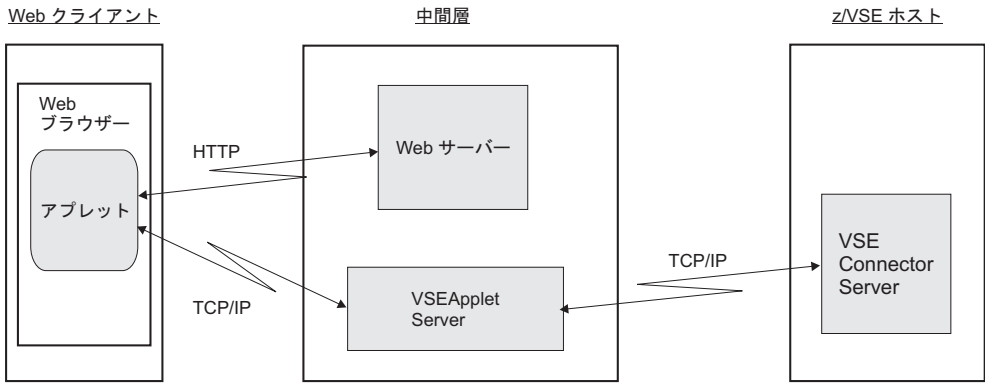


図 *104. 3* 層環境内での *VSEApplet Server* の使用方法

アプレットを使用するときの欠点および制限

以下は、z/VSE 環境でアプレットを使用するときの欠点および制限です。

- アプレットは Web ブラウザー内で実行しなければならないので、3 層環境という非常に高速の物理/論理中間層マシンの利点は得られません。代わりに、処理速度は、個々の Web ブラウザーのリソースおよびネットワーク速度と帯域幅によって決まります。
- WebSphere MQ Client for Java は、アプレットのコネクターとして使用できません。WebSphere MQ Server for z/VSE は、別の WebSphere MQ Server とのみ通信でき、WebSphere MQ Client for Java とは直接通信できないためです。
- **archive** タグは Netscape あるいは Microsoft Internet Explorer 4 (またはこれ以降) でのみ使用でき、Microsoft Internet Explorer 3 では使用できません。サポートされているアプレットのパラメーターの詳細については、ご使用の Web ブラウザーの資料を参照してください。
- VSE ライブラリー・システムには、クラス・ファイルおよび jar ファイルをバイナリー として保管しなければなりません。 ファイル・フォーマットを検査するには、ライブラリアンの **LD** コマンドを使用できます。
- アプレット を作成するときは、アプレット・コード内にユーザー ID およびパスワードを決して「ハードコーディング」 してはなりません。アプレットが Web ブラウザーにダウンロードされて Web ブラウザーのキャッシュに保管されると、この情報が無許可ユーザーによって表示される可能性があります。

データ・マッピング・アプレットのサンプルの実行

このトピックの説明は、VSE コネクター・クライアント で提供されている *VSAM* データ・マッピング・アプレット (単にデータ・マッピング・アプレット とも呼ばれる) というサンプル・アプレットを基にしています。

VSE コネクター・クライアント で同様に提供されているこのほかのアプレットの例には、次のものがあります。

- *VsamSpaceUsage* (使用されている VSAM スペースおよび空き VSAM スペースを表示する)。
- *DB2ConnectorJDBCApplet* (Db2 ストアード・プロシージャーを呼び出し、VSAMSQL コール・レベル・インターフェース (*CLI* と省略される) を使用して VSAM データにアクセスする)。 詳細については、 222 ページの『サンプル VSAM アプレットの実行』を参照してください。
- *VSAM* アプレット (222 ページの『サンプル VSAM アプレットの実行』 ページに説明があります)。
- *DL/I* アプレット (237 ページの『サンプル DL/I アプレットの実行』 ページに説明があります)。

データ・マッピング・アプレットの説明

データ・マッピング・アプレットは、特定の VSAM ファイルのマップとビューをユーザーが作成し保守できるようにする Java アプレットを記述します。 このアプ

レットは、VSAM データ自体を表示または変更する方法を説明するものではありません。 VSAM データを表示または変更するには、以下のどちらかを使用する必要があります。

- サンプルの VSAM アプレット (256 ページの『サーブレットをインプリメントする方法の例』に説明があります) と同じ機能を持つアプレット。
- サーブレット (256 ページの『サーブレットをインプリメントする方法の例』に説明があります)。

注: データ・マッピング・アプレットは、オペレーティング・システムと Web ブラウザーの特定の組み合わせによっては実行できない場合があります。

アプレットは、以下の一般的な処理を実行します。

1. アプレットは一部のウィンドウ・コントロールを表示し、これを使用することにより、ユーザーは z/VSE ホストの IP アドレス、VSE ユーザー ID、およびパスワードを入力できます。
2. アプレットは VSE コネクター・サーバー に接続を行い、VSAM カタログのリストをリトリーブします。
3. カタログ・リストの中の 1 つの項目をダブルクリックすることにより、このカタログ内のすべてのクラスターを示すリストが表示されます。
4. 1 つのクラスターをダブルクリックすると、このクラスターに定義されているすべてのマップを示すリストが表示されます。
5. 1 つのマップをダブルクリックすると、2 つのリストが表示されます。 最初のリストには、このマップのすべてのデータ・フィールドが表示され、2 番目のリストには、このマップのすべてのビューが表示されます。
6. 1 つのビューをダブルクリックすると、このビューのすべてのフィールドを示すリストが表示されます。

214 ページの図 105 に、Web ブラウザー・ウィンドウ内で実行されるアプレットを示します。

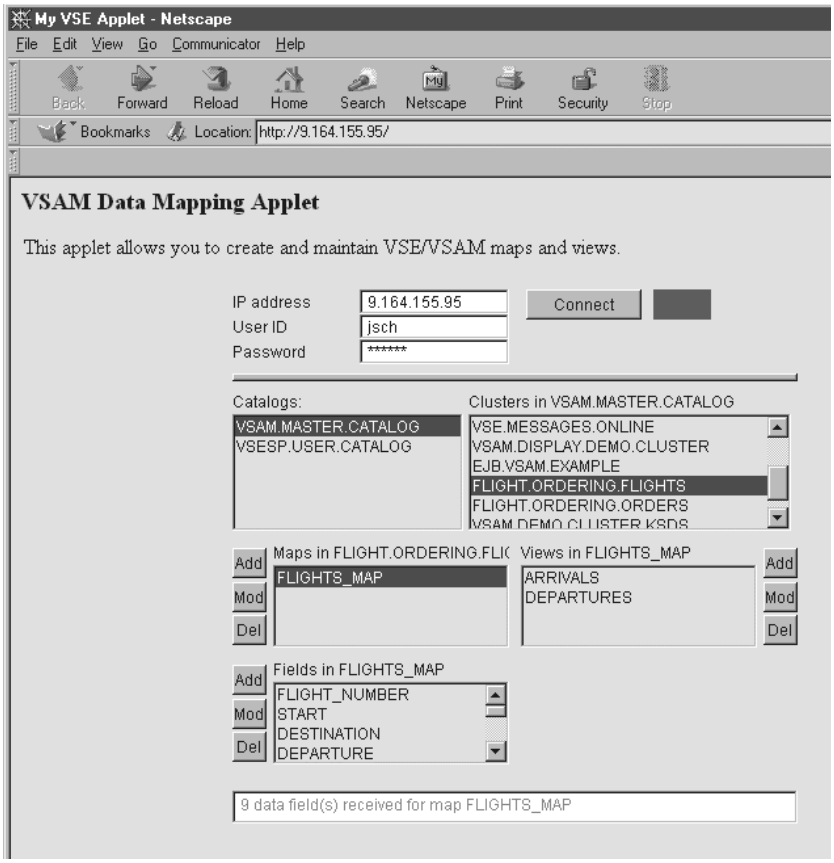


図 *105. VSAM* データ・マッピング・アプレット用ウィンドウ

VSAM ファイル FLIGHT.ORDERING.FLIGHTS には、フライトを記述するデータ・レコードが入っています。 各フライトは、いくつかのデータ・フィールド (DEPARTURE、ARRIVALS、SEATS など) で構成されます。 これらのデータ・フィールドは、完全レコードを記述するマップ FLIGHTS_MAP に入っています。

図 105 に示されているリストの隣に表示されているプッシュボタンを使用することにより、マップ、ビューおよびフィールドを追加、変更、あるいは削除することができます。マップ、ビュー、またはフィールドを追加または変更するプッシュボタンをクリックすると、新規データを入力できるダイアログが表示されます。

マップには、DEPARTURES および ARRIVALS という 2 つの異なるデータ・ビューがあり、これにより、マップのデータ・フィールドのサブセットが提供されます。 すべてのマップおよびビューは、VSAM マッピング定義が入っている VSAM ファイルに保管されています。 詳細については、 119 ページの『z/VSE ホストにおけるマップの保管方法』を参照してください。

z/VSE ホストで必要なアクティビティー

VSAM データ・マッピング・アプレットを実行できるようにするには、ご使用の z/VSE ホストで以下のアクティビティーを実行する必要があります。

1. 次のように、z/VSE コンソールで TCP/IP コマンドを入力することにより、z/VSE 上で Web サーバーを定義します。

```
DEFINE HTTPD,ID=MYHTTPD,ROOT=PRIMARY.TEST
```

これによって、ルート・ライブラリー PRIMARY.TEST を持った HTTP デーモンが開始されます。

2. Web ブラウザーが z/VSE ホストに接続するときに読み取られる、**index.html** という名前のファイルを作成します。 次に、サンプル・アプレットで使用する必要がある **index.html** ファイルを示します。

```
<html>
<head>
<title>VSAM Data Mapping Example Applet</title>
</head>
<body>
<h2>VSAM Data Mapping Applet</h2>
This applet can be used to create and maintain VSAM maps and views.
Please logon to your VSE host. When the connection is established, a
list of VSAM catalogs is displayed.
<p>
<center>
<applet code="com.ibm.vse.samples.VsamMappingApplet" width=440 height=420
        archive="applets.jar, vsecon.jar">  1
</applet>
</center>
</body>
</html>
```

1 「archive」タグは、アプレットを実行するのに必要なすべてのクラスが入っている Java アーカイブ (.JAR) の名前を指定します。 この例では、

- **applets.jar** ファイルに、アプレット固有のコードが入ります。
- ファイル **vsecon.jar** は、**VSEConnectors.jar** に等しいものですが、VSE ライブラリーに入れることができる短いファイル名が付けられている点が異なります。

3. ファイル **index.html** を、HTTP サーバーのルート・ライブラリーに入れます。

データ・マッピング・アプレットのデプロイ

データ・マッピング・アプレットをデプロイするには、以下を実行する必要があります。

1. **vsecon¥samples** ディレクトリーから、Java ソースをコンパイルし、JAR アーカイブを作成します。 これを行うには、以下のステートメントを使用します。

```
call javac com¥ibm¥vse¥samples¥VsamMappingApplet.java
call javac com¥ibm¥vse¥samples¥VsamAppletListener.java
call jar c0fv applets.jar com¥ibm¥vse¥samples¥VsamMappingApplet.class
                          com¥ibm¥vse¥samples¥VsamAppletListener.class
```

2. 作成された JAR アーカイブを、VSE HTTP サーバーのルート・ディレクトリーに (バイナリー・フォーマットで) 送信します。 これを行うには、エミュレーター・プログラムを使用するか、FTP を使用します。 JAR ユーティリティーは、ご使用のローカル Java インストールの一部になっています。 FTP の使用方法の詳細については、 29 ページの『VSE コネクター・クライアント のコピーの取得』 ページを参照してください。

データ・マッピング・アプレットの呼び出し

データ・マッピング・アプレット (またはその他のアプレット) を実行するには、そのアプレットは、

1. z/VSE ホストからローカル・ワークステーションにダウンロードされていなければなりません。
2. ローカル・ワークステーションにインストールされている Web ブラウザーの Java 仮想マシン内で実行されなければなりません。

VSE HTTP Server が実行されている場合、上記の HTML ファイルを表示するには、ご使用の z/VSE ホストの IP アドレスまたはシンボル名を、Web ブラウザーの *address/location* フィールドに入力するだけで済みます。 VSE HTTP Server を開始するには、z/VSE コンソールで、次のように TCP/IP コマンドを入力します。

```
xx q httpds
```

ここで *xx* は、TCP/IP 区画の応答 ID です。

さまざまな Web ブラウザーが JAR ファイルおよびクラス・ファイルを検索する方法

- *Netscape Communicator* は、ローカル・クラスパス内で検索を行うことはありません。 代わりに、常に、アプレットの archive タグに指定されているパスからクラスを取得します。
- *Microsoft Internet Explorer 3* は、archive タグを処理できません。
- *Microsoft Internet Explorer 4* および *5* では、ローカル・システム・クラスパスは、アプレットの archive タグに指定されている JAR ファイルのパスの前に配置されます。その結果として、ローカルに同じクラスがある場合は、z/VSE ホスト・ベースのクラスはロードされません。 Web ブラウザーは、代わりに、ローカル・クラスをロードします。

データ・マッピング・アプレット・クラスのセットアップ

一般的に、アプレットは、Java アプレット・クラス (Java Development Kit で提供されている) を拡張します。 ただし、提供されているデータ・マッピング・アプレットは、プッシュボタン・アクションを検出する *ActionListener* もインプリメントしています。

```
/* Import applet classes */
import java.applet.*;
...

public class VsamMappingApplet extends Applet
implements ActionListener, ItemListener
{
   VSESystem system;
   VSEConnectionSpec spec;
   VSEVsam vsam;
   VSEVsamCatalog catalog;
   VSEVsamCluster cluster;
   VSEVsamMap map;
   VSEVsamView view;
   VSEVsamField field, newField;
   VsamAppletListener vl;
   ...
```

図 *106. Java* クラスをセットアップするためのデータ・マッピング・アプレットのコード

データ・マッピング・アプレットの初期化

アプレットが初めて開始されると、Web ブラウザーから *init()* メソッドが呼び出されます。 この例では、

1. ログオン・コントロールが表示されます。
2. マップ、ビュー、およびデータ・フィールドを追加、変更、または削除するためのさまざまなダイアログの親フレームとして必要なフレームが作成されます。

```
public void init()
{
   f = new Frame();
   pgl = new PowerGridLayout(100, 66); // a box with 100 x 66 units  1
   setLayout(pgl);
   vMapFields = new Vector();
   vViewFields = new Vector();
   displayLogonDialog();       // show the dialogbox
   repaint();
}
```

図 *107.* データ・マッピング・アプレットを初期化するためのサンプル・コード

1 *PowerGridLayout* クラスは Java レイアウト・マネージャーの 1 つで、*VSE Navigator* 関数で頻繁に使用され、また、いくつかの例が VSE コネクター・クライアント に入っています。 *PowerGridLayout* クラスは、**com.ibm.vse.utilities** パッケージに入っています。

HTML ページの再表示または終了

HTML ページを再表示するときは、必ず、Web ブラウザーからアプレット *start()* メソッドを次のように呼び出します。

```
public void start()
{
}
```

HTML ページを終了するときは、Web ブラウザーから、アプレット *stop()* メソッドを次のように呼び出します。

```
public void stop()
{
}
```

マップを VSAM クラスターに追加するためのデータ・マッピング・アプレットの使用

次のコードは、データ・マッピング・アプレットを使用して *VSEVsamMap* を定義する方法を示しています。 次のように、同じ基本メソッドを、ビューおよびフィールドを定義するためにも使用できます。

1. ローカル・マップ・オブジェクトが、そのコンストラクターを使用して作成されます。
2. 次に、ローカル・マップ・オブジェクトが、*create()* メソッドを使用して、z/VSE ホスト上に作成されます。

z/VSE ホストにアクセスすると、*IOException* または *ConnectorException* エラーが起こる場合があります。 したがって、この呼び出しは、次に示すように、

「try-catch」文節の中に入れる必要があります。

```
public int addMap()
{
   ...

   /* Get map name from textfield on dialogbox */
   String name = tfName.getText().toUpperCase();

   /* Create local object */
   map = new VSEVsamMap(   1
           system,
           ((VSEVsamCatalog)(vCatalogs.elementAt(catIndex))).getFileID(),
           ((VSEVsamCluster)(vClusters.elementAt(cluIndex))).getFileID(),
           name);

   /* Create map on host */
   try {
      map.create();   2

      /* Add new mapname to map list ... */
      ...
   }
   catch (Exception e)
   {
      ...
   }
   ...
}
```

図 *108.* マップを *VSAM* クラスターに追加するためのデータ・マッピング・アプレット・コード

以下の番号は、図 108 の中の番号の説明です。

1 ここで、クラス *VSEVsamMap* の新規ローカル・インスタンスが作成されます。 この段階では、実行済みのホスト・アクションはまだありません。

2 *create()* メソッドを使用して、ホスト上にマップ定義が作成されます。

マップを変更するためのデータ・マッピング・アプレットの使用

マップ・オブジェクトは、*rename()* メソッドを使用して名前変更できます。このメソッドは、要求を z/VSE ホストに送信してマップの名前を変更します。 z/VSE ホストにアクセスすると、*IOException* または *ConnectorException* エラーが起こる場合があります。 したがって、この呼び出しは、次に示すように、「try-catch」文節の中に入れる必要があります。

```
public int modMap()
{
   ...
   /* Get name from textfield ... */
   String name = tfName.getText().toUpperCase();
   try {
      /* Change map on host */
      map.rename(name);
      ...
   }
   catch (Exception e)
   {
      ...
   }
   ...
}
```

図 *109.* マップを変更するためのサンプル・アプレット・コード

次のダイアログ・ウィンドウを使用して、マップのプロパティーを変更できます。

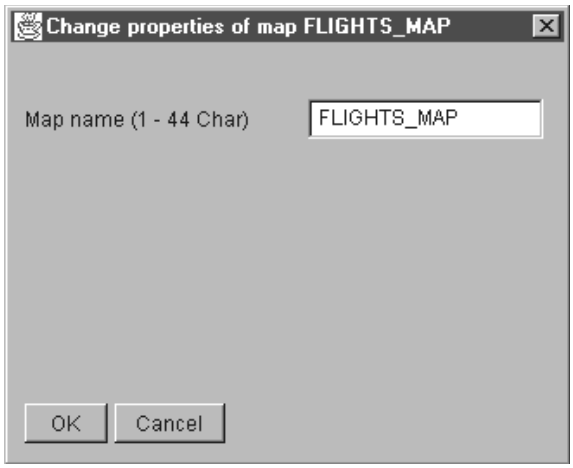


図 *110.* マップのプロパティーを変更するためのウィンドウ

マップのデータ・フィールドを変更するためのデータ・マッピング・アプレットの使用

次の例は、マップ内にあるフィールドを変更する方法を示し、以下のステップで構成されています。

1. マップのリスト内の選択項目を検索してマップをリトリーブします。
2. マップに関連するメソッド (*setFieldName()*、*setFieldType()* など) を使用して、フィールドのプロパティーを変更します。
3. ホスト上のフィールドが変更されると、ローカル・マップ・フィールドのリストが新規名称で更新されます。 さらに、マップ・フィールドのベクトル内のローカル・フィールド・オブジェクトが更新されます。

```
public int modMapfield()
{
   int i,type;
   ...
   /* Get new name from dialogbox */
   String name = tfName.getText().toUpperCase();

   /* Check if this name is already there. It's possible */
   /* to leave the name unchanged, but it's not possible */
   /* to change the name to another existing name. */
   ...

   /* Get other values from dialogbox */
   if (cb1.getState() == true)
      type = VSEVsamMap.TYPE_STRING;
   else if (cb2.getState() == true)
      type = VSEVsamMap.TYPE_BINARY;
   else if (cb3.getState() == true)
      type = VSEVsamMap.TYPE_PACKED;
   else if (cb4.getState() == true)
      type = VSEVsamMap.TYPE_SIGNED;
   else
      type = VSEVsamMap.TYPE_UNSIGNED;
   String len = tfLength.getText().toUpperCase();
   String offset = tfOffset.getText().toUpperCase();

   ...
   /* Get related map */
   i = mapList.getSelectedIndex();
   map = (VSEVsamMap)(vMaps.elementAt(i));    1
   ...
   /* Modify this field on host */
   try {
     map.setFieldName(oldName, name);
     map.setFieldType(map.getIndex(name), type);
     map.setFieldLength(map.getIndex(name),new Integer(len).intValue());
     map.setFieldOffset(map.getIndex(name),new Integer(offset).intValue());
   }
   catch (Exception e)
   {
     ...
   }

   /* Modify local field */
   i = mapFieldList.getSelectedIndex();
   mapFieldList.replaceItem(name, i);
   field = (VSEVsamField)(vMapFields.elementAt(i));
   field.setName(name);
   field.setType(type);
   field.setLength(new Integer(len).intValue());
   field.setOffset(new Integer(offset).intValue());
   ...
}
```

図 *111.* マップのデータ・フィールドを変更するためのサンプル・アプレット・コード

1 ここでは、ローカル・マップ、ビュー、およびフィールド・インスタンスをベクトル内に保持する必要があります。これは、このようなオブジェクトをAWT リスト内に保管することができないからです。

次のダイアログ・ウィンドウは上記のコードに対応し、このウィンドウを使用してマップのデータ・フィールドを変更できます。

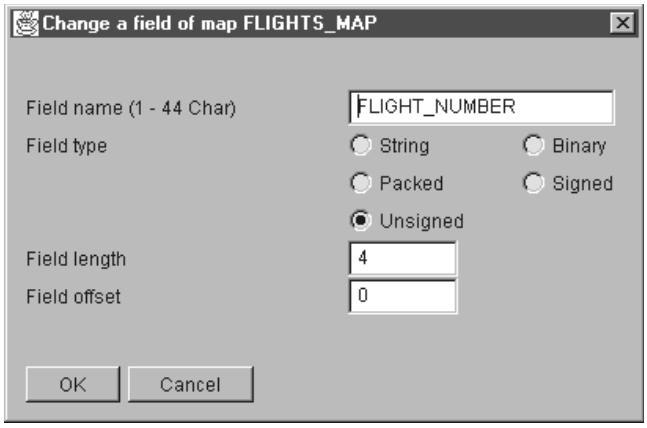


図 *112.* マップのデータ・フィールドを変更するためのウィンドウ

注**:** 長さの値およびオフセット値の有効性の検査は行われません。 また、フィールドを上書きする (同じオフセットを指定するが、それぞれのフィールドに異なる長さを指定する) こともできます。

AppletViewer を使用したローカルでのデータ・マッピング・アプレットの実行

ご使用のシステムをクイックにテストするために、データ・マッピング・アプレットをローカルで 実行できます。この場合、最初にコードを z/VSE ホストに入れる必要はありません。 VSE コネクター・クライアント のオンライン・ドキュメンテーションには **DataMapping.html** というファイルが入っており、ユーザーはこれを使用して、ローカル AppletViewer を使用したサンプル・アプレットを実行できます。 AppletViewer は、ご使用の Java インストールと一緒に提供されます。

```
<html>
<body>
<h2>VSAM Data Mapping Applet</h2>
<applet code="com.ibm.vse.samples.VsamMappingApplet"
        codebase="." archive="../VSEConnector.jar"
        width=460 height=460>
</applet>
</body>
</html>
```

以下の点に注意してください。

- 「codebase」タグおよび「archive」タグは、z/VSE ホスト・ベースの HTML ファイルのタグとは別のものです。 ここでは、「codebase」タグを使用して、すべてのアプレット関連コードが現行ディレクトリー (サブディレクトリーを含む) に入れられることを指定します。
- 「archive」タグは、VSE Java Beans クラス・ライブラリーが入っているオリジナルの **VSEConnector.jar** ファイルを指し示します。
- アプレット固有のコードを入れる 2 番目の JAR ファイルを作成する必要はありません。これは、アプレット・ビューアーが、これらのクラスを、ご使用のローカル・ファイル・システムから直接取得できるからです。

vsecon¥samples ディレクトリーに変更し、次のように、アプレットを呼び出します。

```
set classpath=.;..¥VSEConnector.jar;%classpath%
AppletViewer  MappingApplet.html
```

(ここでは、**VSEConnector.jar** ファイルが、すぐ上位のディレクトリーに保管されていると想定しています。)

この場合、関連する UNIX シェル・スクリプト次のようになります。

```
#! /bin/sh
export CLASSPATH=.:../VSEConnector.jar:$CLASSPATH
appletviewer MappingApplet.html
```

サンプル VSAM アプレットの実行

このトピックの説明は、VSE コネクター・クライアント で提供されている *VSAM アプレット* というサンプル・アプレットを基にしています。

VSE コネクター・クライアント で同様に提供されているこのほかのアプレットの例には、次のものがあります。

- *VsamSpaceUsage* アプレット (使用されている VSAM スペースおよび空き VSAM スペースを表示する)。
- データ・マッピング・アプレット (212 ページの『データ・マッピング・アプレットの説明』 ページに説明があります)。
- *DL/I* アプレット (237 ページの『サンプル DL/I アプレットの実行』 ページに説明があります)。

VSAM アプレット の説明

サンプルの VSAM アプレット は、Db2 ベース・コネクターと一緒にアプレットを使用する方法を説明する例です。 このサンプル・アプレットを使用すると、サンプル VSAM データ・クラスターに保管されている VSAM データを表示し、変更することができます。

VSAM アプレットは、開始されると、Db2 Connect データベース別名 **db2vsewm** に接続されます。 これによって、VSAM アプレットは、***db2vsewm*** を介して、z/VSE ホストのデータベース **sqlds** と通信できるようになります。

VSAM アプレットはサンプルの Db2 ストアード・プロシージャー (この例では VSAMSEL) を呼び出し、VSAMSQL コール・レベル・インターフェース (CLI) を使用して VSAM データにアクセスします。 CLI の詳細な説明については、451 ページの『Db2 ストアード・プロシージャーによる VSAM データへのアクセス』を参照してください。

VSAM アプレットを実行するには、以下のものをカスタマイズしておく必要があります。

- Db2 ベース・コネクター (詳しくは 95 ページの『第 10 章 Db2 ベース・コネクターのカスタマイズ』のステップ 1 から 6 を参照)。

- Db2 ストアード・プロシージャーのサンプル (詳しくは 108 ページの『ステップ 7: VSAM データ・アクセス用に Db2 ベース・コネクターをカスタマイズする』を参照)。
- ご使用の物理/論理中間層上の Db2 Connect (詳しくは 110 ページの『ステップ 10: Db2 Connect のインストールとクライアント/ホスト接続の確立』を参照)。このステップで、**sqlds** を、ご使用の物理/論理中間層上の Db2 Connect に定義します (ここで **sqlds** は、VSAM アプレットで使用される z/VSE ホスト上のサンプル・データベースです)。

注**:** VSAM アプレット は、オペレーティング・システムと Web ブラウザーの特定の組み合わせによっては実行できない場合があります。

3 層環境内のサンプル VSAM アプレットの使用方法を図 113 に示します。

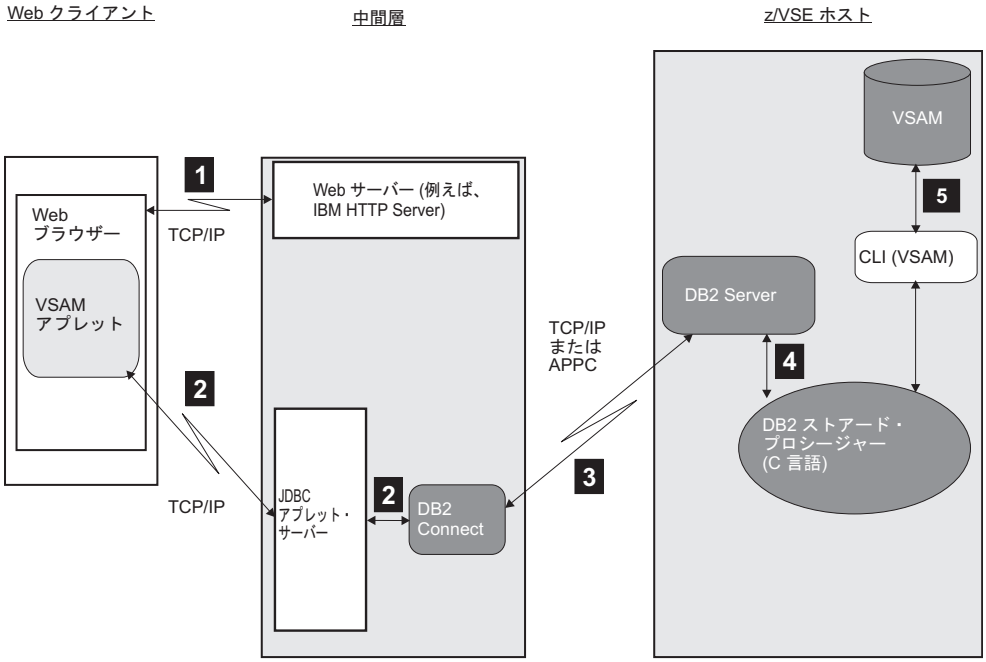


図 *113. VSAM* データにアクセスするためのサンプル *VSAM* アプレット の使用

1 クライアントの Web ブラウザーが、物理/論理中間層上で実行されている IBM HTTP Server (または別の Web サーバー) に HTML ページを要求します。 HTML ページには、*VSAM* アプレット のアプレット・タグが入っています。 VSAM アプレットは Web ブラウザーの Java 仮想マシンにロードされ、実行を開始します。

2 VSAM アプレットは、3 層プラットフォームの物理/論理中間層で実行されている Db2 Connect への接続をオープンします。 これは、*JDBC* アプレット・サーバー (「JDBC」は *Java Database Connectivity* の省略形) を使用して行われます。 JDBC アプレット・サーバーを使用すると、以下のアプレットの制約事項に関係なく行えます。

- アプレットが新規ネットワーク接続をオープンできるのは、アプレットがダウンロードされた元のプラットフォーム (この場合は物理/論理中間層) に対してだけです。

- アプレットがアクセスできるのは、アプレットがダウンロードされた元のプラットフォーム (この場合は物理/論理中間層) のファイル・システムだけです。

次に、VSAM アプレットは、サンプル Db2 ストアード・プロシージャー (この例では VSAMSEL) を呼び出します。

3 Db2 Connect は、Db2 Connect データベース別名 **db2vsewm** を使用して、Db2 Server for VSE と通信します。 これによって、Db2 Connect は、z/VSE ホストのデータベース **sqlds** にアクセスできるようになります。 DRDA (分散リレーショナル・データベース体系) を使用します。基礎のプロトコルは APPC または TCP/IP のどちらでもかまいません。

4 Db2 Server for VSE が、ストアード・プロシージャー・サーバーを使用して、サンプル Db2 ストアード・プロシージャー (この例では VSAMSEL) を実行します。

5 これで、サンプル Db2 ストアード・プロシージャーは、VSAMSQL CLI を使用して z/VSE ホストに保管されている VSAM データにアクセスすることによって VSAM アプレットの要求を実行できます。

注:

1. WebSphere Application Server は、上記のプロセスの物理/論理中間層では必要ありません。
2. VSE コネクター・サーバー は、上記のプロセスの z/VSE ホストでは必要ありません。

サンプル VSAM アプレットの開始

VSAM アプレットを実行できるようにするには、このトピックで説明するステップを実行する必要があります。

1. VSAM アプレットを呼び出すための HTML ファイルの作成

HTML ファイルを作成し、そこから VSAM アプレットを呼び出せるようにしなければなりません。この HTML ページが表示されるたびに、Web ブラウザーの JVM (Java 仮想マシン) にアプレットがロードされて実行されます。 これによって VSAM アプレットが、JDBC を使用して物理/論理中間層のデータベースに接続されます。

次に、そのような HTML ファイルの例を示します。

```
<html>
<head>
<title>JDBC Example Applet to call a Db2 Stored Procedure</title>
</head>
<body>
<h2>JDBC Example Applet that access VSE/VSAM using the Db2-based connector</h2>
This applet lets you browse, insert, update and delete any records from the
VSE/VSAM sample cluster for the Db2-based connector. You can also browse all VSAM records
using the view OFFER. The JDBC Applet calls the corresponding Stored Procedures defined
within your Db2 Server for VSE.
<p>
<center>
<applet code="DB2ConnectorJDBCApplet.class" archive="db2applt.jar, db2java.zip"
            width=440 height=420>
```

```
</applet>
</center>
</body>
</html>
```

2. VSAMSEL.C のコンパイル

VSAMSQL CLI を使用する Db2 ストアード・プロシージャーは *C* 言語 で作成されているため、VSAMSEL.C (この例で使用するサンプルの Db2 ストアード・プロシージャー) を IBM LE/VSE *C (VSE* 版*)* コンパイラーを使用してコンパイルします。また、すべての DB ストアード・プロシージャーは、LE/VSE に準拠していなければなりません。 ジョブ・ストリーム SKCPSTP を使用します。これは ICCF ライブラリー 59 にあります。

サンプルの Db2 ストアード・プロシージャーには SQL ステートメントが入っていません。代わりに、VSAMSQL コール・レベル・インターフェース (CLI) を使用するからです。したがって、SQL プリコンパイラーを実行する必要はありません。ただし、ユーザー独自のデータにアクセスするために SQL ステートメントおよび VSAMSQL CLI の両方を組み込むことにした場合は、追加の SQL プリコンパイル・ステップを実行する必要があります。

次に、*VSAMSEL.C* のサンプル Db2 ストアード・プロシージャーをコンパイルするために使用できる JCL (SKCPSTP 内にある JCL) を示します。同様の方法で、その他のサンプル Db2 ストアード・プロシージャー (VSAMINS.C、VSAMUPD.C、および VSAMDEL.C) もコンパイルします。

```
// JOB SKCPSTP COMPILE SAMPLE STORED PROCEDURE
// LIBDEF *,SEARCH=(PRD2.SCEEBASE,PRD2.DBASE,PRD1.BASE)
// SETPARM CATALOG=1
// IF CATALOG = 1 THEN
// GOTO CAT
// OPTION ERRS,SXREF,SYM,LIST,NODECK
// GOTO ENDCAT
/. CAT
// LIBDEF PHASE,CATALOG=LIB.SUBLIB
// OPTION ERRS,SXREF,SYM,NODECK,CATAL
   PHASE VSAMSEL,*
/. ENDCAT
 INCLUDE @@TRT
 INCLUDE IESVSQLO
// EXEC EDCCOMP,SIZE=EDCCOMP,PARM='LONGNAME RENT SS SOURCE             X
               INFILE(DD:PRD1.BASE(VCLIUTIL.C))'
// EXEC EDCCOMP,SIZE=EDCCOMP,PARM='LONGNAME RENT SS SOURCE             X
               INFILE(DD:PRD1.BASE(VSAMSEL.C))'
/*
// IF CATALOG NE 1 OR $MRC GT 4 THEN
// GOTO NOLNK
// EXEC EDCPRLK,SIZE=EDCPRLK,PARM='NATLANG(ENU)/UPCASE'
/*
// EXEC LNKEDT,SIZE=256K
/. NOLNK
/&
```

3. Db2 Server for VSE への VSAMSEL の定義

ここで、VSAMSEL (この例で使用する Db2 ストアード・プロシージャー) を、ご使用の Db2 Server for VSE システムに定義しなければなりません。このためには、以下を実行する必要があります。

1. VSAMSEL フェーズ (前のステップでコンパイルされている) を、ストアード・プロシージャー・サーバー の検索パスに入っているライブラリーの中に入れます。
2. CREATE PROCEDURE ステートメントを使用して、VSAMSEL をデータベース・マネージャーに定義します。この目的のために、ICCF ライブラリー 59 に入っているジョブ・ストリーム SKCRESTP を使用できます。

   次に、VSAMSEL 用の CREATE PROCEDURE ステートメントを示します。

```
CREATE PROCEDURE VSAMSEL (IN char(180),INOUT INT,OUT INT,OUT CHAR(20),-
OUT CHAR(20),OUT CHAR(20),OUT INT,OUT INT,OUT CHAR(20),OUT CHAR(20),-
OUT CHAR(20),OUT INT,OUT INT,OUT CHAR(20),OUT CHAR(20),OUT CHAR(20),-
OUT INT,OUT INT,OUT CHAR(6),OUT INT,OUT CHAR(252)) EXTERNAL,LANGUAGE C,-
STAY RESIDENT YES,SERVER GROUP,PARAMETER STYLE GENERAL
```

4. VSAM データ・クラスターの定義

ジョブ SKVSSAMP (ICCF ライブラリー 59 にある) を使用して以下のことを行います。

- サンプル VSAM アプレットで使用される VSAM データ・クラスターを作成する。
- サンプル・レコードを、VSAM データ・クラスターにロードする。
- RECMAP コマンドを使用して、VSAM データ・クラスター用のサンプルのマップとビューを作成する。 RECMAP について詳しくは 119 ページの『RECMAP を使用したマップの定義』を参照してください。

```
// JOB SKVSSAMP LOAD VSE/VSAM CONNECTOR SAMPLE DATA CLUSTER
* ********************************************************************
*                                                                    *
*   NOTE: IF YOU SPECIFY A DIFFERENT CATALOG THAN                    *
*         VSESP.USER.CATALOG, YOU HAVE TO CHANGE THE PATH FOR        *
*         THE RECORD MAP (CATALOG/CLUSTER/MAP/VIEW) IN THE           *
*         CLIENT PROGRAMS THAT CALL THE STORED PROCEDURE AS WELL     *
*         (CVSAMSEL.SQC ETC.)                                        *
*                                                                    *
* ********************************************************************
*
*   DEFINING THE VSAM CONNECTOR SAMPLE DATA CLUSTER 'VCSAMPD'
*
* ********************************************************************
// EXEC IDCAMS,SIZE=AUTO
 DELETE VSAM.CONN.SAMPLE.DATA PURGE CATALOG(VSESP.USER.CATALOG)
 DEFINE CLUSTER ( -
        NAME ( VSAM.CONN.SAMPLE.DATA ) -
        RECORDS ( 30 30 ) -
        SHAREOPTIONS ( 2 ) -
        RECORDSIZE ( 120  120 ) -
        VOLUMES ( DOSRES SYSWK1 ) -
        NOREUSE -
        INDEXED -
        FREESPACE ( 15 7 ) -
        KEYS ( 4   0 ) -
        NOCOMPRESSED ) -
        DATA ( NAME ( VSAM.CONN.SAMPLE.DATA.@D@ ) -
        CONTROLINTERVALSIZE ( 4096  ) ) -
        INDEX ( NAME ( VSAM.CONN.SAMPLE.DATA.@I@ )) -
        CATALOG (VSESP.USER.CATALOG )
   IF LASTCC NE 0 THEN CANCEL JOB
/*
// OPTION STDLABEL=DELETE
                 VCSAMPD
/*
// OPTION STDLABEL=ADD
```

```
// DLBL VCSAMPD,'VSAM.CONN.SAMPLE.DATA',,VSAM,                          X
               CAT=VSESPUC
/*
* $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
*                                                                     *
*               NOW LOADING THE SAMPLE DATA                           *
*                                                                     *
* $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
// LIBDEF *,SEARCH=(PRD1.BASE)
// EXEC VSAMSMPD,SIZE=AUTO
// ON $RC>0 GOTO FINISH
/*
* $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
*                                                                     *
*    DEFINE MAPS AND VIEWS USING THE RECORD MAPPING UTILITY           *
*                                                                     *
* $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
// EXEC IDCAMS,SIZE=AUTO
 RECMAP DEFINE ( MAP( USEDCARS )  -
   MAPCOLUMN( -
    ( ARTICLENO    FIELD( 0(0)   L(4)   T(SINTEG))  POS(1) )   -
    ( MANUFACTURER FIELD( 0(4),  L(20)  T(STRING))  POS(2) )   -
    ( TYPE         FIELD( 0(24)  L(20)  T(STRING))  POS(3) )   -
    ( MODEL        FIELD( 0(44)  L(20)  T(STRING))  POS(4) )   -
    ( HP           FIELD( 0(64)  L(2)   T(SINTEG))  POS(5) )   -
    ( DISPLACEMENT FIELD( 0(66)  L(2)   T(SINTEG))  POS(6) )   -
    ( CYLINDERS    FIELD( 0(68)  L(2)   T(SINTEG))  POS(7) )   -
    ( COLOUR       FIELD( 0(70)  L(20)  T(STRING))  POS(8) )   -
    ( FEATURES     FIELD( 0(90)  L(20)  T(STRING))  POS(9) )   -
    ( PRICE        FIELD( 0(110) L(4)   T(SINTEG))  POS(10))   -
   ) -
 ) -
 CATALOG( VSESP.USER.CATALOG )  -
 CLUSTER( VSAM.CONN.SAMPLE.DATA )
 RECMAP DEFINE ( MAP( USEDCARS )  -
   VIEW( OFFER ) -
    VIEWCOLUMN( ( ARTICLENO     REFCOLUMN( ARTICLENO    )) -
     ( MANUFACTURER  REFCOLUMN( MANUFACTURER )) -
     ( TYPE          REFCOLUMN( TYPE         )) -
     ( MODEL         REFCOLUMN( MODEL        )) -
     ( PRICE         REFCOLUMN( PRICE        )) -
   ) -
 ) -
 CATALOG( VSESP.USER.CATALOG )  -
 CLUSTER( VSAM.CONN.SAMPLE.DATA )
 RECMAP LIST (CLUSTERS)
/*
/. FINISH
/*
/&
```

5. VSAM アプレット用の JAR ファイルの作成

VSAM アプレットを実行する前に、以下を実行する必要があります。

1. アプレットに関連するクラス・ファイルをこの JAR (Java アーカイブ) ファイルにコピーすることによって JAR ファイルを作成します。 これを行うには、ご使用の VSE コネクター・クライアント・システムのサンプルのディレクトリーに変更し、以下のステートメントを実行します。

   ```
   call jar c0fv db2applt.jar com¥ibm¥vse¥db2¥DB2ConnectorJDBCApplet.class
      com¥ibm¥vse¥db2¥MessageDialog.class
      com¥ibm¥vse¥db2¥PowerGridLayout.class
      com¥ibm¥vse¥db2¥PowerGridLayoutInfo.class
   ```

2. 上記の (1.) の JAR ファイルを、物理/論理中間層プラットフォーム上のご使用の Web サーバー (例えば、IBM HTTP サーバー、または Apache サーバー) の HTML ディレクトリーにコピーします。

注: VSAM アプレットは、上記 (1.) の代わりの方法として *3* 層環境 で実行されるので、Web サーバー上の HTML ディレクトリーにクラス・ファイルを直接コピーできます。これは、アプレット・コードが z/VSE ホストに保管されておらず、したがって、クラス・ファイルをショート・ネーム・アーカイブに保管する必要がないからです。

3. JDBC アプレット・サーバーを物理/論理中間層プラットフォームで開始します。 このサーバーは、VSAM アプレットによって開始された要求を処理します。 また、ユーザーは、JDBC アプレット・サーバーが使用できる未使用の TCP/IP ポート番号を選択する必要があります (すべてのポートは、*services* ファイルに定義されています)。 したがって、TCP/IP ポート **6789** (デフォルト) を選択する場合、次のように入力します。

```
db2jstrt 6789
```

VSAM アプレットの呼び出し

HTML ページがロードされると、VSAM アプレットは、次のように処理されます。

1. 物理/論理中間層サーバーからローカル・ワークステーションにダウンロードされる。
2. ローカル・ワークステーションにインストールされている Web ブラウザーの Java 仮想マシン内で実行されなければなりません。

VSAM アプレットは、次の 2 つの方法で呼び出すことができます。

- ご使用のローカル Java システムの一部であるアプレット・ビューアー (Windows および OS/2 ではファイル **AppletViewer.exe**) を直接使用する方法。これを行うには、コマンド・プロンプトで次のように入力します。

  ```
  AppletViewer db2index.html
  ```

  ここで、**db2index.html** には、 224 ページの『1. VSAM アプレットを呼び出すための HTML ファイルの作成』に示されている HTML タグが入ります。
- Web ブラウザーを使用する方法。 この方法を実行するには、ご使用の物理/論理中間層プラットフォームのシンボル名または IP アドレスを入力し、その後に、ご使用の Web ブラウザーのアドレス・フィールドの中のサンプル HTML ファイルの名前を続けます。 この例の場合は、以下のように入力します。

  ```
  http://ebusvse/db2/db2index.html
  ```

VSAM アプレットは、呼び出されると、 229 ページの図 114 に示すメイン・ウィンドウを表示します。

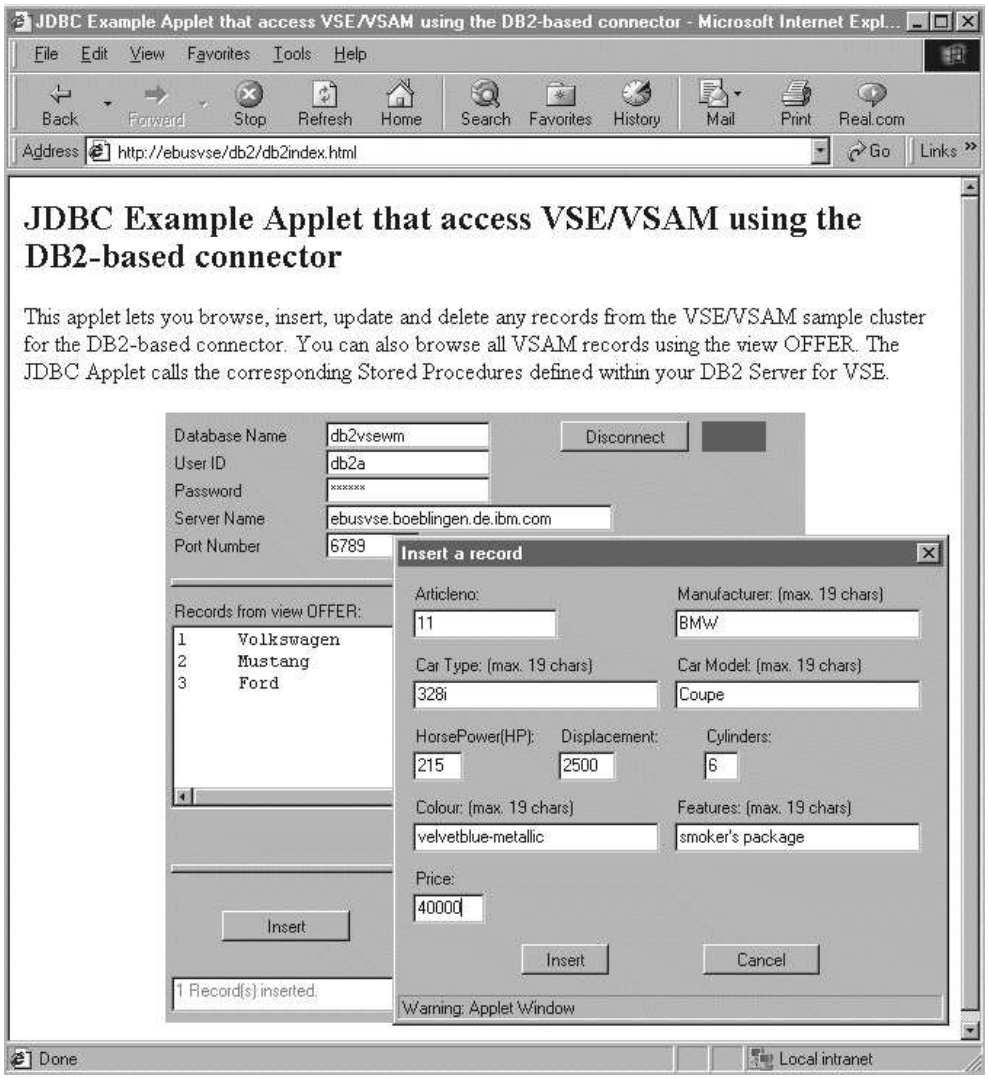


図 *114.* サンプル *VSAM* アプレットによって表示されるウィンドウ

図 114 には、**Disconnect** ボタンの右に長方形があります。 これは、VSAM アプレットと **sqlds** サンプル・データベース (これは Db2 Connect のデータベース別名 **db2vsewm** を使用してアクセスされる) の間の接続の状況を表します。 この長方形が、

- 緑色 であるときは、アプレットは **sqlds** に接続されていて、ボタンには **Disconnect** が表示されます。
- 赤色 であるときは、アプレットは **sqlds** に接続されておらず、ボタンには **Connect** が表示されます。

図 114 に示したメイン・ウィンドウには、「*Insert a record* (レコードを挿入する)」というダイアログ・ウィンドウが組み込まれており、ユーザーはこのダイアログ・ウィンドウを使用して、VSAM レコードをサンプル VSAM クラスターに挿入します。 ユーザーは、その他の同様のダイアログ・ウィンドウを使用して、サンプル VSAM データ・クラスター内のレコードを最新表示、更新、あるいは削除することができます。 現在表示されているダイアログ・ウィンドウに応じて、以下のことを行うことができます。

押すボタン
: 実行される内容 ...

Connect
: z/VSE への接続が Db2 Connect を使用して再確立され、次に Db2 ストアード・プロシージャー **VSAMSEL** が **OFFER** ビューと一緒に使用されて、サンプル VSAM データ・クラスターにあるすべてのレコードが選択されます。

Insert Db2 ストアード・プロシージャー **VSAMINS** が使用され、新規レコードがサンプル VSAM データ・クラスターに挿入されます。

Update
: Db2 ストアード・プロシージャー **VSAMUPD** が使用され、サンプル VSAM データ・クラスター内のレコードが置き換えられます。

Delete
: Db2 ストアード・プロシージャー **VSAMDEL** が使用され、サンプル VSAM データ・クラスターから VSAM レコードが削除されます。

Refresh
: Db2 ストアード・プロシージャー **VSAMSEL** が **OFFER** ビューと一緒に使用され、サンプル VSAM データ・クラスターにあるすべてのレコードが選択されます。

229 ページの図 114 の最下部には「状況表示行」があり、ここに、ユーザーが実行中のアクションに関連したエラー・メッセージまたは状況メッセージが表示されます。

サンプル VSAM データ・クラスターは 226 ページの『4. VSAM データ・クラスターの定義』に説明があります。

DB2ConnectorJDBCApplet.java (クライアント・サイド・プログラム) の説明

このトピックでは、VSAM アプレットの機能のほとんどで使用される *DB2ConnectorJDBCApplet.java* のメイン・ステップ について説明します。これは、223 ページの図 113 の Web ブラウザーの JVM で実行されます。

さらに、VSAM アプレットでは、以下のヘルパー・クラス が使用されます。

- *MessageDialog.java*。これは、**OK** ボタンと一緒に、テキストの特定行が入るメッセージ・ダイアログ・ウィンドウを表示します。
- *PowerGridLayout.java* および *PowerGridLayoutInfo.java*。これらは、Java レイアウト・マネージャーのタイプで、VSE コネクター・クライアント のサンプルによって頻繁に使用されます。この Java レイアウト・マネージャーは、*com.ibm.vse.utilities* パッケージ内で提供されます。 ユーザーは、独自のアプリケーションを作成するときに、この Java レイアウト・マネージャーを使用することができます。

ステップ 1: JDBC (Java Database Connectivity) クラスのインポート

最初のステップで、JDBC クラスがインポートされます。 JDBC は、Db2 Server for VSE で使用される VSAMSEL (サンプル Db2 ストアード・プロシージャーのうちの 1 つ) を呼び出すために必要です。

さらに、アプレット固有のクラスがインポートされます。 アプレットは Java のアプレット・クラスを一般的に拡張する ので、VSAM アプレットは、以下のものをインプリメントします。

- *ActionListener*。これは、マウスのクリックおよびプッシュボタン・アクションを listen します。
- *WindowListener*。これは、ウィンドウ・アクションを処理します。

```
 ...
/* import JDBC classes */
import java.sql.*;

/* Import AWT classes */
import java.awt.*;
import java.awt.event.*;

/* Import applet classes */
import java.applet.*;


public class DB2ConnectorJDBCApplet extends Applet
implements ActionListener, WindowListener
{
```

ステップ 2: 必要な JDBC ドライバー・クラスのロード

2 番目のステップで、必要な JDBC ドライバー・クラスがロードされます。 これは、通常、静的セクションで実行されます。 *net* というドライバー・クラスが使用され、アプレットがどのクライアント・ワークステーションでも実行できるようにします。

```
...
// register the JDBC driver with DriverManager
static
{
   try
   {
      Class.forName ("COM.ibm.db2.jdbc.net.DB2Driver");
   }
   catch (Exception e)
   {
      System.out.println ("¥n  Error loading DB2 Driver...¥n");
      e.printStackTrace ();
   }
} // end static block
...
```

ステップ 3: init() メソッドのインプリメント

3 番目のステップで、*init()* メソッドがインプリメントされます。 *init()* メソッドは、VSAM アプレットが最初に開始されるときに Web ブラウザーから呼び出されます。 サンプル VSAM データ・クラスター内の VSAM レコードを最新表示、挿

入、更新、または削除する際に使用されるさまざまなダイアログの親になるフレームが作成されます (226 ページの『4. VSAM データ・クラスターの定義』に説明があります)。

```
...
public void init()
{
   /* Create a frame that is needed for the dialogs */
   /* to insert/update/delete records and to display message dialogs */
   f = new Frame();

   msgDialog = new MessageDialog(f, true);
   pgl = new PowerGridLayout(100, 66);
   setLayout(pgl);

   displayMainDialog();
   repaint();
}
...
```

ステップ 4: Db2 Connect を使用した、z/VSE データベースへの接続の確立

4 番目のステップで、Db2 Connect のデータベース別名 **db2vsewm** を使用して、z/VSE ホストのデータベース **sqlds** への接続が確立されます。 これによって、Db2 Connect は、z/VSE ホストへのデータベース要求の経路を定めることができます。

次の JDBC URL フォーマットを使用して、接続をセットアップします。

```
<protocol>:<subprotocol>://<hostname or tcpip address>:<port number>/<database name>
```

<database name> のもとに指定されているデータベースの場合は、以下のものが、JDBC *getConnection()* 呼び出しで提供されます。

- JDBC URL
- ユーザー ID
- パスワード

getConnection() 呼び出しは、これらの値を、メイン・ウィンドウの *connect* セクションのテキスト・フィールドからリトリーブします。

```
...

public void connectToDB()
{
   String url    = "jdbc:db2://" + DBServerAddr.getText() + ":"
                     + DBServerPort.getText() + "/"
                     + DBName.getText();
   ...
   try
   {
      // connect with user-provided username and password
      con = DriverManager.getConnection(url, userid.getText(), passw.getText());
   }
   catch (SQLException sqlExc)
   {
      ...
   }
   ...
}
```

ステップ 5: VSAMSEL の呼び出し

最終ステップで、VSAMSEL (この例で使用されているサンプルの Db2 ストアード・プロシージャー) が呼び出されます。

また、VSAMSEL は、エンド・ユーザーが **Refresh** ボタンを押したときにも呼び出されます。

```
...
/**
 * call Db2-based connector Stored Procedure on z/VSE
 * This will retrieve all records from view OFFER, which then will be
 * put in the corresponding listbox of the applet.
 * The Stored Procedure VSAMSEL uses the VSAMSQL CLI to access the VSAM
 * records.
 */
 public void callStpVSAMSEL()
 {
    ...
 }
```

Db2 ストアード・プロシージャーに JDBC が呼び出しを出す方法の詳細については、VSE コネクター・クライアント のオンライン・ドキュメンテーションで提供されている JDBC のアプリケーション・サンプルを参照してください。 (オンライン・ドキュメンテーションについては 30 ページの『オンライン・ドキュメンテーション・オプションの使用』に説明があります。)

VSAMSEL の説明

このトピックでは、 223 ページの図 113 の z/VSE ホストで実行される VSAMSEL (この例で使用するサンプルの Db2 ストアード・プロシージャー) のメイン・ステップ について説明します。このほかのサンプル Db2 ストアード・プロシージャー (VSAMINS、VSAMUPD、および VSAMDEL) は、このトピックでは説明しません。

VSAMSEL は、 229 ページの図 114 の **Connect** ボタンまたは **Refresh** ボタンのどちらかが押されたときに呼び出され、さらに、VSAMSQL CLI (コール・レベル・インターフェース) を使用して VSAM データがアクセスされた方法を示します。

ステップ 1: ヘッダー・ファイル iesvsql.h の VSAMSEL への組み込み

最初のステップで、ヘッダー・ファイル **iesvsql.h** が VSAMSEL に組み込まれます。 このヘッダー・ファイルは、VSAMSQL CLI インターフェースを使用して VSAM データ・クラスターにアクセスするすべての Db2 ストアード・プロシージャーで必要です。

ヘッダー・ファイル **iesvsql.h** は、次のことを行います。

- すべての関数プロトタイプおよび VSAMSQL CLI 定義を保持します。
- *check_error()* 関数のプロトタイプが入っています。この関数は、VSAMSQL CLI 呼び出しのエラー処理ルーチンの例を提供します。 *check_error()* 関数は、それぞれの VSAMSQL CLI 呼び出しの後で呼び出されます。

```
...
// include VSAMSQL CLI
#include "iesvsql.h"
// include error utility functions
#include "vcliutlh.h"
...
```

ステップ 2: VSAMSQL CLI 環境の初期化

2 番目のステップで、*VSAMSQL CLI* (コール・レベル・インターフェース) 環境が初期化されます。 VSAMSQL CLI については、 451 ページの『Db2 ストアード・プロシージャーによる VSAM データへのアクセス』に説明があります。

以下のハンドル が、初期化されます。

- 環境ハンドル。これは、グローバル情報 (有効接続ハンドルおよびアクティブ接続ハンドルなど) にアクセスを提供します。
- 接続ハンドル。これは、接続情報 (接続に関する有効ステートメントおよび記述子ハンドルなど) にアクセスを提供します。
- ステートメント・ハンドル。これは、ステートメント情報 (VSAMSQL ステートメント処理に関するエラー・メッセージおよび状況情報など) にアクセスを提供します。

```
....

/*****************************************************************/
/*      initialize VSAMSQL CLI Environment                       */
/*****************************************************************/
// allocate Environment
rc = VSAMSQLAllocHandle(VSAMSQL_HANDLE_ENV,VSAMSQL_NULL_HANDLE, &hEnv);
cont = check_error(VSAMSQL_HANDLE_ENV,hEnv,rc,
                   o_sqlstate,o_message,&o_native_error,SYSLST,"VSAMSEL-1");

//allocate Connection
if (cont == STP_CONT) {
   rc = VSAMSQLAllocHandle(VSAMSQL_HANDLE_DBC,hEnv,&hDBC);
   cont = check_error(VSAMSQL_HANDLE_DBC,hDBC,rc,
                      o_sqlstate,o_message,&o_native_error,SYSLST,VSAMSEL-2");
} // end if

//allocate Statement
if (cont == STP_CONT) {
   rc = VSAMSQLAllocHandle(VSAMSQL_HANDLE_STMT,hDBC,&hStmt);
   cont = check_error(VSAMSQL_HANDLE_STMT,hStmt,rc,
                      o_sqlstate,o_message,&o_native_error,SYSLST,"VSAMSEL-3");
} // end if
...
```

ステップ 3: VSAM レコードの読み取りの開始

3 番目のステップで、VSAM レコードの読み取りを開始する照会ステートメントが準備され、実行されます。 *i_recordmap* というビュー内に入っているすべてのフィールドが選択されます (ビュー *OFFER* は、この例の内のクライアント・プログラムから受け渡されます)。

3 番目のステップで、次のことが行われます。

1. *VSAMSQLPrepare()* 関数呼び出しは、SELECT ステートメント・ストリングをステートメント・ハンドルに関連付けます。

2. *VSAMSQLBindParameter()* は、*i_lastkey* を、SELECT ステートメント内の WHERE 文節にバインドします (ここで **?** は、対応するプレースホルダーです)。
3. *i_lastkey* が、VSAMSEL (サンプルの Db2 ストアード・プロシージャー) の入力パラメーターとして受け渡されます。
4. *VSAMSQLExecute()* は、ステートメントが正常に準備されると、そのステートメントを実行します。 ステートメント・ハンドル *hStmt* は、照会ステートメントを修飾します。

```
...

/*****************************************************************/
/*      prepare and execute query statement                      */
/*****************************************************************/
if (cont == STP_CONT) {
   sprintf(vsamsqlstmt, "SELECT * FROM %s WHERE ARTICLENO > ? ",
           i_recordmap);
   rc = VSAMSQLPrepare(hStmt, vsamsqlstmt, VSAMSQL_NTS);
   cont = check_error(VSAMSQL_HANDLE_STMT,hStmt,rc,
                      o_sqlstate,o_message,&o_native_error,SYSLST,"VSAMSEL-4");
} // end if

// Bind local Variables to Stmt
if (cont == STP_CONT) {
   rc = VSAMSQLBindParameter(hStmt,1,VSAMSQL_PARAM_INPUT,
                             VSAMSQL_C_LONG,VSAMSQL_INTEGER,
                             0,0,&i_lastkey,0,NULL);
   cont = check_error(VSAMSQL_HANDLE_STMT,hStmt,rc,
                      o_sqlstate,o_message,&o_native_error,SYSL"VSAMSEL-5");

} // end if

if (cont == STP_CONT) {
   rc = VSAMSQLExecute(hStmt);
   cont = check_error(VSAMSQL_HANDLE_STMT,hStmt,rc,
                      o_sqlstate,o_message,&o_native_error,SYSLST,"VSAMSEL-6");
} // end if
...
```

ステップ 4: 照会ステートメントの結果の取得

4 番目のステップでは、すでに照会ステートメントが実行されているので、その結果を取り出すことができます。 しかし、主に、結果セットにあるすべての列は、ローカル変数 (*tmp.articleno*) に関連付ける必要があります。これは、*VSAMSQLBindCol()* 関数呼び出しを使用して行われます。この関数呼び出しは、結果セット内のそれぞれの列ごとに実行する必要があります。

for()-loop 内で、ビュー *OFFER* にある単一レコードが、*VSAMSQLFetch()* を使用してリトリーブされます。

VSAMSEL は、次のように、3 つの結果レコードを取り出して、クライアント・プログラムに戻すことができます。

1. 存在する結果レコードが 3 より少ない場合、条件 VSAMSQL_NO_DATA_FOUND が表示されます。
2. *for()-loop* の外側の *VSAMSQLFetch()* 関数呼び出しは、3 より多い結果レコードが存在するかどうかを検査します。

3. 3 より多い結果レコードが存在する場合、*o_resultrows* が 1 つ増やされます。この値は、次のレコードをリトリーブするために VSAMSEL を呼び出す必要があるか否かを判別するために、クライアント・プログラムによってインディケーターとして使用されます。

```
...

/*****************************************************************/
/*      retrieve result set                                      */
/*****************************************************************/
// bind columns to local variables and retrieve results
if (cont == STP_CONT) {
   // bind parameter articleno
   rc = VSAMSQLBindCol(hStmt,1,VSAMSQL_C_LONG,
                       &tmp.articleno,0,&buf_len);
   cont = check_error(VSAMSQL_HANDLE_STMT,hStmt,rc,
                      o_sqlstate,o_message,&o_native_error,SYSLST,"VSAMSEL-7");
} // end if
...

// fetch result row(s)
for (i=0; i < NUM_ROWS; i++)
{
   if (cont == STP_CONT) {
      rc = VSAMSQLFetch(hStmt);
      cont = check_error(VSAMSQL_HANDLE_STMT,hStmt,rc,
                     o_sqlstate,o_message,&o_native_error,SYSLST,"VSAMSEL-C");
   } // end if

   // if successful - store retrieved columns
   if ( cont == STP_CONT && rc != VSAMSQL_NO_DATA_FOUND) {
      o_resultrows ++;
      o_parm[i].articleno = tmp.articleno;
      strcpy(o_parm[i].manufacturer,tmp.manufacturer);
      strcpy(o_parm[i].type,tmp.type);
      strcpy(o_parm[i].model,tmp.model);
      o_parm[i].price     = tmp.price;
   } // end if
   ...
} // end for

// check if more result exist
if (cont == STP_CONT) {
   rc = VSAMSQLFetch(hStmt);
   cont = check_error(VSAMSQL_HANDLE_STMT,hStmt,rc,
                   o_sqlstate,o_message,&o_native_error,SYSLST,"VSAMSEL-D");
   if (rc != VSAMSQL_NO_DATA_FOUND)
      o_resultrows ++;
} // end if
...
```

ステップ 5: VSAMSQL CLI 環境の割り振り解除

5 番目のステップでは、すべての VSAMSQL CLI 処理がすでに完了しているので、関数 *VSAMSQLFreeHandle()* を使用して、VSAMSQL CLI 環境を割り振り解放できます。 これは、割り振りステップ (234 ページの『ステップ 2: VSAMSQL CLI 環境の初期化』) の逆順を実行することによって行えます。

```
                              ...
/*****************************************************************/
/*      free VSAMSQL CLI Environment                             */
/*****************************************************************/
// free handle Statement
rc = VSAMSQLFreeHandle(VSAMSQL_HANDLE_STMT,hStmt);
cont = check_error(VSAMSQL_HANDLE_STMT,hStmt,rc,
```

```
                    o_sqlstate,o_message,&o_native_error,SYSLST,"VSAMSEL-E");

// free handle Connection
rc = VSAMSQLFreeHandle(VSAMSQL_HANDLE_DBC,hDBC);
cont = check_error(VSAMSQL_HANDLE_DBC,hDBC,rc,
                    o_sqlstate,o_message,&o_native_error,SYSLST,"VSAMSEL-F");

// free handle Environment
rc = VSAMSQLFreeHandle(VSAMSQL_HANDLE_ENV,hEnv);
cont = check_error(VSAMSQL_HANDLE_ENV,hEnv,rc,
                    o_sqlstate,o_message,&o_native_error,SYSLST,"VSAMSEL-10");
...
```

ステップ 6: ホスト出力変数へのローカル出力変数の割り当て

最終ステップでは、クライアント・プログラムに戻すローカル出力変数を、対応するホスト出力変数に割り当てる必要があります。

```
...
/*****************************************************************/
/*  assign number rows with valid data, if >=4 more data exists */
/*****************************************************************/
*(VSAMSQLINTEGER *)argv[2] = o_resultrows;       // rows returned

// copy result rows to output parameters
j=0;
for (i=0; i < NUM_ROWS; i++)
{
   *(VSAMSQLINTEGER *)argv[j+3] = o_parm[i].articleno;
   strcpy(argv[j+4], o_parm[i].manufacturer);
   strcpy(argv[j+5], o_parm[i].type);
   strcpy(argv[j+6], o_parm[i].model);
   *(VSAMSQLINTEGER *)argv[j+7] = o_parm[i].price;
   j = j + 5;
} // end for
...
```

サンプル DL/I アプレットの実行

このトピックの説明は、VSE コネクター・クライアント で提供されている *DL/I アプレット* というサンプル・アプレットを基にしています。

VSE コネクター・クライアント で同様に提供されているこのほかのアプレットの例には、次のものがあります。

- *VsamSpaceUsage* アプレット (使用されている VSAM スペースおよび空き VSAM スペースを表示する)。
- データ・マッピング・アプレット (212 ページの『データ・マッピング・アプレットの説明』 ページに説明があります)。
- *VSAM* アプレット (222 ページの『サンプル VSAM アプレットの実行』 ページに説明があります)。

DL/I アプレット の説明

サンプルの DL/I アプレットは、Db2 ベース・コネクターと一緒にアプレットを使用する方法を説明する例で、この例を使用すると、サンプルの DL/I データベースに保管されている DL/I データを表示し、変更することができます。

DL/I アプレットは、開始されると、Db2 Connect データベース別名 **db2vsewm** に接続されます。 これによって、DL/I アプレットは、**db2vsewm** を介して、z/VSE ホストのデータベース **sqlds** と通信できるようになります。

次に、DL/I アプレットは、サンプルの Db2 ストアード・プロシージャー (この例では DLIREAD) を呼び出し、*AIBTDLI* インターフェース を介して DL/I データにアクセスします。 AIBTDLI インターフェースの説明については 460 ページの『AIBTDLI インターフェースの概要』を参照してください。

DL/I アプレットを実行するには、以下のものをカスタマイズしておく必要があります。

- Db2 ベース・コネクター (詳しくは 95 ページの『第 10 章 Db2 ベース・コネクターのカスタマイズ』のステップ 1 から 6 を参照)。
- Db2 ストアード・プロシージャーのサンプル (詳しくは 108 ページの『ステップ 7: VSAM データ・アクセス用に Db2 ベース・コネクターをカスタマイズする』を参照)。
- ご使用の物理/論理中間層上の Db2 Connect (詳しくは 110 ページの『ステップ 10: Db2 Connect のインストールとクライアント/ホスト接続の確立』を参照)。このステップで、**sqlds** を、ご使用の物理/論理中間層上の Db2 Connect に定義します (ここで **sqlds** は、VSAM アプレットで使用される z/VSE ホスト上のサンプル・データベースです)。

注**:** DL/I アプレット は、オペレーティング・システムと Web ブラウザーの特定の組み合わせによっては実行できない場合があります。

3 層環境内のサンプル DL/I アプレットの使用方法を図 115 に示します。

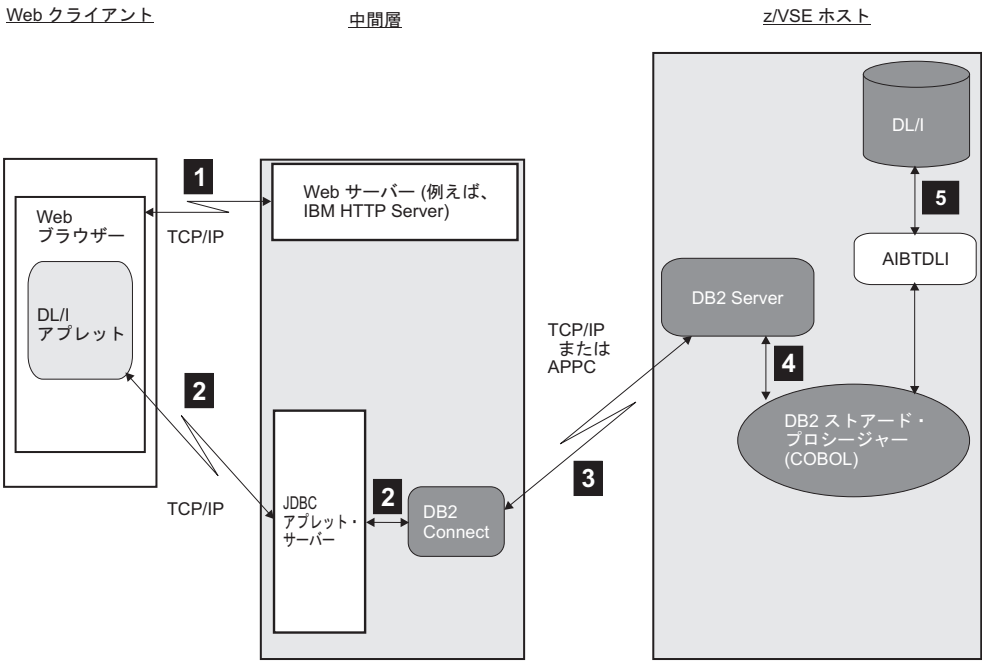


図 *115. DL/I* データにアクセスするためのサンプル *DL/I* アプレット の使用

1. クライアントの Web ブラウザーが、物理/論理中間層上で実行されている IBM HTTP Server (または別の Web サーバー) に HTML ページを要求します。 HTML ページには、*DL/I* アプレット のアプレット・タグが入っています。 DL/I アプレットは Web ブラウザーの Java 仮想マシンにロードされ、実行を開始します。
2. DL/I アプレットは、物理/論理中間層で実行されている Db2 Connect への接続をオープンします。これは、*JDBC* アプレット・サーバー (「JDBC」は *Java Database Connectivity* の省略形) を使用して行われます。 JDBC アプレット・サーバーを使用すると、以下のアプレットの制約事項に関係なく行えます。
   - アプレットが新規ネットワーク接続をオープンできるのは、アプレットがダウンロードされた元のプラットフォーム (この場合は物理/論理中間層) に対してだけです。
   - アプレットがアクセスできるのは、アプレットがダウンロードされた元のプラットフォーム (この場合は物理/論理中間層) のファイル・システムだけです。

   次に、DL/I アプレットは、サンプル Db2 ストアード・プロシージャー (この例では DLIREAD) を呼び出します。
3. Db2 Connect は、Db2 Connect データベース別名 **db2vsewm** を使用して、Db2 Server for VSE と通信します。 現在、Db2 Connect は z/VSE ホスト データベース **sqlds** にアクセスできます。DRDA (分散リレーショナル・データベース体系) を使用します。基礎のプロトコルは APPC または TCP/IP のどちらでもかまいません。
4. Db2 Server for VSE が、ストアード・プロシージャー・サーバーを使用して、サンプル Db2 ストアード・プロシージャー (この例では DLIREAD) の実行を管理します。
5. これによって、Db2 ストアード・プロシージャーは、z/VSE ホストに保管されている DL/I データに *AIBTDLI* インターフェース (460 ページの『AIBTDLI インターフェースの概要』に説明があります) を使用してアクセスすることによって、DL/I アプレットの要求を実行できるようになります。

注:

1. WebSphere Application Server は、上記のプロセスの物理/論理中間層では必要ありません。
2. VSE コネクター・サーバー は、上記のプロセスの z/VSE ホストでは必要ありません。

サンプル DL/I アプレットの開始

DL/I アプレットを実行できるようにするには、このトピックで説明するステップを実行する必要があります。

1. DL/I アプレットを呼び出すための HTML ファイルの作成

HTML ファイルを作成し、そこから DL/I アプレットを呼び出せるようにしなければなりません。 この HTML ページが表示されるたびに、Web ブラウザーの

JVM (Java 仮想マシン) にアプレットがロードされて実行されます。 これによって、アプレットが、JDBC を使用して物理/論理中間層のデータベースに接続されます。

次に、そのような HTML ファイルの例を示します。

```
<html>
<head>
<title>
JDBC Example Applet that accesses DL/I using the Db2-based connector
</title>
</head>
<body>
<h2>JDBC Example Applet that accesses DL/I data</h2>
This applet lets you browse, insert, update and delete segments from/into the
DL/I inventory sample database via the Db2-based connector.
The JDBC Applet calls the corresponding Stored Procedures defined
within your Db2 Server for VSE using JDBC interface.
The Stored Procedures use the AIBTDLI interface to access the DLI/VSE data.
<p>
<center>
<applet code="DB2DLIConnectorJDBCApplet.class"
           archive="db2dliapplt.jar, db2java.zip"
           width=440 height=420>
</applet>
</center>
</body>
</html>
```

2. DLIREAD.C のコンパイル

AIBTDLI インターフェースを使用する Db2 ストアード・プロシージャーは *COBOL* で作成されているので、IBM LE/VSE *COBOL* コンパイラーを使用して DLIREAD.C (この例で使用する Db2 ストアード・プロシージャー) をコンパイルします。コンパイル・ジョブ・ストリーム SKDLICMP は、ICCF ライブラリー 59 にあります。

DLIREAD.C には SQL ステートメントがありません。代わりに、AIBTDLI インターフェースを使用します。したがって、SQL プリコンパイラーを実行する必要はありません。 ただし、ユーザー独自のデータにアクセスするために SQL ステートメントおよび DL/I 呼び出しの両方を組み込むことにした場合は、追加の SQL プリコンパイル・ステップを実行する必要があります。

DL/I データにアクセスするために使用されるその他のサンプル Db2 ストアード・プロシージャー (DLIUPINS.C および DLIDEL.C) をコンパイルするには、同じスケルトン SKDLICMP を使用します。

3. Db2 Server for VSE への DLIREAD の定義

ここでは、DLIREAD を Db2 Server for VSE システムに定義します。 これを行うには、ジョブ・スケルトン SKDLISTP (VSE/ICCF ライブラリー 59 にあります) を使用して、DL/I サンプル・データベースにアクセスするために使用する Db2 ストアード・プロシージャーを作成します。

1. DLIREAD フェーズ (前のステップでコンパイルされている) を、ストアード・プロシージャー・サーバー の検索パスに入っているライブラリーの中に入れます。

2. CREATE PROCEDURE ステートメントを使用して、DLIREAD をデータベース・マネージャーに定義します。この目的のために、ICCF ライブラリー 59 に入っているジョブ・ストリーム SKDLISTP を使用できます。

   次に、DLIREAD の CREATE PROCEDURE ステートメントの例を示します。

```
CREATE PROCEDURE DLIREAD (INOUT CHAR(6),OUT SMALLINT,
                          OUT CHAR(6),OUT CHAR(25),
                          OUT INT,OUT CHAR(6),OUT CHAR(6),
                          OUT CHAR(6),OUT CHAR(25),
                          OUT INT,OUT CHAR(6),OUT CHAR(6),
                          OUT CHAR(6),OUT CHAR(25),
                          OUT INT,OUT CHAR(6),OUT CHAR(6),
                          OUT CHAR(4),OUT CHAR(120))
                          EXTERNAL,LANGUAGE COBOL,
                          STAY RESIDENT YES,
                          SERVER GROUP,
                          PARAMETER STYLE GENERAL;
```

4. DL/I データベースの定義

DL/I アプレットを使用できるようにするには、DL/I データベースを定義しておかなければなりません。 サンプルの DL/I データベースを定義してロードするには、ジョブ SKDLISMP (ICCF ライブラリー 59 にあります) を使用します。

5. DL/I アプレット用の JAR ファイルの作成

DL/I アプレットを実行する前に、以下を実行する必要があります。

1. アプレットに関連するクラス・ファイルをこの JAR (Java アーカイブ) ファイルにコピーすることによって JAR ファイルを作成します。 これを行うには、ご使用の VSE コネクター・クライアント・システムのサンプルのディレクトリーに変更し、以下のステートメントを実行します。

```
call jar c0fv db2dliapplt.jar com¥ibm¥vse¥db2¥DB2DLIConnectorJDBCApplet.class
   com¥ibm¥vse¥db2¥MessageDialog.class
   com¥ibm¥vse¥db2¥PowerGridLayout.class
   com¥ibm¥vse¥db2¥PowerGridLayoutInfo.class
```

2. 上記の (1.) の JAR ファイルを、物理/論理中間層プラットフォーム上のご使用の Web サーバー (例えば、IBM HTTP サーバー、または Apache サーバー) の HTML ディレクトリーにコピーします。

   注: DL/I アプレットは、上記 (1.) の代わりの方法として *3* 層環境 で実行されるので、Web サーバー上の HTML ディレクトリーにクラス・ファイルを直接コピーできます。これは、アプレット・コードが z/VSE ホストに保管されておらず、したがって、クラス・ファイルをショート・ネーム・アーカイブに保管する必要がないからです。

3. JDBC アプレット・サーバーを物理/論理中間層プラットフォームで開始します。 このサーバーは、DL/I アプレットによって開始された要求を処理します。 また、ユーザーは、JDBC アプレット・サーバーが使用できる未使用の TCP/IP ポート番号を選択する必要があります。 したがって、TCP/IP ポート **6789** (デフォルト) を選択する場合、次のように入力します。

```
db2jstrt 6789
```

DL/I アプレットの呼び出し

HTML ページがロードされると、DL/I アプレットは、次のように処理されます。

1. 物理/論理中間層サーバーからローカル・ワークステーションにダウンロードされる。
2. ローカル・ワークステーションにインストールされている Web ブラウザーの Java 仮想マシン内で実行されなければなりません。

DL/I アプレットは、次の 2 つの方法で呼び出すことができます。

- ご使用のローカル Java システムの一部であるアプレット・ビューアー (Windows および OS/2 ではファイル **AppletViewer.exe**) を直接使用する方法。これを行うには、コマンド・プロンプトで次のように入力します。

  ```
  AppletViewer index.html
  ```

  ここで、**index.html** には、 239 ページの『1. DL/I アプレットを呼び出すための HTML ファイルの作成』に示されている HTML タグが入ります。
- Web ブラウザーを使用する方法。 この方法を実行するには、ご使用の物理/論理中間層プラットフォームのシンボル名または IP アドレスを入力し、その後に、ご使用の Web ブラウザーのアドレス・フィールドの中のサンプル HTML ファイルの名前を続けます。 この例の場合は、以下のように入力します。

  ```
  http://ebusvse/db2dli/index.html
  ```

DL/I アプレットは、呼び出されると、 243 ページの図 116 に示すメイン・ウィンドウを表示します。

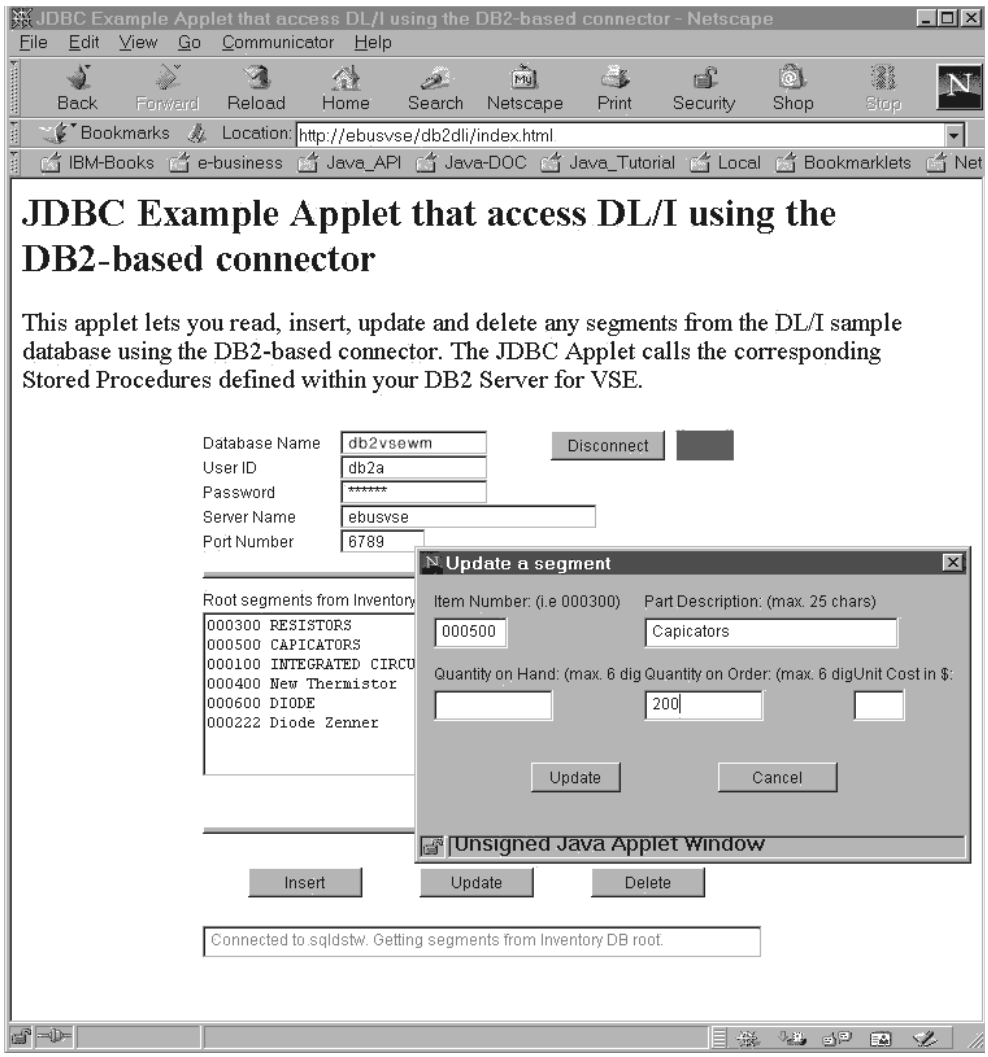


図 *116. サンプル DL/I アプレットによって表示されるウィンドウ*

図 116 には、**Disconnect** ボタンの右に長方形があります。 この長方形は、DL/I アプレットと z/VSE ホスト (Db2 Connect データベース別名 **db2vsewm** を使用してアクセスされる) 上の **sqlds** サンプル・データベースとの間の接続の状況を示します。 この長方形が、

- 緑色 であるときは、アプレットは **sqlds** に接続されていて、ボタンには **Disconnect** が表示されます。
- 赤色 であるときは、アプレットは **sqlds** に接続されておらず、ボタンには **Connect** が表示されます。

図 116 に示したメイン・ウィンドウには、「*Update a segment* (セグメントを更新する)」というダイアログ・ウィンドウが組み込まれており、ユーザーはこのダイアログ・ウィンドウを使用して、DL/I セグメントをサンプル DL/I データベースに挿入します。 ユーザーは、その他の同様のダイアログ・ウィンドウを使用して、サンプル DL/I データベース内のセグメントを最新表示、挿入、あるいは削除することができます。 現在表示されているダイアログ・ウィンドウに応じて、以下のことを行うことができます。

押すボタン
: 実行される内容 ...

Connect
: Db2 Connect を使用して z/VSE への接続が再確立され、次に Db2 ストアード・プロシージャー **DLIREAD** を使用して、サンプル DL/I データベース内のすべてのセグメントが読み取られます。

Insert Db2 ストアード・プロシージャー **DLIUPINS** が使用され、新規セグメントがサンプル DL/I データベースに挿入されます。

Update
: Db2 ストアード・プロシージャー **DLIUPINS** が使用され、サンプル DL/I データベース内のセグメントが更新されます。

Delete
: Db2 ストアード・プロシージャー **DLIDEL** が使用され、サンプル DL/I データベース内のセグメントが削除されます。

Refresh
: Db2 ストアード・プロシージャー **DLIREAD** が使用され、サンプル DL/I データベース内のすべてのセグメントが読み取られます。

243 ページの図 116 の最下部には「状況表示行」があり、ここに、ユーザーが実行中のアクションに関連したエラー・メッセージまたは状況メッセージが表示されます。

DB2DLIConnectorJDBCApplet.java (クライアント・サイド・プログラム) の説明

このトピックでは、DL/I アプレットの機能のほとんどで使用される *DB2DLIConnectorJDBCApplet.java* のメイン・ステップ について説明します。これは、 238 ページの図 115 の Web ブラウザーの JVM で実行されます。

さらに、メソッド *callStpDLIREAD()* が、DL/I データベースの読み取り/ブラウズを実行する DL/I アプレットの一部になっています。 入力変数および出力変数はすべて、該当するパラメーターのプレースホルダー (DL/I アプレットのサンプルに疑問符 (?) として示されています) に設定する必要があります。

ステップ 1: DLIREAD を呼び出すための SQL ステートメントの準備

最初のステップを実行できるようにするには、接続オブジェクト *con* が、z/VSE ホストに対して存在していなければなりません。 この接続オブジェクトは、メソッド *connectToDB()* (232 ページの『ステップ 4: Db2 Connect を使用した、z/VSE データベースへの接続の確立』に説明があります) を使用してオープンされます。

```
public void callStpDLIREAD()
{
   CallableStatement stmt; // SQL Statement Handle
   String sql =            // JDBC Stored Procedure Call String
       "Call " + name + "(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)";

   // Prepare Stored Procedure Call Statement
```

```
try
{
   stmt = con.prepareCall(sql);
   // add header lines into applet list box
   RecordList.add("ITEMNO ITEM-DESCRIPTION          QUAN-H QUAN-O PRICE");
   RecordList.add("==================================================");
   do
   {
      //** set input variables **
      stmt.setString (1, nextkey);

      //** register output parameters **
      // next key returned from procedure
      stmt.registerOutParameter(1, Types.CHAR);
      stmt.registerOutParameter(2, Types.SMALLINT);

      // 3 data rows that can hold data from root segment of
      // inventory DB
     for (int i=3; i <= 13; i=i+5)
     {
        stmt.registerOutParameter ( i  , Types.CHAR);
        stmt.registerOutParameter ( i+1, Types.CHAR);
        stmt.registerOutParameter ( i+2, Types.INTEGER);
        stmt.registerOutParameter ( i+3, Types.CHAR);
        stmt.registerOutParameter ( i+4, Types.CHAR);
     }

     // Variables for error handling
     stmt.registerOutParameter(18, Types.CHAR); // return code
     stmt.registerOutParameter(19, Types.CHAR); // error message
```

ステップ 2: DLIREAD の呼び出し

2 番目のステップでは、ステップ 1 で作成されたステートメントを実行することによって、DLIREAD (この例で使用するサンプル Db2 ストアード・プロシージャー) が呼び出されます。

```
stmt.executeQuery();
```

ステップ 3: DLIREAD からの戻りコードの検査

3 番目のステップでは、DLIREAD によって戻された戻りコードが検査されます。

```
// Get return code from Stored Procedure
ret_code = stmt.getString(18);

// Check if the Stored Procedure returned an error
if ((ret_code.compareTo("0000") == 0) ||
    (ret_code.substring(0,1).compareTo("G") == 0))
{
   // no error
   // get number of result rows (1-3)
   res_rows  = stmt.getShort(2);
```

エラーが検出されなかった場合は、出力変数が DLIREAD からリトリーブされます。 出力パラメーターが読み取られ、次に、結果が DL/I アプレットによって表示されます。

```
// Get returned fields from root segment
for (int i = 0, j = 3; i < 3;  i++, j=j+5)
{
   itemno[i]       = stmt.getString(j);
   itemdesc[i]     = stmt.getString(j+1);
   unit_cost[i]    = stmt.getInt(j+2);
   quan_hand[i]    = stmt.getString(j+3);
   quan_ord[i]     = stmt.getString(j+4);
```

```
   // add fields from root segment into listbox
   if (i < res_rows)
   {
      String temp = Integer.toString(unit_cost[i]);
      while(temp.length()<5)
         temp += ' ';
      RecordList.add(itemno[i] + " " + itemdesc[i]
          + " " + quan_hand[i] + " "
          + quan_ord[i] + " " + temp + "$ ");
   }
}
```

プログラムは、さらに、次のことを行います。

1. まだ結果があるかどうかを検査します。 まだ結果がある 場合、DLIREAD 呼び出しが行われます。
2. *nextkey* 変数を使用して、さらにデータ結果行があるかどうかを検査します。 さらにデータ行がある 場合、次の 3 行に対して DLIREAD が再び呼び出されます。

```
     nextkey    = stmt.getString(1);
     if (nextkey == "000000")
        moreresults = false;
     else  // more data available in the database
        moreresults = true;
  }
  else
  {
     // error occured
     // check if DB is empty
    if ((ret_code.substring(0,2) == "GB") &&
        (RecordList.getItemCount() == 0))
       setStatus("Inventory Database is empty!!");
    else {
       DLIerrmsg = stmt.getString(19);  // get DLI error message
       setStatus("AIBTDLI Return Code: 0x" + ret_code);
       msgDialog.setTitle("Stored Procedure Error");
       System.out.println("DLI Error: " + DLIerrmsg);
       msgDialog.setMessage("Check Java Console for DLZ messages.");
       msgDialog.setVisible(true);
    } // end if
  } // end if
}
while ( moreresults && ret_code.compareTo("0000") == 0);
```

ステップ 4: sqlds データベースへの接続のリセット

最終ステップでは、エンド・ユーザーが 243 ページの図 116 の **Disconnect** ボタンを押すたびに、プログラムは、接続を、z/VSE ホストにある **sqlds** データベースにリセットします。 **sqlds** は、Db2 Connect のデータベース別名 **db2vsewm** を介してアクセスされます。

```
       if ((ret_code.compareTo("0000") == 0) ||
           (ret_code.substring(0,2).compareTo("GB") == 0))   // no error
       {
          setStatus(RecordList.getItemCount()-2 +
                   " segments retrieved from Inventory DB.");
       }

       // close Statement Handle
       stmt.close();
     }
     catch (SQLException sqlExc)   // handle SQL exceptions
     {
```

```
        closeConnection();
        ...
        return;
      }
    }
   ...
}
...
```

DL/I セグメントの挿入、更新、または削除の方法の詳細については、VSE コネクター・クライアント のオンライン・ドキュメンテーションで提供されている DL/I アプレットの完全ソース・コードを参照してください。 (オンライン・ドキュメンテーションについては 30 ページの『オンライン・ドキュメンテーション・オプションの使用』に説明があります。)

DLIREAD の説明

このトピックでは、この例で使用するサンプルの Db2 ストアード・プロシージャーである DLIREAD のメイン・ステップ について説明します。これは COBOL で作成されており、*AIBTDLI* インターフェース (460 ページの『AIBTDLI インターフェースの概要』に説明があります) を介して DL/I 呼び出しを実行します。 DLIREAD は 238 ページの図 115 の z/VSE ホストで実行され、 243 ページの図 116 の「Connect」ボタンまたは「Refresh」ボタンが押されたときに呼び出されます。

完全ソース・コードは、VSE コネクター・クライアント と一緒に提供されるファイル DLIREAD.COB に入っています。 このほかのサンプル Db2 ストアード・プロシージャー (DLIUPINS および DLIDEL) は、このトピックでは説明していません。

DLIREAD を 1 回呼び出すことにより、サンプルの DL/I データベースから最大 3 つのセグメントを読み取り、戻すことができます。

- データは、DLIREAD の出力パラメーターを使用して戻されます。この場合、DL/I セグメントの各データ・フィールドには、対応する出力パラメーターがあります。
- このほかの出力パラメーターにはエラー標識変数 があり、ここには、Db2 または DL/I で起こったエラーに関する情報が入るか、あるいは、Db2 ストアード・プロシージャーからの要求の状況が示されます。
- DL/I アプレット には、取得されるのを待っている DL/I セグメントがさらに (3 つよりも多く)あるかどうかを検査するために使用する変数があります。リトリーブする必要があるレコードが 3 つより多くある 場合は、クライアント・プログラムは昇順キーを使用して再び DLIREAD を呼び出します。

ステップ 1: AIBTDLI 用の変数および入出力域の定義

最初のステップで、以下のものを定義します。

- DL/I AIBTDLI インターフェースの変数。
- DL/I と通信するために使用する入出力域。

```
...
*
*  UNQUALIFIED SSA FOR GETTING FIRST ROOT SEGMENT
*
   77  SSAUNQ        PIC X(9)  VALUE 'STPIITM  '.
```

```
*
*  QUALIFIED SSA FOR GETTING ROOT SEGMENT VIA KEY
*
   01  SSAQUAL.
     02  FILLER    PIC X(19) VALUE 'STPIITM (STQIINO = '.
     02  ROOTKEY   PIC X(6).
     02  FILLER    PIC X(1)  VALUE ')'.
*
* I-O AREA FOR RECEIVING ALL SEGMENTS
*
   01  IOAREA        PIC X(160).
   01  STPIITM         REDEFINES IOAREA.
     02  ITNUMB    PIC X(6).
     02  ITDESC    PIC X(25).
     02  IQOH      PIC X(6).
     02  IQOR      PIC X(6).
     02  FILLER    PIC X(6).
     02  IUNIT     PIC 9(6).
     02  FILLER    PIC X(105).
  LINKAGE SECTION.
```

ステップ 2: DLIREAD 用のパラメーターの定義

2 番目のステップで、DLIREAD で使用するパラメーターを定義します。 これらのパラメーターは、DL/I アプレットと、z/VSE ホスト上で実行される Db2 Server for VSE との間のインターフェースを作成します。

次に示す例では、3 つの行のパラメーターが定義されます。このようにすると、DLIREAD を 1 回呼び出すことにより、DL/I のサンプル・データベースから最大 3 つの行をリトリーブできます。

```
 *
 * STORED PROCEDURE PARAMETERS
 *
   01  NEXT-KEY     PIC X(6).
   01  NUM-ROWS     PIC S9(4)   COMP.

 * FIRST RESULT ROW
   01  ITEM-NUMB1   PIC X(6).
   01  ITEM-DESC1   PIC X(25).
   01  UNIT-COST1   PIC S9(6)   COMP.
   01  QUAN-HAND1   PIC X(6).
   01  QUAN-ORD1    PIC X(6).

 * SECOND RESULT ROW
   01  ITEM-NUMB2   PIC X(6).
   01  ITEM-DESC2   PIC X(25).
   01  UNIT-COST2   PIC S9(6)   COMP.
   01  QUAN-HAND2   PIC X(6).
   01  QUAN-ORD2    PIC X(6).

 * THIRD RESULT ROW
   01  ITEM-NUMB3   PIC X(6).
   01  ITEM-DESC3   PIC X(25).
   01  UNIT-COST3   PIC S9(6)   COMP.
   01  QUAN-HAND3   PIC X(6).
   01  QUAN-ORD3    PIC X(6).
   01  RET-CODE     PIC X(4).
   01  DLI-ERR-MSG  PIC X(120).
   COPY DLIAIB.
   ...
```

ステップ 3: COBOL への DLIREAD のパラメーターの定義

3 番目のステップでは、DLIREAD の PROCEDURE DIVISION で、DLIREAD への外部インターフェースを表すパラメーターを定義します。

```
 ....
*****************************
* BEGIN OF PROGRAMMING SECTION
*****************************
  PROCEDURE DIVISION USING
            NEXT-KEY NUM-ROWS
            ITEM-NUMB1 ITEM-DESC1
            UNIT-COST1 QUAN-HAND1
            QUAN-ORD1
            ITEM-NUMB2 ITEM-DESC2
            UNIT-COST2 QUAN-HAND2
            QUAN-ORD2
            ITEM-NUMB3 ITEM-DESC3
            UNIT-COST3 QUAN-HAND3
            QUAN-ORD3
            RET-CODE DLI-ERR-MSG.
```

ステップ 4: 照会ステートメントの結果の取得

4 番目のステップでは、DL/I 処理が次のように開始されます。

1. DLIREAD が、セグメントをリトリーブするために、最大 3 つの DL/I 呼び出しを実行します。
2. リトリーブするセグメントがこれ以上ない場合は、3 番目の呼び出しが行われた後で、処理が終了します。
3. リトリーブする結果行がある場合、DLIREAD は、再び 3 回呼び出されます。

```
* ******************************************************
* SCHEDULE THE PSB
* ******************************************************
   SCHEDULE-PSB.
   MOVE 'STBICLG' TO PSB-NAME.
   CALL 'AIBTDLI' USING FUNCT-PCB, PSB-NAME, ADDRESS OF DLIAIB.
   IF AIBFCTR NOT = LOW-VALUES  GO TO BASERR
   SET ADDRESS OF PCB-PTRS TO AIBPCBAL.
   SET ADDRESS OF INV-L-PCB TO B-PCB1-PTRS.

* ******************************************************
* DO A GET NEXT CALL TO RETRIEVE THE FIRST ROOT SEGMENT
* ******************************************************
   MOVE NEXT-KEY TO ROOTKEY.
   IF NEXT-KEY = '000000'
      CALL 'AIBTDLI' USING FUNCT-GN, INV-L-PCB, IOAREA, SSAUNQ
```

DL/I への呼び出しは、最大 3 つのセグメントに対して行われます。 値は、DLIREAD 用に定義された対応パラメーターを使用して DL/I アプレットに戻されます。

```
...
* ******************************************************
* DO A GET UNIQUE CALL TO RETRIEVE A SEGMENT VIA KEY
* ******************************************************
   ELSE
     CALL 'AIBTDLI' USING FUNCT-GU, INV-L-PCB, IOAREA, SSAQUAL
   END-IF.
   IF AIBFCTR NOT = LOW-VALUES
     GO TO BASERR.

* ******************************************************
* ASSIGN VALUES TO OUTPUT PARAMETERS FOR ROW 1
```

```
* ****************************************************
   ADD 1 TO COUNTR.
   MOVE ITNUMB  TO ITEM-NUMB1.
   MOVE ITDESC  TO ITEM-DESC1.
   MOVE IQOH    TO QUAN-ORD1.
   MOVE IQOR    TO QUAN-HAND1.
   MOVE IUNIT   TO UNIT-COST1.
   MOVE COUNTR TO NUM-ROWS.
```

2 番目の呼び出しをストアード・プロシージャーに出して
2 番目の結果行をリトリーブします。

```
* ****************************************************
* ISSUE GET NEXT CALL
* ****************************************************
 CALL 'AIBTDLI' USING FUNCT-GN, INV-L-PCB, IOAREA,
        SSAUNQ.
 IF AIBFCTR NOT = LOW-VALUES
   GO TO BASERR.

* ****************************************************
* ASSIGN VALUES TO OUTPUT PARAMETERS FOR ROW 2
* ****************************************************
  ADD 1 TO COUNTR.
  MOVE ITNUMB  TO ITEM-NUMB2.
  MOVE ITDESC  TO ITEM-DESC2.
  MOVE IQOH    TO QUAN-ORD2.
  MOVE IQOR    TO QUAN-HAND2.
  MOVE IUNIT   TO UNIT-COST2.
  MOVE COUNTR TO NUM-ROWS.
```

次に示すコードで、DLIREAD への 3 番目の (最終の) 呼び出しが実行されます。
3 番目の結果行がリトリーブされます。

```
* ****************************************************
* ISSUE GET NEXT CALL
* ****************************************************
  CALL 'AIBTDLI' USING FUNCT-GN, INV-L-PCB, IOAREA, SSAUNQ.
  IF AIBFCTR NOT = LOW-VALUES
    GO TO BASERR.

* ****************************************************
* ASSIGN VALUES TO OUTPUT PARAMETERS FOR ROW 3
* ****************************************************
  ADD 1 TO COUNTR.
  MOVE ITNUMB  TO ITEM-NUMB3.
  MOVE ITDESC  TO ITEM-DESC3.
  MOVE IQOH    TO QUAN-ORD3.
  MOVE IQOR    TO QUAN-HAND3.
  MOVE IUNIT   TO UNIT-COST3.
  MOVE COUNTR TO NUM-ROWS.
```

ステップ 5: さらに DL/I セグメントがあるかどうかの検査

5 番目のステップでは、さらに DL/I セグメントがあるかどうかが検査されます。
変数 NEXTKEY が、初期条件を満たす次の DL/I セグメント・キーにセットされます。

```
* ****************************************************
*  DETERMINE IF FURTHER SEGMENTS EXIST AND SAVE KEY
* ****************************************************
  CALL 'AIBTDLI' USING FUNCT-GN, INV-L-PCB, IOAREA, SSAUNQ.*
```

```
 * MORE SEGMENTS AVAIL - SAVE KEY FOR THE NEXT PROCEDURE CALL
 *
      MOVE INV-KEY-FBCK(1:6) TO NEXT-KEY.
      GO TO END-PROC.
```

ステップ 6: エラー処理ルーチンの実行

最終ステップでは、DLIREAD はエラー処理ルーチンを使用して、要求が正常に処理されたかどうかを判別します。

```
*
 *  DLI-CALL ERROR HANDING
  ...
```

第 18 章 Java サーブレットによるデータへのアクセス

このトピックに含まれるのは次のとおりです。

- 『3 層環境内でのサーブレットの使用方法』
- 255 ページの『サーブレットのコンパイルおよび呼び出し』
- 256 ページの『サーブレットをインプリメントする方法の例』

3 層環境内でのサーブレットの使用方法

サーブレットは、3 層環境の物理/論理中間層では、図 117 で示されているように使用されます。

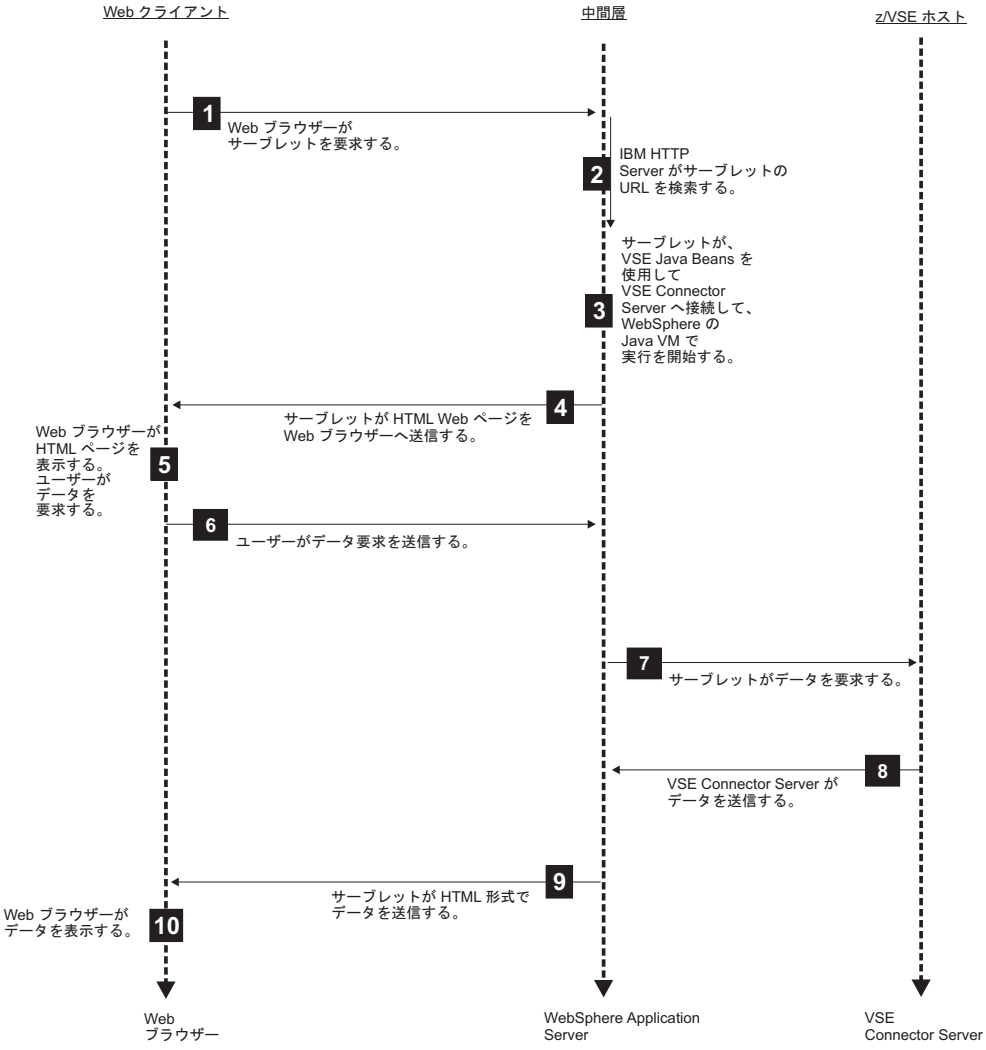


図 *117. z/VSE 3* 層環境内でのサーブレットの使用方法

図 117 では、z/VSE ホスト上で稼働する VSE コネクター・サーバーが、データ取得に使用されることが前提になっています。

Web クライアントと物理/論理中間層との間のデータの送受信には HTTP セッションが使用されます。 物理/論理中間層と z/VSE ホストとの間のデータの送受信にはソケット接続が使用されます。

以下の各リスト項目の番号は、 253 ページの図 117 に示されているステップを説明しています。

1. クライアントの Web ブラウザーが、物理/論理中間層上で実行されている IBM HTTP Server に、サーブレットの要求を送信します。
2. IBM HTTP Server がサーブレットの URL をリトリーブし、これを WebSphere Application Server に渡します。
3. WebSphere Application Server が、自身の Java 仮想マシン (JVM) 内でサーブレットを実行します。 また、サーブレットは、VSE Java Beans のクラス・ライブラリー (**VSEConnector.jar**) を使用して、z/VSE ホスト上で実行されている VSE コネクター・サーバー への接続を構築します。
4. サーブレットは、動的に作成された HTML Web ページを、クライアントの Web ブラウザーに TCP/IP を使用して送信します。
5. クライアントの Web ブラウザーが Web ページを表示します。 これにより、エンド・ユーザーは、z/VSE ホストに保管されているデータを要求できるようになります。
6. クライアントの Web ブラウザーは、データに対する要求を、WebSphere Application Server の Java 仮想マシンにまだロードされているサーブレットに送信します。
7. サーブレットは VSE Java Beans を使用して z/VSE ホスト上で実行されている VSE コネクター・サーバー と通信を行い、VSE ベースのデータを要求します。
8. VSE コネクター・サーバー はデータを取得し、それをサーブレットに送信します。

   注**:**

   1. VSE コネクター・サーバー は、VSE/VSAM、VSE/POWER、VSE/ICCF、またはライブラリアン・データにアクセスするために使用できます。
   2. z/VSE ホストに保管されている VSAM データにアクセスする代替メソッドは、z/VSE ホスト上の VSAM ファイル・システムと通信を行っている物理/論理中間層上の Db2 ストアード・プロシージャーを使用する方法です。このことについては、 451 ページの『Db2 ストアード・プロシージャーによる VSAM データへのアクセス』に説明があります。
9. サーブレットは、HTML コードから成る動的 Web ページを生成し、この Web ページを、要求されたデータと一緒に、クライアントの Web ブラウザーに送信して戻します。
10. クライアントの Web ブラウザーが、要求データと一緒に Web ページを再表示します。

サーブレットのコンパイルおよび呼び出し

サーブレットをコンパイルするには、サーブレットに関連した Java クラスが必要です。 通常、これらのクラスは、ご使用の WebSphere Application Server システムと一緒に提供されます。 これらのクラスは JAR ファイルに入っており、このファイルは、(ユーザーのコンパイル用の) ローカル・クラスパス、および WebSphere Application Server のクラスパスに組み込む必要があります。

これで、サーブレットは、さまざまな方法で呼び出すことができるようになりました。 このトピックでは、以下の、最もよく使用されている方法を示します。

コマンド行から

次のように入力するだけで済みます。

```
http://server_host_name/servlet_engine_name/servlet_name
```

例えば、ご使用の Web ブラウザーのコマンド行で、次のように入力します。

```
http://MyComputer/webapp/VseServletEngine/MyServlet
http://9.164.123.456/webapp/VseServletEngine/MyServlet
```

WebSphere Application Server がセッション情報を保管する方法

以下の TCP/IP 接続は、一時的なものです。

- Web クライアントから物理/論理中間層プラットフォームまでの接続。
- 物理/論理中間層から z/VSE ホストまでの接続。

サーブレットの *HTTPSession* オブジェクトを使用すると、Web クライアントが一連の HTTP 要求にわたって識別されるようになります。 WebSphere は、これを、例えば Web クライアントに「Cookie」を保管することによって、内部的に実行します。

サーブレットは、呼び出されるたびに、*HTTPSession* オブジェクト (これは「cookie」にバインドされている場合があります) を検索して、Web クライアントが過去にすでに接続したことがあったかどうかを検査します。 セッションがすでに存在している場合、サーブレットは、次に、そのセッションに、前に使用したセッションについての情報が入っているか検査します。

httpSession インスタンスは、サーブレットが初めて呼び出されたときに作成されます (これは、セッションがまだ存在していない場合です)。したがって、以下の 2 つのアクションを実行する必要があります。

1. httpSession インスタンスを作成し、それを HTTPServletRequest に保管する。
2. 特定のセッション・プロパティー (複数) を定義し、それらを httpSession に保管する。

256 ページの図 118 に、セッション情報が保管され、処理される方法についての「論理ビュー」を示します。 *httpSession* にはプロパティーが入っており、Web クライアント上に cookie として保管されます。

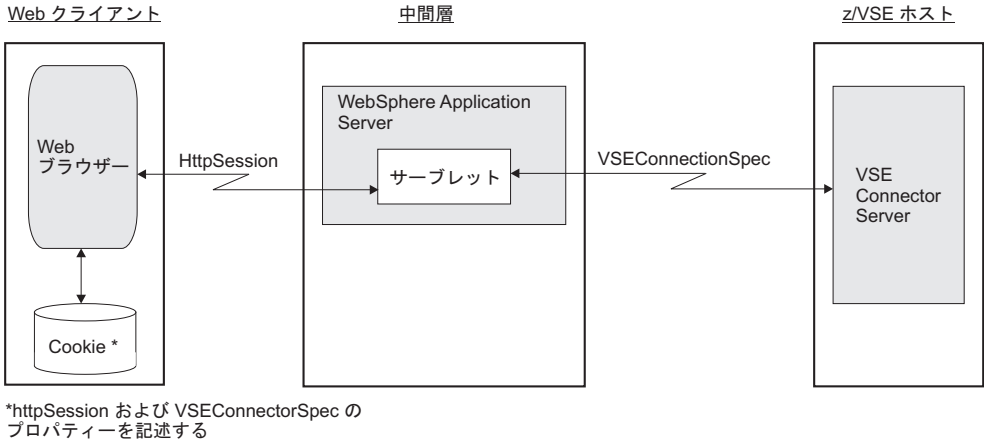


図 *118. WebSphere Application Server* によってセッション情報が再使用される方法

サーブレットをインプリメントする方法の例

このトピックでは、オンライン・ドキュメンテーションの一部として提供されている *FlightOrderingServlet* というサンプルについて説明します。 このサンプル・サーブレットでは、単純な VSAM ベースのフライト (航空便) 予約システムに、フライトと予約に関する情報を保守する機能が提供されます。 サーブレットは、データを保持する 2 つの VSAM クラスターをまず定義して、使用します。

この例で使用されるクラスターのデータ・マップとビューの定義方法の詳細については、 212 ページの『データ・マッピング・アプレットのサンプルの実行』を参照してください。

サンプル・サーブレットの一般説明

このサーブレットは、**FlightOrderingServlet.java** という Java ソース・ファイルの中にインプリメントされています。 このソース・ファイルには、フライトと予約を処理し、VSAM クラスターなどを作成するためのいくつかの内部クラスとともに、サーブレットのメイン・コードがインプリメントされています。

サーブレットは、以下の一般的な処理を実行します。

1. ユーザーがいくつかのアクションを選択するために使用する Web ページを表示します。この最初の Web ページで、ユーザーは、必要な VSAM クラスターを作成し、クラスターにデータを入れ、フライトを予約し、あるいは、フライトを取り消すことができます。
2. ユーザーが「**Order a flight (**フライトを予約する**)**」をクリックすると、使用可能なフライトを示す表が表示されます。 ここで、ユーザーはフライト番号を選択できます。フライト番号を選択すると、サーブレットは、このフライトのプロパティーが入っている別の Web ページ、および、ユーザーが予約を行うために必要なウィンドウ・コントロールを表示します。 ユーザーは、以下を入力します。

- 名前
- 予約する席数。
- 予約した席が禁煙席でなければならないかどうか。

3. フライトを予約するプッシュボタンをクリックすると、フライトの詳細情報が入っている VSAM レコードのプロパティーが更新されます。 さらに、新規レコードが ORDERS クラスターに作成されます。

以下のトピックでは、サンプル・サーブレットの Java コードのもっとも重要な部分について説明します。完全ソース・コードは、VSE コネクター・クライアントのオンライン・ドキュメンテーションに入っています。

また、以下の追加のサーブレットのサンプルについては、VSE コネクター・クライアント のオンライン・ドキュメンテーションを参照してください。

- *SearchServlet*。これは、VSE ライブラリー・システム (POWER、ICCF、VSE ライブラリー、VSAM) で、特定のファイル、および、特定のテキストが入っているファイルの検索方法を示しているサーブレットです。
- *SdlServlet*。これは、SVA にロードされている VSE フェーズのリストの表示方法を示しているサーブレットです。

サンプル用の VSAM クラスターの作成

次に、サンプルを実行するために必要な 2 つの VSAM クラスターを示します。両方のクラスターとも、VSE/ESA 2.5 以上の VSAM カタログ VSAM.VSESP.USER.CATALOG (VSESPUC) に事前定義されています。

```
FLIGHT.ORDERING.FLIGHTS (FLIGHTS) - KSDS
Offset Length Type      Key    Field Name     Description
-----------------------------------------------------------------
0      4      UNSIGNED  yes    FLIGHT_NUMBER  Flight Number
4      20     STRING    no     START          Start
24     20     STRING    no     DESTINATION    Destination
44     5      STRING    no     DEPARTURE      Departure (hh:mm)
49     5      STRING    no     ARRIVAL        Arrival   (hh:mm)
54     4      UNSIGNED  no     SEATS          Seats
58     4      UNSIGNED  no     RESERVED       Seats reserved
62     4      PACKED    no     PRICE          Price
66     20     STRING    no     AIRLINE        Airline
```

図 *119. FLIGHT.ORDERING.FLIGHTS の VSAM* 構造

```
Offset Length Type      Key    Field Name     Description
-----------------------------------------------------------------
0      20     STRING    no     FIRST_NAME     First Name
20     20     STRING    no     LAST_NAME      Last Name
40     4      UNSIGNED  no     FLIGHT_NUMBER  Flight Number
44     4      UNSIGNED  no     SEATS          Seats
48     1      BINARY    no     NON_SMOKE      Non Smoke 0=no
```

図 *120. FLIGHT.ORDERING.ORDERS の VSAM* 構造

サンプルで使用される HTML 構成

サーブレットは、動的 *Web* ページ を生成するために使用されるので、HTML 構文が、各サーブレットの重要な部分になります。 次に、サーブレットで使用される典型的な 2 つの HTML 構成を示します。

サーブレットが HTML 内にテーブルを作成する方法

次の一般的な方法で、テーブルが HTML 内に定義されます。

```
<table>
<tr><td>"First table cell"</td><td>"Second cell"</td></tr>  (First row)
<tr>  .... </tr>                                           (Second row)
...
</table>
```

サーブレットは、プログラム変数の値をテーブルに書き込むことによって動的テーブル を作成することができます。 動的テーブルは、次のようになります。

```
out.println("<table>");
out.println("<tr><td>" + string1 + </td></tr>");
out.println("<tr><td>" + string2 + </td></tr>");
...
out.println("</table>");
```

ここで、

- *string1* と *string2* は、ストリング変数です。
- *out* は、サーブレットの *doGet()* メソッドまたは *doPost()* メソッド内の *HttpServletResponse* から取得された PrintWriter インスタンスです。

ユーザーの入力を取得するためのフォームの使用

サーブレットは、Web ページを表示した後で、次のサーブレット要求を処理するために、ユーザーの入力を取得しなければなりません。 これは、通常、テキスト・フィールドあるいはプッシュボタンなどのウィンドウ・コントロールを表示する *forms* (フォーム) を使用して行われます。 ユーザーがフォームに関連したアクション (通常はプッシュボタン) を実行すると、このアクションと、テキスト・フィールド、チェック・ボックスおよびその他の入力コントロールからとられた入力パラメーターを使用して、サーブレットが再び呼び出されます。

```
out.println("<form action=¥"/servlet/FlightOrderingServlet¥" method=get>"); 1
out.println("<input type=hidden name=¥"action¥" value=¥"order3¥">");
out.println("<input type=hidden name=¥"flight¥" value=¥"" +
                                                    flightNumber + "¥">");
out.println("<table>");
out.println("<tr><td><b>First Name:</b></td>");
out.println("<td><input size=20 maxlength=20 name=¥"firstname¥"></td>");
out.println("</tr>");

out.println("<tr><td><b>Last Name:</b></td>");
out.println("<td><input size=20 maxlength=20 name=¥"lastname¥"></td>");
out.println("</tr>");
...

out.println("<tr><td><b>Non smoking:</b></td>");
out.println("<td><input type=checkbox name=¥"nonsmoke¥"
                                          value=¥"yes¥">Yes</td>");
out.println("</tr>");
out.println("</table><p>");
out.println("<input type=submit value=¥"Order It!¥">");
out.println("</form>");
```

注: ここで、サーブレット呼び出しを実行するために使用されるストリングは、以下のものによって決まります。

- インストールされている WebSphere Application Server のバージョン。
- ご使用のサーブレット・エンジンに定義した名前。

図 *121.* ユーザーの入力を取得するためにフォームを使用するサーブレットの例

method=get を指定することによって (**1** を参照)、サーブレットの *doGet()* メソッドが呼び出されます。ただし、

- *method=post* を指定して、*doPost()* メソッドを呼び出すこともできます。
- *method=get* と *method=post* の違いは、以下のとおりです。
  - *doGet()* を使用すると、Web ブラウザーのアドレス/ロケーション・フィールドに、生成されたサーブレット呼び出しストリングが表示されます。
  - *doPost()* を使用すると、Web ブラウザーのアドレス/ロケーション・フィールドが抑止されます。
- フォームにパスワード・フィールドがある場合は、*doPost()* を使用することをお勧めします。 *doGet()* を使用すると、パスワードが平文で表示されます。

図 121 に示した HTML コードにより、次に示すウィンドウ・コントロールが表示されます。

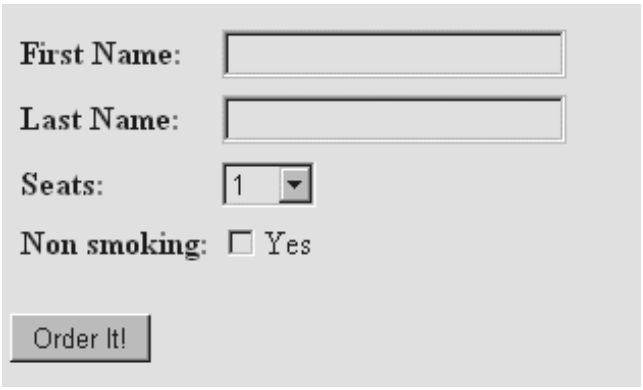


図 122. ウィンドウ・コントロールを表示するためのフォームの使用例

ユーザーがプッシュボタン「**Order It!** (予約)」を押すと、次に示すサーブレット呼び出しストリングが生成されます。

```
http://computername/servlet/FlightOrderingServlet?action=order3&flight=34
&firstname=name1&lastname=name2&seats=1&nonsmoke=no
```

ここで、

- *name1* は、*firstname* フィールドに入力されたストリングです。
- *name2* は、*lastname* フィールドに入力されたストリングです。
- *computername* は、ご使用のネットワーク内のご使用のワークステーションの名前、またはその IP アドレスです。

また、このストリングは、ご使用の Web ブラウザーのアドレス/ロケーション・フィールドに手動で入力することもできます。

サンプル・サーブレット・ステップ 1: フライト・リストの表示

このステップでは、選択可能なフライトのリストが表示され、ユーザーは、このリストから予約を入力できます。 *selectRecords()* メソッドは、FLIGHT.ORDERING.FLIGHTS クラスター内のすべてのレコードを受け取って表示します。

```
public class FlightOrderingServlet extends HttpServlet
{
   // Names of the Clusters and Maps.
   String   vsamCatalog    = "VSESP.USER.CATALOG";
   String   flightsCluster = "FLIGHT.ORDERING.FLIGHTS";
   String   ordersCluster  = "FLIGHT.ORDERING.ORDERS";
   String   flightsMapName = "FLIGHTS_MAP";
   String   ordersMapName  = "ORDERS_MAP";
   ...

   public void doOrderStep1(PrintWriter out,VSESystem system)
   {
      VSEVsamCluster flights = null;
      VSEVsamMap  flightsMap = null;
      FlightsListener fl;

      // create the instances of the flights cluster with its map
      flights = new VSEVsamCluster(system,vsamCatalog, flightsCluster);  1
      flightsMap = flights.getVSEVsamMap(flightsMapName);
      ...

      try
      {
         // use the Listener to build the table
         fl = new FlightsListener(out);  2
         flights.addVSEResourceListener(fl);

         // select all records of the flights cluster (no filter)
         flights.selectRecords(flightsMap);  3
         flights.removeVSEResourceListener(fl);
      }
      catch(Throwable t)
      {
         ...
      }
      ...
   }
   ...
```

図 *123.* フライト・リストを表示するためのサンプル・サーブレット・コード

以下の番号は、図 123 の中の番号の説明です。

1 *VSEVsamCluster* のローカル・インスタンス *flights* が作成されます。 次に、指定した名前が付いた新規マップ・インスタンス *flightsMap* が作成されます。 このマップは、次のステップ 2 で、実際のデータ・フィールドによって充てんされます。 ただし、ステップ 1 では、これらのオブジェクトはローカルのみで、ホストへのアクセスは行われていません。

2 *VSEResourceListener* が作成され、ホストから VSAM レコードを受け取ります。

3 すべてのレコードがホストからリトリーブされます。 *selectRecords()* メソッドが戻ると、すべての VSAM レコードが現行の HTML ページに表示されます。

サンプル・サーブレット・ステップ 2: ホストからのフライト・インスタンスの取得

以下のコードは、内部クラス *FlightsListener* から抜き出したものです。 *listAdded()* メソッドは、受け取られたそれぞれの VSAM レコード・インスタンスごとに呼び出されるコールバック関数です。 コールバック関数について詳しくは 162 ページの

の『VSE Java Beans のコールバック・メカニズムの使用』を参照してください。

```
public void listAdded(VSEResourceEvent event)
{
   String flightNumber, start, destination, departure;
   String arrival, price, airline;

   // The event data has to be a VSEVSamRecord
   if (!(event.getData() instanceof VSEVsamRecord))
     return;

   // Get the record -> it is a record of the flights cluster
   VSEVsamRecord flight = (VSEVsamRecord)(event.getData());   1
   try
   {
      // Get the fields of the record ...
      flightNumber = ((Integer)flight.getKeyField(0)).toString();
      start        = fli ght.getField(1).toString().trim();
      destination  = flight.getField(2).toString().trim();
      departure    = flight.getField(3).toString().trim();
      arrival      = flight.getField(4).toString().trim();
      price        = ((Integer)flight.getField(7)).toString().trim();
      airline      = flight.getField(8).toString().trim();

      // Write out an HTML line in the table
      ...  2

   }
   catch(Throwable t)
   {
         ...
  }
```

図 *124.* ホストからフライト・インスタンスを取得するためのサンプル・サーブレット・コード

以下の番号は、図 124 の中の番号の説明です。

1 *VSEResourceEvent* データから *VSEVsamRecord* インスタンスを取得するには、明示的キャストが必要です。

2 コードの詳細については、VSE コネクター・クライアント のオンライン・ドキュメンテーションを参照してください。

図 124 で説明した Java コードによって、次の Web ページが表示されます。

図 *125.* サンプル・サーブレットによって生成されるフライト予約選択ウィンドウ

図 125 で、最初のサーブレット呼び出しが終了します。 これは、サーブレット・コードが WebSphere Application Server によって、まだ物理/論理中間層プラットフォームのメモリーにロードされていますが、サーブレットが再び呼び出されるまで、これ以上の処理は行われないことを意味します。

注:

1. 次のサーブレット呼び出しは、実際には、新規のプログラム呼び出しになり、したがって、ある呼び出しから次の呼び出しの間にグローバル変数に情報を保管する方法はありません。つまり、すべての入力パラメーターは、サーブレット・パラメーター として受け渡す必要があります。
2. 実際には、サーブレットの存続期間にわたって保管されているプロパティーが 1 つあります。それは、VSE ホスト接続仕様 (*VSEConnectionSpec*) です。 これは、クライアント接続 (*HttpSession*) に保管されており、すべての サーブレット呼び出しで常に同じ値です。詳細については、VSE コネクター・クライアント で提供されているオンライン・ドキュメンテーションを参照してください (30 ページの『オンライン・ドキュメンテーション・オプションの使用』を参照)。

サンプル・サーブレット・ステップ 3: フライトのプロパティーの表示

ここで説明するメソッドでは、予約を入力する際に必要なコントロールと一緒に、フライトのプロパティーが表示されます。 HTML フォームを使用し、以下を実行します。

- ユーザーの入力を取得します。
- 新規サーブレット呼び出しを開始します。

```
public void doOrderStep2(PrintWriter out,VSESystem system,
                         Hashtable parameters)
{
   VSEVsamCluster flights = null;
   VSEVsamMap     flightsMap = null;
   VSEVsamRecord  flight = null;
   int flightNumber, price, seats, reserved;
   String start, destination, departure, arrival, airline;

   // create the instances of the flights cluster and its map
   flights = new VSEVsamCluster(system, vsamCatalog, flightsCluster);
   flightsMap = flights.getVSEVsamMap(flightsMapName);

   // get a instance of a record of the flights cluster
   flight = flights.getVSEVsamRecord(flightsMap);   1

   // get the parameters
   flightNumber = 0;
   try
   {
      flightNumber = Integer.parseInt(getParameterValue(parameters,   2
                                                        "FLIGHT",true));
   }
   catch (Throwable t) {}
   ...

   // get the record and display the properties
   try
   {
      // set the key to identify the locat record with the record
      // on the host
      flight.setKeyField(0,new Integer(flightNumber));   3

      // now get the other fields
      start        = flight.getField(1).toString().trim();
      destination  = flight.getField(2).toString().trim();
      departure    = flight.getField(3).toString().trim();
      arrival      = flight.getField(4).toString().trim();
      seats        = ((Integer)flight.getField(5)).intValue();
      reserved     = ((Integer)flight.getField(6)).intValue();
      price        = ((Integer)flight.getField(7)).intValue();
      airline      = flight.getField(8).toString().trim();

      // display the fields
      ...

      // check if enough seats are available
      ...

      // write out a form where the user can enter its name and select
      // the number of seats to order
      out.println("<form action=¥"/servlet/FlightOrderingServlet¥"   4
                                                          method=get>");
      ...
      out.println("<input type=submit value=¥"Order It!¥">");
      out.println("</form><p>");
   }
   catch(Throwable t)
   {
      ...
   }
   ...
}
```

図 *126.* フライトのプロパティーを表示するためのサンプル・サーブレット・コード

以下の番号は、 264 ページの図 126 の中の番号の説明です。

1 クラス *VSEVsamRecord* のローカル・インスタンスが作成されます。この時点では、割り当てられているプロパティーはありません。 このコードでは、オブジェクトだけが作成されます。 また、クラスのコンストラクターを使用して、同じ結果を得ることができます。

2 関数 *getParameterValue()* を使用して、フォームからフライト番号を取得します。 このメソッドのソース・コードは、VSE コネクター・クライアントのオンライン・ドキュメンテーションにあります。

3 このステートメントによって、FLIGHTS クラスター内の 1 つの特定レコードが識別されます。フライト番号は、ユーザーの入力で提供されます。これで、このフライトのその他のすべてのプロパティーが、このオブジェクトからリトリーブできます。

4 ユーザーが予約を入力するのに必要なウィンドウ・コントロールが表示されます。

264 ページの図 126 で説明したコーディングによって、次の Web ページが表示されます。

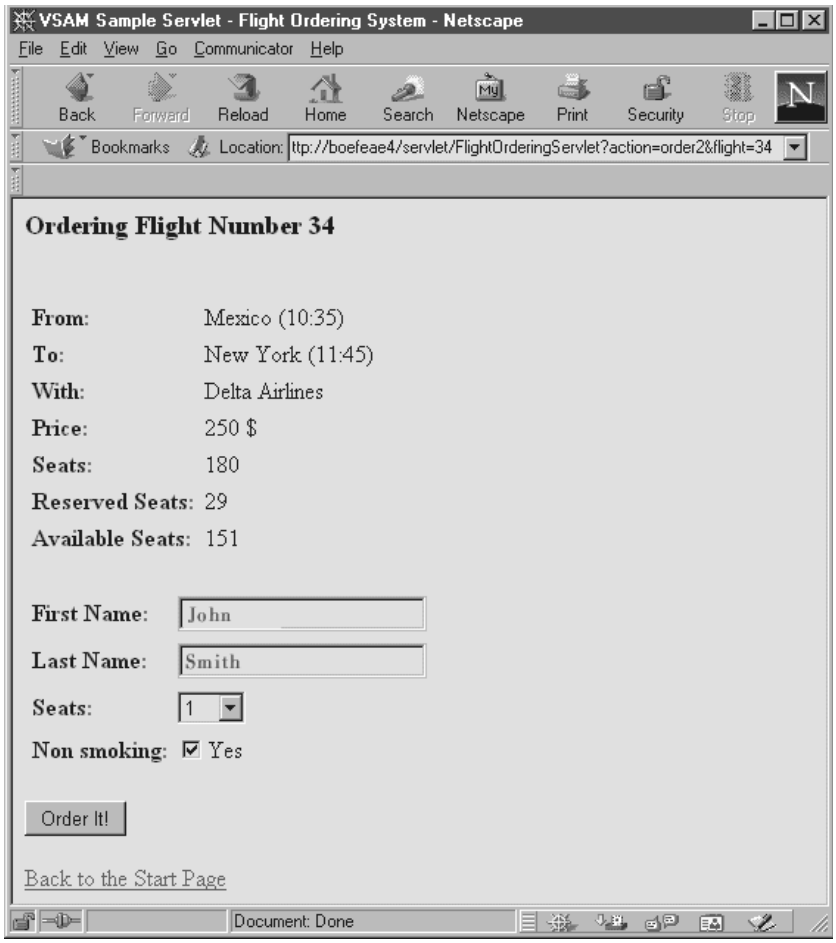


図 *127.* サンプル・サーブレットによって生成されるフライト予約入力ウィンドウ

サンプル・サーブレット・ステップ 4: 予約の入力

このトピックで説明するメソッドを使用して、予約を入力します。ユーザーが「**Order It (**予約**)**」ボタンを押すと、このメソッドが呼び出されて、以下のことが行われます。

1. ユーザーの入力がフォームから読み取られます。
2. フライトのプロパティーが更新されます (このフライトの予約済み席の数が、予約された席の数によって増やされます)。
3. 新規レコードが ORDERS クラスターに追加されます。

```
public void doOrderStep3(PrintWriter out, VSESystem system,
                         Hashtable parameters)
{
   VSEVsamCluster flights = null, orders = null;
   VSEVsamMap flightsMap = null, ordersMap = null;
   VSEVsamRecord flight = null, order = null;
   int flightNumber, seatsToOrder, seats, reserved, price;
   String firstName, lastName;
   boolean nonSmoke, ok;

   // get the parameters from the form
   try
   {
      flightNumber = Integer.parseInt(   1
                         getParameterValue(parameters,"FLIGHT",true));
      seatsToOrder = Integer.parseInt(
                         getParameterValue(parameters,"SEATS",true));
      firstName = getParameterValue(parameters,"FIRSTNAME",false);
      lastName = getParameterValue(parameters,"LASTNAME",false);

      // Check user input ...
      ...

      nonSmoke = false;
      if (getParameterValue(parameters,"NONSMOKE",true) != null)
        nonSmoke = true;
   }
   catch(Throwable t)
   {
      // if not all parameters has been specified -> redisplay Step 2
      doOrderStep2(out,system,parameters);
      return;
   }

   // create instances of the flight cluster and its map
   flights = new VSEVsamCluster(system,vsamCatalog,flightsCluster);   2
   flightsMap = flights.getVSEVsamMap(flightsMapName);

   // get a instance of a record of the flights cluster
   flight = flights.getVSEVsamRecord(flightsMap);

   // create instances of the orders cluster and its map
   orders = new VSEVsamCluster(system,vsamCatalog,ordersCluster);   3
   ordersMap = orders.getVSEVsamMap(ordersMapName);

   // get a instance of a record of the orders cluster
   order = orders.getVSEVsamRecord(ordersMap);

   // Write HTML header
   ...
 try
   {
      // get the flight record
      // set the key to identify the local record with
      // the record on the host
      flight.setKeyField(0,new Integer(flightNumber));   4
      // get some fields of interest
      seats        = ((Integer)flight.getField(5)).intValue();
```

```
      reserved     = ((Integer)flight.getField(6)).intValue();
      price        = ((Integer)flight.getField(7)).intValue();

      // check if enough seats are available
      ...

      // display the order properties
      ...
      // use the OrderCounter to get teh highest record number.
      // This is necessarry because RRDS can only add a record
      // with a non existing record number
      OrderCounter oc = new OrderCounter();  5
      // select all records to find the highest record number
      orders.addVSEResourceListener(oc);
      orders.selectRecords(ordersMap);
      orders.removeVSEResourceListener(oc);

      // now create the new order
      ok = createOrder(out,order,oc.getHighestRecNo()+1,  6
                firstName,lastName,flightNumber,seatsToOrder,nonSmoke);

      out.println("</table><p>");

      // check if creating was ok ...
      if (ok)
      {
         // now update the flight record's fields
         // increase the reserved seats by the number of seats to order
         reserved += seatsToOrder;

         // set the field in the local record
         flight.setField(6,new Integer(reserved));

         // and commit the changes to make them permanent
         flight.commit();  7       }
   }
   catch (Throwable t)
   {
      ...
   }

   // write out some status information and HTML footer
   ...

}
```

以下の番号は、上の例に示されている番号を指しています。

1 サーブレット・パラメーターをフォーム から取得します。

2 FLIGHTS クラスターのローカル・インスタンスと、このクラスターに属すレコードを 1 つ作成します。 この時点までは、これらのオブジェクトに関連付けられているデータはありません。

3 ORDERS クラスターのローカル・インスタンスと、このクラスターに属すレコードを 1 つ作成します。 この時点までは、これらのオブジェクトに関連付けられているデータはありません。

4 FLIGHTS レコードのキー・フィールドをセットします。これは、レコードをこれ以上処理する前に必要です。 この情報は、ホスト上のレコードにアクセスするために、内部的に必要です。

5 RRDS クラスターの場合、新規レコードが、固有の相対レコード番号を使用して追加されます。 したがって、最高位の相対レコード番号を決定してか

ら、新規レコードを追加する必要があります。 すべてのレコードを受け取り、それらをカウントし、さらに、最高位の相対レコード番号を戻すには、*OrderCounter* クラスを使用します。

6 ここで、メソッド *createOrder()* を使用して、新規レコードを ORDERS クラスターに (これに続くコードを使用して) 追加します。

7 *commit()* メソッドを使用して、更新を永久にします。

サンプル・サーブレット・ステップ 5: 新規フライトの作成

ここで説明するメソッドによって、新規レコードが FLIGHTS クラスターに追加されます。 そのレコード (そのキーによって識別される) がすでに存在する場合は、該当するエラー・メッセージが生成されます。

```
public boolean createFlight(PrintWriter out,VSEVsamRecord flight,
                        int flightNumber,String start,String
                        destination,String departure, String arrival,
                        int seats,int reserved,int price,String airline)
throws IOException,ConnectorEx ception
{
   try
   {
       // Set the key of the record
       flight.setKeyField(0,new Integer(flightNumber));  1

       // Set all other fields
       flight.setField(1,makeString(start,20));
       flight.setField(2,makeString(destination,20));
       flight.setField(3,makeString(departure,5));
       flight.setField(4,makeString(arrival,5));
       flight.setField(5,new Integer(seats));
       flight.setField(6,new Integer(reserved));
       flight.setField(7,new Integer(price));
       flight.setField(8,makeString(airline,20));
       // try to add this record
       flight.add();  2

       // Add new row to table
       out.println("<tr><td>" + flightNumber + "</td>");
       out.println("<td>" + start + "</td>");
       out.println("<td>" + destination + "</td>");
       out.println("<td>" + departure + "</td>");
       ...
       out.println("</tr>");
   }
   catch (AlreadyExistentException e)
   {
      // already existing
      return(false);
   }
   return(true);
}
```

図 *128.* 新規フライトを作成するためのサンプル・サーブレット・コード

以下の番号は、図 128 の中の番号の説明です。

1 *KSDS* クラスター の場合、特定のレコードのすべてのキー・フィールドをセットしてから、そのレコードに対するアクションを実行する必要があります。

2 新規フライト・レコードをクラスターに追加します。

サンプル・サーブレット・ステップ 6: 新規予約の作成

ここで説明するメソッドによって、新規レコードが ORDERS クラスターに追加されます。

```
public boolean createOrder(PrintWriter out,VSEVsamRecord order,int recNo,
                        String firstName,String lastName,int flightNumber,
                        int seats,boolean nonSmoke)
throws IOException,ConnectorException
{
   byte[] b = new byte[1];
   try
   {
      // Set the relative record number
      order.setRelRecNo(recNo);   1

      // set all other fields
      order.setField(0,makeString(firstName,20));
      order.setField(1,makeString(lastName,20));
      order.setField(2,new Integer(flightNumber));
      order.setField(3,new Integer(seats));
      if(nonSmoke)
        b[0] = (byte)0x01;
      else
        b[0] = (byte)0x00;
      order.setField(4,b);

      // Add the new record
      order.add();   2

      // Write table row
      out.println("<tr><td>"+firstName+"</td>");
      out.println("<td>"+lastName+ ... +"</td></tr>");
   }
   catch (AlreadyExistentException e)
   {
      // Already existing ...
      return(false);
   }
   return(true);
}
```

図 *129.* 新規予約を作成するためのサンプル・サーブレット・コード

以下の番号は、図 129 の中の番号の説明です。

1 *RRDS* クラスター の場合、相対レコード番号をセットしてから、新規レコードを追加する必要があります。

2 新規予約レコードをクラスターに追加します。

図 129 で説明したコーディングによって、次の Web ページが表示されます。

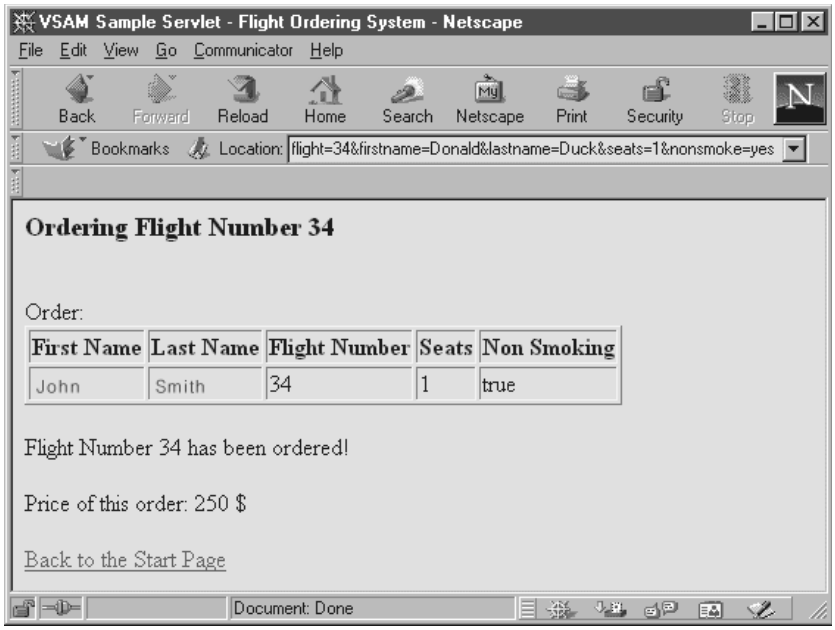


図 *130.* サンプル・サーブレットによって生成されるフライト予約確認ウィンドウ

第 19 章 Java Server Pages によるデータへのアクセス

JSP Requests Java Server Pages (JSP) は、HTML に習熟している開発者が簡単にサーブレット を作成できる方法です。これは、JSP がサーブレットの中にコンパイルされるからです。 また、JSP は、サーブレット、および、動的 HTML コンテンツのその他の生成プログラムが静的 HTML に統合されなければならない Java アプリケーションに非常に役立ちます。 JSP でよく使われる命名規則は、接尾部が **.jsp** で終わるものです。

このトピックに含まれるのは次のとおりです。

- 『3 層環境内での JSP の使用方法』
- 273 ページの『単純な Java Server Pages の例』

3 層環境内での JSP の使用方法

272 ページの図 131 に、z/VSE *3* 層環境 内での JSP の使用方法を示します。以下の各リスト項目の番号は、 272 ページの図 131 に示されているステップを説明しています。

1. クライアントの Web ブラウザーが、物理/論理中間層上で実行されている IBM HTTP Server に、JSP URL に対する要求を送信します。 それぞれの JSP は、IBM HTTP Server の通常の文書階層に保管されています。
2. IBM HTTP Server が JSP をリトリーブし、これを WebSphere Application Server に送信します。
3. ここで WebSphere Application Server によって実行されるアクションは、この JSP が、前にサーブレットの中にコンパイルされたことがあるかどうかによって異なります。
   - JSP が、前に、サーブレットの中にコンパイルされていない 場合 (あるいは、JSP が最後に使用されたときから変更されている場合)、WebSphere Application Server は JSP エンジンを使用してコンパイルを行います。
   - JSP が、前に、サーブレットの中にコンパイルされている (さらに、最後に使用されたときから変更されていない) 場合、結果として得られているサーブレットは、すでにサーブレット・リポジトリーに保管されているはずです。 したがって、JSP のコンパイルは必要ありません。

   次に、WebSphere Application Server が、自身の Java 仮想マシン (JVM) 内でサーブレットを実行します。サーブレットは VSE Java Beans のクラス・ライブラリー (**VSEConnector.jar**) を使用して、VSE コネクター・サーバー への接続を構築します。
4. サーブレットは、必要な HTML Web ページを、クライアントの Web ブラウザーに TCP/IP を使用して送信します。
5. クライアントの Web ブラウザーが Web ページを表示します。 ここで、

Web ブラウザーは、データに対する要求を、WebSphere Application Server の Java 仮想マシンで実行されているサーブレットに送信します。

6 サーブレットは、データに対する要求を、z/VSE ホストで実行されている VSE コネクター・サーバー に (VSE Java Beans を使用して前に構築された接続を使用して) 送信します。

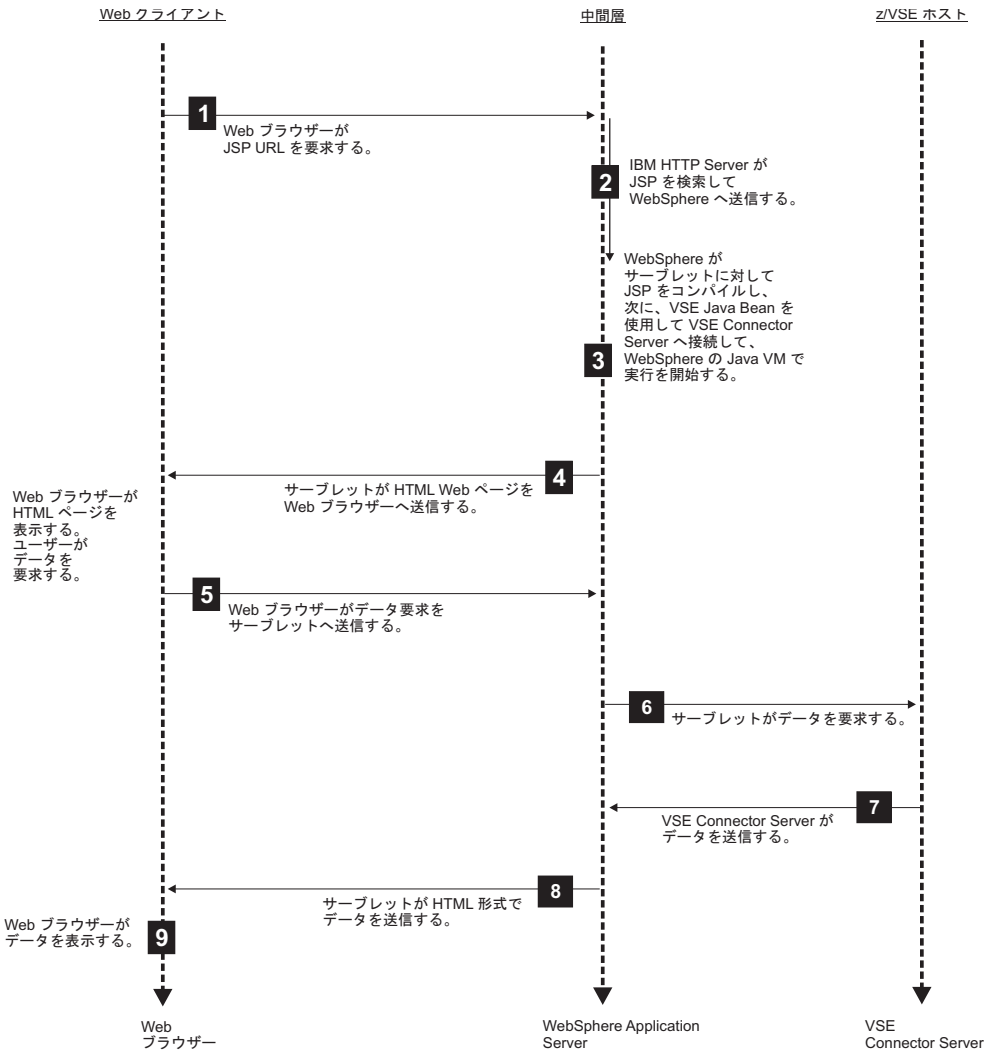


図 *131. z/VSE 3* 層環境内での *JSP* の使用方法

7 VSE コネクター・サーバー はデータを取得し、それをサーブレットに送信します。

注:

1. VSE コネクター・サーバー は、POWER、VSE/ICCF、またはライブラリアンにアクセスするために、あるいは、コンソール・コマンドを送信するために使用することもできます。
2. データにアクセスするために *VSE* コネクター・サーバー を使用する方法の代替メソッドは、z/VSE ホスト上の VSAM ファイル・システムと通信を直接行っている物理/論理中間層上の Db2 ストアード・プロシージャーを使用する方法です。

3. エンド・ユーザーが Db2 データを要求した場合は、データは、物理/論理中間層上の Db2 Connect ルーター、および、z/VSE ホスト上の Db2 Server for VSE を使用して取得できます。また、代替方法として、Db2 データは、物理/論理中間層上の Db2 Connect ルーター、および、z/VSE ホスト上の Db2 ストアード・プロシージャーを使用してリトリーブできます。
4. エンド・ユーザーが DL/I データを要求した場合は、データは、z/VSE ホスト上の Db2 ストアード・プロシージャーと直接通信し、それによって z/VSE ホストの DL/I データベースにアクセスできる、物理/論理中間層上のユーザー作成アプリケーションを使用してリトリーブできます。
5. エンド・ユーザーが CICS データを要求した場合は、データは、物理/論理中間層上の WebSphere MQ Server ルーター、および、z/VSE ホスト上の WebSphere MQ Server for z/VSE を使用して取得できます。

8 サーブレットは、HTML コードから成る動的 Web ページを生成し、この Web ページを、要求されたデータと一緒に、クライアントの Web ブラウザーに送信して戻します。

9 クライアントの Web ブラウザーが、要求データと一緒に Web ページを再表示します。

Web クライアントと物理/論理中間層との間のデータの送受信には HTTP セッションが使用されます。 物理/論理中間層と z/VSE ホストとの間のデータの送受信には接続セッションが使用されます。

サーブレットではなく JSP を使用する利点は、JSP を使用するほうが、ご使用の HTML の機能性が広くなることです。 サーブレットから HTML を作成することはできますが、多くのプログラミング労力を要します。 しかし、JSP を使用すると、変更を行い、その変更が含まれているサーブレットを WebSphere Application Server に再コンパイルさせて実行させるだけで済みます。

また、JSP には、十分な HTML の知識を持っていない作成者が使用できるという利点があります。 JSP は、通常、データベース照会とビジネス・ロジックをカプセル化している Enterprise Java Bean (EJB) にアクセスします。 EJB から戻されたデータは、HTML フォーマットで Web クライアントに送信され、次に、現行の Web ページに動的に組み込むことができます。

単純な Java Server Pages の例

274 ページの図 132 に、現在日付を表示するための JSP の使用方法を示します。特殊なタグ (<% および %>) は、Java コードを囲むために使用されています。JSP エンジンは、JSP をサーブレットの中に動的にコンパイルします。 次に、サーブレットが実行され、Web ページが表示されます。

```
<html>
<head>
<title>My first JSP</title>
</head>
<% import java.util.*; %>
<% Date date = new Date();
   response.println("The current date is " + date);
   ...
%>
</body>
</html>
```

図 *132. Java* サーバー・ページ *(JSP)* の例

第 20 章 EJB を使用したデータの表現

EJB は、IBM の WebSphere Application Server などの Web アプリケーション・サーバー環境で実行される「1 回書き込めばどこでも実行できる」分散 Bean です。

EJB は、以下のいずれかを表すことを目的としています。

- データベース (Entity Bean) 内のデータ。
- リモート・データ・ストア (Session Bean) への接続。

z/VSE 3 層環境内では、EJB を使用して以下のものを表すことができます。

- リレーショナル・データベース (Db2) 内のデータ。
- DL/I または VSAM などの非リレーショナル・データ。 そのためには、すでに、そのような非リレーショナル・データをリレーショナル 構造にマップしてあることが必要です (詳しくは 117 ページの『第 12 章 リレーショナル構造への VSE/VSAM データのマッピング』を参照してください)。

EJB は簡単にインプリメントできます。 さらに、

- EJB は、WebSphere Application Server 環境の最も強力な機能です。
- EJB を使用することにより、アプリケーション・プログラマーは、データへのアクセスをコーディングせずに、Java アプリケーションの開発に集中できます。

このトピックに含まれるのは次のとおりです。

EJB アーキテクチャーの概要

ここでは、EJB アーキテクチャーの概要について説明します。 EJB をインプリメントする方法の実際的な例の説明は 283 ページの『VSAM ベースの EJB をインプリメントする例』にあります。

EJB を適切に管理するために、WebSphere Application Server は、いくつかのサービスを EJB に提供する必要があります。 これらのサービスは、*EJB* コンテナー (277 ページの『EJB コンテナーの使用方法の概要』に説明があります) というエンティティーによって提供されます。

EJB には、*Entity Bean* と *Session Bean* という 2 つのタイプがあります。 次の表に、Entity Bean と Session Bean のプロパティーを示します。

表 7. *Session Bean* と *Entity Bean* のプロパティー

Bean タイプ	説明	状態およびパーシスタンス
Session	単一クライアント・セッションに結合される短命の Bean。	Session Bean の 2 つの主要なフォーム: ステートレス 簡単な操作を実行します。 保持されるデータはありません。 ステートフル クライアント要求間の変数を保持します。
Entity	多くのクライアント・セッションにまたがって存在する長命の Bean。	通常、Session Bean によりラップされます。 EJB の例では、1 つの Entity Bean にアクセスする 2 つの Session Bean が提供されます。 すべての Bean はデータを保留しますが、パーシスタンス (データベースとの同期) を管理する方法には、以下の 2 つの方法があります。 **Bean** による管理 Bean が、行を、データベースとの間で get および put する必要があります。 コンテナーによる管理 Bean が、データを、自動的に get および put します。 EJB の例にある Entity Bean では、*Bean* による管理 のパーシスタンスをインプリメントしています。

Session Bean は、アプリケーションの中のビジネス・ロジックを実行します。 これらの Bean は、例えば、オンライン発注システムのショッピング・「カート」を表したり、購入品の消費税を計算するために使用できます。 Session Bean は、単一クライアント・セッションに結合される通常の Java クラスです。 Session Bean にはその他にいくつかの制限があり、 以下については実行できません。

- 新規スレッドの開始 (これは、EJB が Java 仮想マシン内ではなく、EJB コンテナー内で実行されるためです)。
- データベースの 1 つの行 (VSAM レコードまたは Db2 行) を表すこと以外のすべての機能の実行。
- 読み取り/書き込みの静的変数を入れること。
- **java.io** クラスを使用すること。

Entity Bean には、以下の特性があります。 Entity Bean は、

- JDBC を使用するか、その他の手段によって、データにアクセスしなければなりません (これは、Entity Bean がデータを直接表すからです)。 EJB の例 (283 ページの『VSAM ベースの EJB をインプリメントする例』 ページにあります) では、VSE Java Beans クラス・ライブラリーを使用して、リモート z/VSE ホスト上の VSAM データにアクセスします。
- JDBC インターフェース、または、そのほかのデータ・ソースへのコネクターなどの低レベル・サポートなしに使用することはできません。
- 同時に、複数のクライアントが使用できます。

- Entity Bean が表す基礎となるデータの存続期間と同じ長さの存続期間を持ちます。

EJB コンテナーの使用方法の概要

EJB コンテナーは、EJB とクライアントとの間の仲介役を果たし、また、さらに、複数の EJB インスタンスを管理します。 EJB が作成されたら、WebSphere Application Server (または別のアプリケーション・サーバー) にあるコンテナーに保管する必要があります。

図 133 に、コンテナーを使用して EJB を管理する方法を示します。

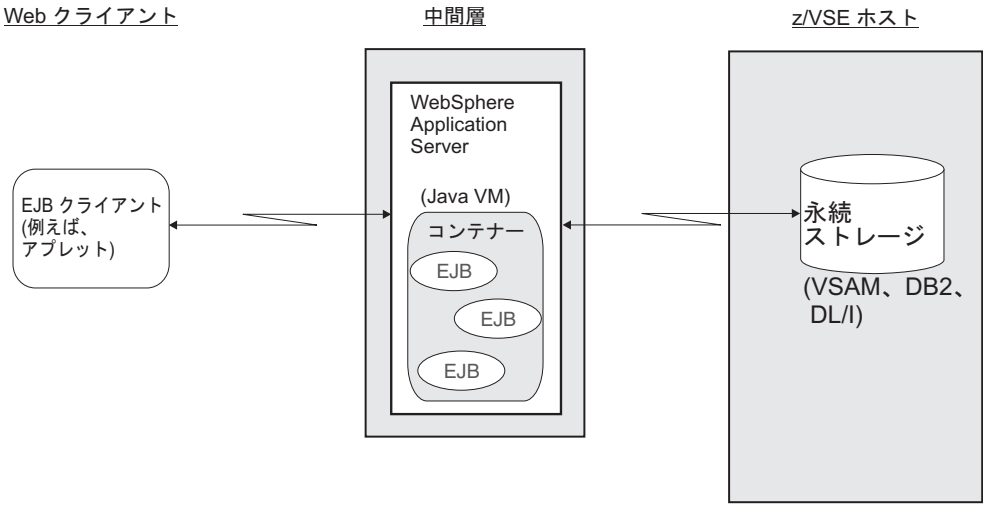


図 *133. EJB* を管理するためのコンテナーの使用方法

EJB コンテナーは、EJB/クライアントの仲介役としての役割を果たすために、いくつかのタスクを実行しなければなりません。 これらのタスクには、以下のものがあります。

- インスタンスの非活動化/活動化。EJB を一時的にスワッピングして、ストレージとの間で出し入れを行う。
- インスタンスのプーリング。複数のクライアントの間で EJB のインスタンスを共用する。
- データベース接続のプーリング。EJB が、既存の接続のプールにあるデータベースへのオープン接続を使用できるようにする。
- 事前にキャッシュに入れられたインスタンス。EJB の最初の作成を迅速に行えるようにするために、EJB の状態情報をキャッシュに入れる。

これにより、コンテナーは、すべてのスレッド化および EJB とのクライアント相互作用を管理し、EJB のインプリメンテーション・プロセスを単純化します。 また、コンテナーは、接続とインスタンス・プーリングを調整して、サーバーとデータベースにかかる負荷を減らします。 EJB が標準化されるにしたがって、アプリケーション・サーバーとは別個の、ベンダーによる EJB コンテナーの提供が増えていくはずです。

EJB を JavaBeans / Java サーブレットと比較する方法

EJB と JavaBeans との違いは、主にその役割にあります。 JavaBeans は、グラフィカル・ユーザー・インターフェース (GUI) で、例えば、ボタンやラベルなどのビジュアル・コンポーネントとして使用されることを目的としています。 他方、EJB は、データベース内のデータを表す、あるいは、このデータに対してアクションを実行するために設計されており、したがって、その使用には制限があります。

EJB およびサーブレットの両方とも、複数層アーキテクチャーに常駐します。 サーブレットは HTTP を使用して要求を受け取り、その要求を処理し、さらに、HTTP を使用して、Web ブラウザーで表示できる HTML コードのフォームで応答を送信します。 したがって、サーブレットは、本質的に、ユーザー・インターフェースの機能とビジネス・ロジックを提供するものです。 ただし、EJB を使用するときは、ユーザー・インターフェースの提供はクライアントが行います。

次に、EJB を使用する際の制限を要約しておきます。

- EJB は、Web サーバー・プラットフォームのローカル・ファイル・システムにアクセスできません。
- EJB は常に単一スレッド・プログラムであるので、コールバック・メカニズムをインプリメントする方法はありません。 これは、とくに実行時間が長いアクション (例えば、オブジェクトのリストをリトリーブするあらゆる要求) では欠点になります。
- アクションがすべて終了するまで、実行中のアクションを停止する、あるいは、結果リストの処理を始める方法はありません。

クライアント・アプリケーションのインプリメント

Enterprise Java Bean を使用すると、クライアント・ユーザー・インターフェースの開発を、ビジネス・ロジックの開発と切り離して行うことができます。 その結果、さまざまなクライアント・プラットフォームおよび複数のユーザー・インターフェースを使用することができます。 EJB は、CORBA コネクティビティーを使用することによって、多くの多様なクライアント・タイプ (その他の EJB、Java アプリケーション、Java アプレット、Java サーブレット、および、非 Java コンポーネントなど) と相互作用することができるようになっています。

Enterprise Java Bean は、すべての Java 互換性および CORBA 互換性のコンポーネントに、簡単なコネクティビティーを提供します。

1. クライアント・プログラムは、JNDI または別の命名システムを使用して EJB ホームを見付け、そのホームを使用してリモート・インターフェースを作成して Bean にアクセスします。
2. これによって EJB コンテナーがトリガーされ、Bean が作成されます。
3. リモート・インターフェースが作成されると、Bean がそのほかのクラスと同じようにクライアントに表示されます。ただし、受け渡されたすべてのオブジェクトは直列化して、ネットワークを使用して送信する必要があります。

EJB は RMI と「スタブ」を使用して、EJB コンテナーとクライアント・コンピューターの間で情報を送信します。 RMI では、ネットワークをまたがって受け渡されるすべてのオブジェクトは直列化可能にする必要があります。 279 ページの図

134 に、単一 EJB メソッド呼び出しで必要なエンティティーを示します。

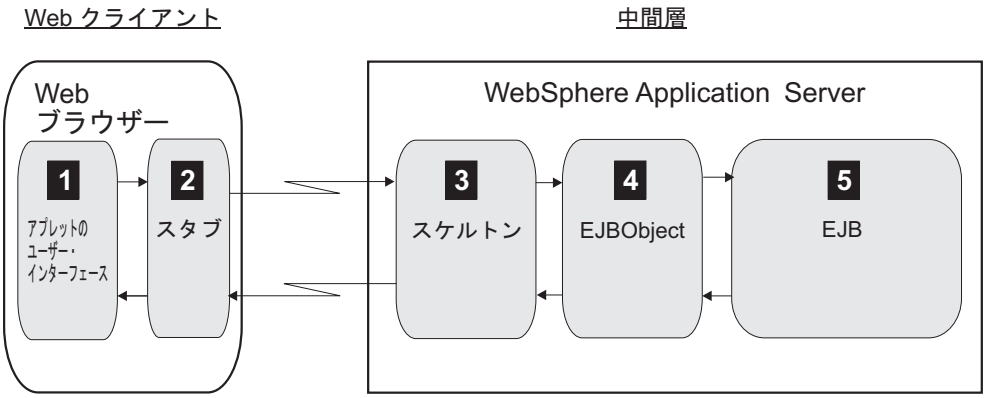


図 *134. EJB* メソッド呼び出しで必要なエンティティーの概要

メソッド呼び出しの際の一般処理は、次のように行われます。

1. ユーザー・インターフェースは、コンテナーによって生成されたスタブを呼び出し、スタブは、メソッド呼び出しの引数を直列化します。
2. 生成されたスタブは、ネットワークを使用してデータを送信します。
3. EJB サーバーに保管されているスケルトンは、引数の直列を解除し、その引数を、コンテナーによって生成された EJBObject (これはリモート・インターフェースの 1 つのインプリメンテーションです) に受け渡します。
4. 次に、EJBObject は、EJB と相互作用します。
5. EJB は、処理を完了すると、結果を EJBObject に戻します。

返信は、上記の逆のパスを使用して、ユーザー・インターフェースに戻されます。したがって、

- サーバー は、EJBObject への呼び出しを管理します。
- コンテナー は、EJBObject を使用して、EJB への呼び出しを管理します。

EJB は同じ呼び出しメソッドを使用して相互に接続し、一緒に通信する EJB から成るシステムを簡単にインプリメントできます。 複数の EJB が 1 つのコンテナーの中にある場合でも、マルチスレッド化の問題を回避するために、それぞれのメソッド呼び出しはコンテナーを使用して行うことが必要です。

CORBA (Common Object Request Broker Architecture) は、非 Java アプリケーションが EJB に接続するメソッドを提供します。 CORBA は RMI とほぼ同じように作動しますが、CORBA から EJB へのマッピングをサポートするアプリケーション・サーバーを必要とし、さらに、さまざまな命名サービスの使用が必要です。

EJB クライアントが EJB にアクセスする方法

280 ページの図 135 に、EJB クライアントが、Enterprise Java Bean (EJB) にアクセスする方法を示します。 以下の説明は、EJB の例、および、一般的な EJB の両方に当てはまります。

クライアントが、EJB と直接通信することはありません。 代わりに、クライアントは、そのホーム・インターフェース および、そのリモート・インターフェース を使用して、EJB と「お話をします」。 ホーム・インターフェースおよびリモート・

インターフェース・コードは、両方とも、EJB のデプロイメント中に WebSphere Application Server によって作成されます。

EJB サーバーと EJB コンテナーは、ご使用の Web アプリケーション・サーバー (例えば WebSphere Application Server) の一部であり、独立ソフトウェア販売会社 (ISV) によって提供される場合もあります。

したがって、以下のコードを EJB 用にインプリメントする必要があります。

- *EJB* クライアント。これは、EJB と通信します。これは、通常はサーブレットまたはアプレットですが、このほかの Java プログラムでもかまいません。
- *EJB* ホーム・インターフェース。 EJB 開発者は、インターフェースを指定するだけで済みます。 そうすることにより、EJB ホーム・インターフェースは、WebSphere Application Server が EJB をデプロイするときに WebSphere Application Server によって作成されます。
- *EJB* リモート・インターフェース。 EJB リモート・インターフェースも、WebSphere Application Server が EJB をデプロイするときに WebSphere Application Server によって作成されます。
- *EJB* クラス。 このクラスは、EJB のビジネス・ロジックをインプリメントします。

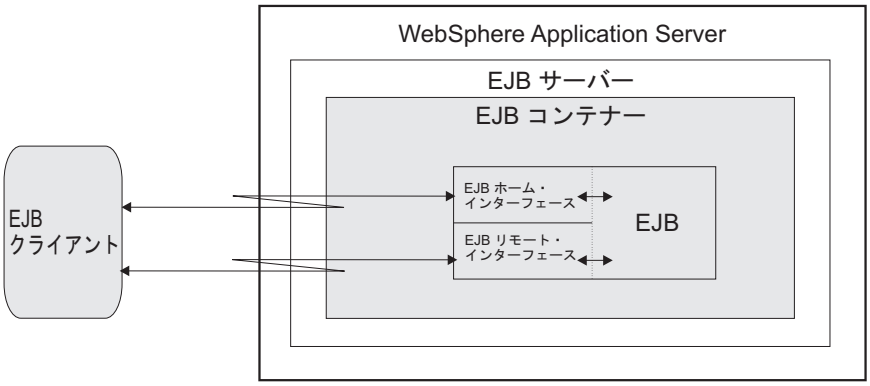


図 *135. EJB* クライアントが *EJB* と通信する方法

図 135 において、*EJB* クライアント は、以下のいずれかになります。

サーブレットまたは **JSP**
: EJB と同じ WebSphere Application Server で実行されます。 サーブレットを EJB クライアントとして使用すると、会社のイントラネットまたはインターネットを使用して、どの Web ブラウザーからでも EJB データにアクセスできるようになります。 デプロイされた **ejb-jar** ファイルは、WebSphere Application Server を実行しているサーバー上にあります。 例えば、オンライン・ショッピングは、多くの場合、静的 Web ページとサーブレットの組み合わせを使用して実行されます。

Java アプリケーション **(Java application)**
: 別のワークステーション上で実行され、TCP/IP 接続を使用して EJB と通信します。 通常、ご使用の開発環境内では、テスト目的には、Java アプリケーションを EJB クライアントとして使用します。 Java アプリケーションは最も単純な EJB クライアントで、さまざまなテスト・ケースをインプ

リメントするために使用できます。デプロイされた **ejb-jar** ファイルは、EJB クライアントと WebSphere Application Server の両方でアクセスできるようにする必要があるため、 EJB クライアントが実行されるワークステーションにこのファイルをコピーするか、WebSphere Application Server と同じマシンで EJB クライアントを実行する必要があります。 また、このファイルは、会社のネットワークを使用して、両方のプラットフォームでアクセスできるようにする必要があります。

アプレット
: 別のワークステーション上の Web ブラウザーで実行され、TCP/IP 接続を使用して EJB と通信します。 HTML で可能なインターフェースより機能が拡張されたユーザー・インターフェースが必要な場合は、アプレットを EJB クライアントとして使用できます。 例えば、オンライン・バンキング・システムでは、アプレットを頻繁に使用します。 そのようなシステムでは、アプレットが必要とするすべてのクラス (デプロイされた **ejb-jar** ファイルを含む) が、アプレットが実行されるときに Web ブラウザー・クライアントにダウンロードされます。これは、特にアプレットがインターネットを介してアクセスされる場合、パフォーマンス上の問題を起こす原因になる可能性があります。

別の **EJB**
: EJB と同じ WebSphere Application Server、または、別の WebSphere Application Server で実行されます。 例えば、VSE コネクター・クライアント の例で提供されている雇用主 Bean と従業員 Bean は、レコード Bean にアクセスするときには EJB クライアントとして行動します。

つまり、EJB クライアントとは、EJB にアクセスする Java プログラムのことです。

EJB を使用した VSAM データへのアクセスの例

282 ページの図 136 に、z/VSE 3 層環境内でアプレットと一緒に EJB を使用して、VSAM データにアクセスする方法を示します。

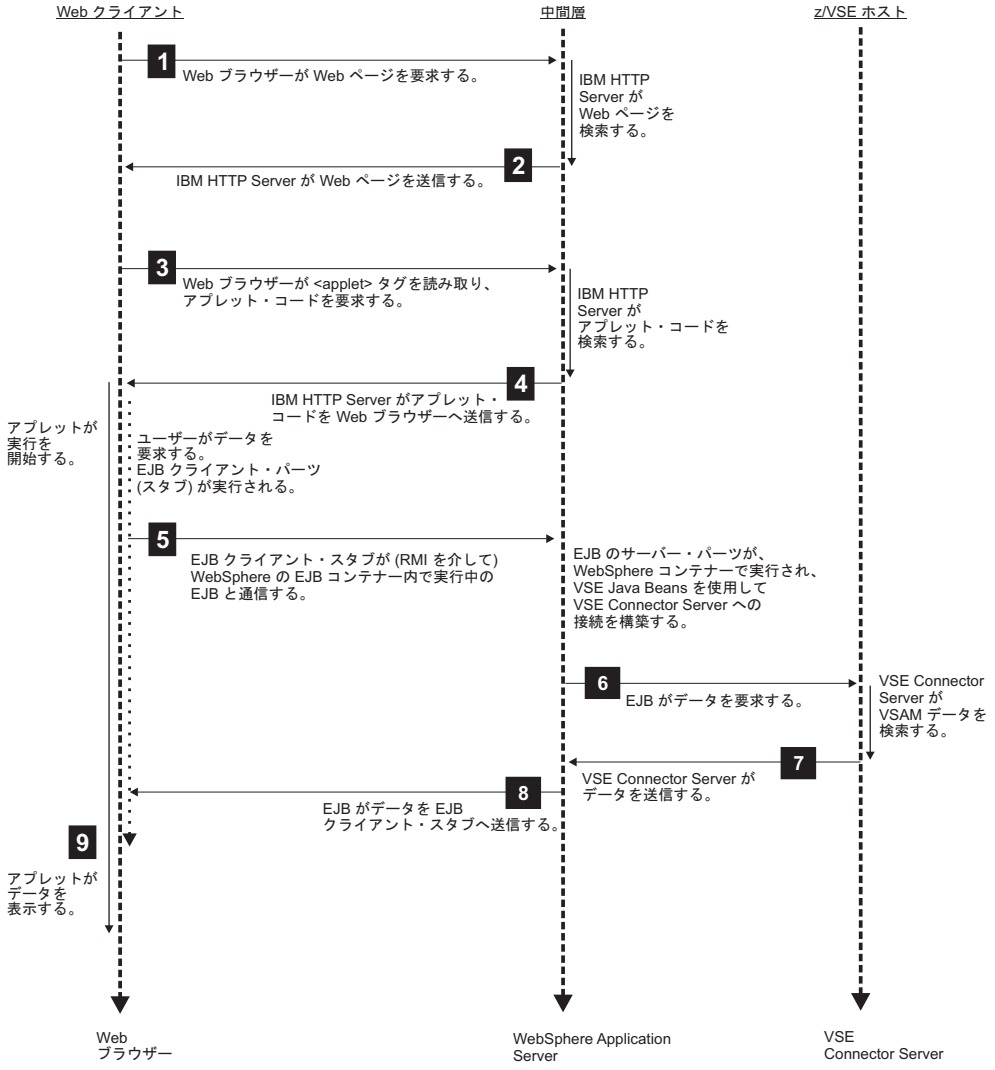


図 *136. 3* 層環境内でアプレットと一緒に *EJB* を使用する方法

Web クライアントと物理/論理中間層との間のデータの送受信には HTTP セッションが使用されます。 物理/論理中間層と z/VSE ホストとの間のデータの送受信にはソケット接続が使用されます。

以下の各リスト項目の番号は、図 136 に示されているステップを説明しています。

1 クライアントの Web ブラウザーが、物理/論理中間層上で実行されている IBM HTTP Server に、HTML ページに対する要求を送信します。 この例では、HTML ファイルにアプレット・タグが入っていて、そのタグによって、アプレットが呼び出されることを想定しています。 アプレットは、EJB クライアントとして作動して EJB データにアクセスします。

2 IBM HTTP Server は Web ページをリトリーブし、それを、クライアントの Web ブラウザーに送信します。

3 クライアントの Web ブラウザーは、<applet> タグを読み取り、アプレット・コードに対する要求を、物理/論理中間層上で実行されている IBM HTTP Server に送信します。 IBM HTTP Server は、*JAR* ファイル (アプ

レット、および、そのアプレットを実行するのに必要な Java ルーチンが入っている) を、クライアントの Web ブラウザーに送信します。

4 IBM HTTP Server はアプレット・コードをリトリーブし、それを、クライアントの Web ブラウザーに送信します。

5 クライアントの Web ブラウザーがアプレットを実行します。 エンド・ユーザーが、物理/論理中間層上に保管されている EJB を使用して、z/VSE ホストに保管されているデータを要求します。 アプレットは EJB スタブ・クラス を使用し、RMI を介して EJB のサーバー部分と通信します。 詳しくは、 277 ページの図 133 および 279 ページの図 134 を参照してください。 EJB のサーバー部分は VSE Java Beans のクラス・ライブラリー (**VSEConnector.jar**) を使用して、VSE コネクター・サーバー への接続を構築します。

6 EJB のサーバー部分は、前に VSE Java Beans を使用して構築された接続を使用して、z/VSE ホスト上で実行されている VSE コネクター・サーバー にあるデータを要求します。

7 VSE コネクター・サーバー は、要求された VSAM データを取得し、これを、EJB のサーバー部分に送信します。

8 EJB のサーバー部分は、クライアントの Web ブラウザーで実行されている EJB スタブにデータを送信します。

9 クライアントの Web ブラウザー内で実行されているアプレットが、要求されたデータと一緒に Web ページを表示します。

VSAM ベースの EJB をインプリメントする例

EJB は、通常、1 つの EJB インスタンスが 1 つのリレーショナル表の行のデータをカプセル化するように、使用されます。 ここで説明する例では、EJB を使用して、リレーショナル構造にマップされた非リレーショナル VSAM データを表す方法について説明します。

リレーショナル構造への非リレーショナル・データのマッピングについては、 117 ページの『第 12 章 リレーショナル構造への VSE/VSAM データのマッピング』 に説明があります。

この例は、*3* 個の EJB から成っています。

- *RecordBean*。これは Entity Bean で、従業員に関するデータが入っている VSAM クラスターに属する、マップされた完全 VSAM レコードを表し、これを入れています。
- *EmployerBean* および *EmployeeBean*。これは Session Bean であり、*RecordBean* によって表されるデータにアクセスするビジネス・ロジックを提供します。 これらの Bean は、このデータに関する別々のビュー、すなわち従業員ビューと雇用主ビューをインプリメントするために使用されます。

この例では、VSAM クラスターをサンプル・データで充てんするために使用するユーティリティーも提供されています。

以下のトピックに、サンプル EJB の実行についての説明があります。

ステップ 1: サンプルの VSAM クラスターの定義

最初のステップでは、この例で使用される VSAM クラスターが作成され、サンプル・データで充てんされます。

注: IESVCLUP ステップでのラベル情報の列配置は重要であるので、このファイルを定義するには、z/VSE 対話式インターフェース・ダイアログを使用することをお勧めします。

以下のジョブを使用して、サンプル・クラスター EJB.VSAM.EXAMPLE を定義します。

```
 * $$ JOB JNM=DEFINE,CLASS=0,DISP=D,NTFY=YES
 // JOB  DEFINE EJB SAMPLE CLUSTER
 // EXEC IDCAMS,SIZE=AUTO
  DEFINE CLUSTER ( -
         NAME (EJB.VSAM.EXAMPLE                          ) -
         CYLINDERS(2        2          ) -
         SHAREOPTIONS (2) -
         RECORDSIZE (80     80    ) -
         VOLUMES (DOSRES ) -
         NOREUSE -
         INDEXED -
         FREESPACE (15 7) -
         KEYS (4   0     ) -
         NOCOMPRESSED -
         TO (99366  )) -
         DATA (NAME (EJB.VSAM.EXAMPLE.@D@                      ) -
         CONTROLINTERVALSIZE (4096  )) -
         INDEX (NAME (EJB.VSAM.EXAMPLE.@I@                    )) -
         CATALOG (VSESP.USER.CATALOG                     )
   IF LASTCC NE 0 THEN CANCEL JOB
 /*
 // OPTION STDLABEL=ADD
 // DLBL EJBSAMP,'EJB.VSAM.EXAMPLE',,VSAM,                          X
                CAT=VSESPUC
 /*
 // EXEC IESVCLUP,SIZE=AUTO
 A EJB.VSAM.EXAMPLE                            EJBSAMP VSESPUC
 /*
 /&
 * $$ EOJ
```

ステップ 2: 従業員用のレコード・レイアウトの作成

以下のレイアウトを使用して、従業員用のレコード・レイアウトを作成します。

```
EMPNUM     Unsigned, offset=0,  length=4   (employee number = key)
PASSWORD   String,   offset=4,  length=10
NAME       String,   offset=14, length=25
```

```
DEPT       Unsigned, offset=39, length=4   ("department number")
HOURLY     Unsigned, offset=43, length=4   ("hourly wage")
PTD        Unsigned, offset=47, length=4   ("paid to day")
TNP        Unsigned, offset=51, length=4   ("time not paid")
```

- 特定の従業員のレコードにアクセスするには、その従業員の番号とパスワードを指定する必要があります。
- 新規データをクラスターに追加するには、ユーティリティー・プログラム *EJBPrepareData.java* を使用します。 このプログラムは、VSE コネクター・クライアント と一緒に提供されています。
- マップとそのデータ・フィールドを作成するには、ユーティリティー・プログラム *EJBPrepareData.java* を使用します。
- サンプル・データを表示するには、Java プログラム *EJBShowData.java* を使用します。これは、VSE コネクター・クライアント と一緒に提供されています。また、*VSE Navigator* アプリケーションを使用してデータを表示することもできます。このアプリケーションは、VSE ホーム・ページからダウンロードできます。VSE ホーム・ページにアクセスする方法の詳細については、 xvii ページの『本書の追加情報の入手先』を参照してください。
- プリコンパイルされたユーティリティーを実行するには、**¥<install-directory>¥samples** ディレクトリーに変更し、次のように入力します。

```
java com.ibm.vse.ejb.vsamexample.EJBPrepareData
java com.ibm.vse.ejb.vsamexample.EJBShowData
```

ステップ 3: EJB のホーム・インターフェースの指定

このステップでは、*RecordBean* 用に、*EJB* ホーム・インターフェース (280 ページの図 135 に説明があります) が指定されます。 そのほかの EJB のホーム・インターフェースを指定する方法の詳細については、 VSE コネクター・クライアント のオンライン・ドキュメンテーション (詳しくは 30 ページの『オンライン・ドキュメンテーション・オプションの使用』を参照) を参照してください。

EJB のホーム・インターフェースは、そのメソッドをインターフェース *EJBHome* から継承します。 以下のメソッドについて、メソッド・シグニチャーを定義する必要があります。

- すべての *create()* メソッド。
- *findByPrimaryKey()* メソッド。
- その他の「finder」メソッド。

EJB がデプロイされると、WebSphere Application Server は、リモート・インターフェースおよびサービス・スタブ・クラスと一緒に、EJB にアクセスするために使用するホーム・インターフェース・コードを作成します。

メソッド・シグニチャーは、次に示すように、ソース・ファイル *RecordHome.java* の中にインプリメントされます。

```
package com.ibm.vse.ejb.vsamexample;
import javax.ejb.*;
import java.rmi.*;

public interface RecordHome extends EJBHome
{
   public Record create(int empnum, String name, String password, int dept,
                        int hourly, int paytodate, int hoursnotpaid)
```

```
    throws java.rmi.RemoteException, javax.ejb.CreateException;

    public Record findByPrimaryKey(RecordPK pk)
    throws RemoteException, FinderException;
}
```

ステップ 4: EJB のリモート・インターフェースの指定

このステップでは、*RecordBean* 用に、*EJB* リモート・インターフェース (280 ページの図 135 に説明があります) が指定されます。 そのほかの EJB のリモート・インターフェースを指定する方法の詳細については、 VSE コネクター・クライアント のオンライン・ドキュメンテーション (詳しくは 30 ページの『オンライン・ドキュメンテーション・オプションの使用』を参照) を参照してください。

EJB がデプロイされると、WebSphere Application Server は、ホーム・インターフェースおよびサービス・スタブ・クラスと一緒に、EJB にアクセスするために使用するリモート・インターフェース・コードを作成します。

リモート・インターフェースは、以下を提供する必要があります。

- インターフェース *EJBObject* からの継承。
- EJB のビジネス・メソッドに関連したメソッド・シグニチャー。
  - これらのメソッドのコードは、Bean をデプロイするプロセスの際に生成されます。
  - メソッド・シグニチャーは、次に示すように、ソース・ファイル *Record.java* の中に提供されます。

```
package com.ibm.vse.ejb.vsamexample;
import javax.ejb.*;
import java.rmi.*;
import java.io.*;
import java.util.*;

public interface Record extends EJBObject
{
   public int getEMPNum() throws RemoteException;
   public String getPassword() throws RemoteException;
   public String getName() throws RemoteException;
   public int getDept() throws RemoteException;
   public int getHourly() throws RemoteException;
   public int getPayToDate() throws RemoteException;
   public int getTimeNotPaid() throws RemoteException;
   public void setDept(int dept) throws RemoteException;
   public void setEMPNum(int empnum) throws RemoteException;
   public void setName(String name) throws RemoteException;
   public void setHourly(int hourly) throws RemoteException;
   public void setPassword(String passwd) throws RemoteException;
   public void setPayToDate(int paytodate) throws RemoteException;
   public void setTimeNotPaid(int timenotpaid) throws RemoteException;
}
```

ステップ 5: RecordPK クラスのインプリメント

RecordPK クラスは 1 つのヘルパー・クラスであり、*java.io.Serializable* をインプリメントするオブジェクトを提供するために Entity Bean が必要とします。 直列化可能なオブジェクトのみが、RMI を介して、引数として、あるいは、戻り値として受け渡されます。

直列化可能なオブジェクトは、例えば、*ejbCreate()* または *ejbFindByPrimaryKey()* によって戻されます。

コードは、ソース・ファイル **RecordPK.java** の中にインプリメントされます。Entity Bean 用の主キー・クラスを、EJB デプロイメント・プロセス (292 ページの『ステップ 8: EJB のデプロイ』に説明があります) 中に指定する必要があります。

```
package com.ibm.vse.ejb.vsamexample;

public class RecordPK implements java.io.Serializable
{
   public int empnum;
}
```

ステップ 6: EJB コードのインプリメント

ここで説明するコードによって、サンプル EJB *RecordBean* がインプリメントされます。 以下のことを行うメソッドが提供されます。

- *RecordBean* のデータへのアクセス。
- リモート z/VSE ホストへの接続。
- データベースにあるデータの取得。
- データベースのアップデート。

コードの一部のみが、このトピックで示されています。完全ソース・コードが必要な場合は、VSE Connector Client と一緒に提供されているファイル **RecordBean.java** を参照してください (詳しくは 30 ページの『オンライン・ドキュメンテーション・オプションの使用』を参照)。

```
...
public class RecordBean implements EntityBean
{
   /* the indexes of the fields in the backend map */
   private int EMPNUM_INDEX = 0;
   private int NAME_INDEX = 2;
   private int PASSWORD_INDEX = 1;
   private int DEPT_INDEX = 3;
   private int HOURLY_INDEX = 4;
   private int PAYTODATE_INDEX = 5;
   private int TIMENOTPAID_INDEX = 6;

   private transient VSESystem system;
   private transient VSEVsamRecord record;

   private transient EntityContext ctx;
   private int empnum = 0;
   ...
```

ステップ 6.1: EntityBean インターフェースのメソッドのインプリメント

ここで説明するメソッドによって、インターフェース *EntityBean* のメソッドがインプリメントされます。 これらのメソッドは、

1. リモート・データベースにアクセスします。
2. 特定の EJB インスタンスを 1 つの VSAM レコードで充てんします。

この例では *ejbFindByprimaryKey()* メソッドを使用します。これは、EJB が、*Bean* によって管理された パーシスタンスを使用してインプリメントされているからです。[3] この結果として、EJB は、オブジェクトをデータベースに送り出し、再びそれを読み取って戻すために必要なすべてのデータベース呼び出しをインプリメントします。

データベースにアクセスするためのロジックは、*ejbLoad()* メソッドおよび *ejbStore()* メソッドにインプリメントされます。 *ejbCreate()* メソッドは、以下に示すパラメーターを使用して新規従業員レコードを作成し、その主キーを戻します。

```
public RecordPK ejbCreate(int empnum, String name, String password,
                          int dept, int hourly, int paytodate,
                          int hoursnotpaid)
throws CreateException
{
   RecordPK rpk= new RecordPK();
   rpk.empnum = empnum;
   try {
      rpk = this.ejbFindByPrimaryKey(rpk);
   }
   catch (Exception e)
   {
      try {
         system = connectVSE();
         record = new VSEVsamRecord(system, "VSESP.USER.CATALOG",
                                    "EJB.VSAM.EXAMPLE", "EJBMAP");
         record.setKeyField(EMPNUM_INDEX, new Integer(empnum));
         record.setField(NAME_INDEX, name);
         record.setField(PASSWORD_INDEX, password);
         record.setField(DEPT_INDEX, new Integer(dept));
         record.setField(HOURLY_INDEX, new Integer(hourly));
         record.setField(PAYTODATE_INDEX, new Integer(paytodate));
         record.setField(TIMENOTPAID_INDEX, new Integer(hoursnotpaid));
         record.add();
         rpk.empnum = empnum;
         this.empnum = empnum;
         return (rpk);
      }
      catch (IOException ioe)
      {
         ...
      }
   }
}
```

EJB が *Bean* によって管理された パーシスタンスをインプリメントするので、以下の 3 つのメソッドはインプリメントしなければなりません。

- ejbLoad()
- ejbStore()
- ejbFindByPrimaryKey()

```
/**
 * fill the EJB with new data from the remote database.
 */
public void ejbLoad() throws RemoteException
```

3. Bean によって管理されたパーシスタンスの反対語はコンテナーによって管理された パーシスタンスです。 ここでは、EJB 開発者は、データベースとの同期化について悩む必要はありません。 代わりに、Entity Bean のデプロイメント記述子によって、EJB コンテナーが管理するフィールドが指定されます。 実行時には、必要に応じて、コンテナーが *ejbLoad()* メソッドおよび *ejbStore()* メソッドを呼び出しますが、EJB 開発者は、これらのメソッドにコードを提供してはなりません。

```
{
   RecordPK pk = (RecordPK) ctx.getPrimaryKey();
   system = connectVSE();
   try {
      record = findRecord(pk.empnum, system);
   }
   catch (FinderException e)
   {
      ...
   }
}

/**
 * make a change permanent in the remote database.
 */
public void ejbStore() throws RemoteException
{
   try {
      record.commit();
   }
   catch (IOException e)
   {
      ...
   }
}

/**
 * looks up the record and returns pk back to the caller.
 * If the record is not found, a FinderException is thrown
 * We have to provide this method, because we implement the EJB
 * using bean-managed persistence.
 */
public RecordPK ejbFindByPrimaryKey(RecordPK pk)
throws FinderException, RemoteException
{
   VSESystem system = connectVSE();
   VSEVsamRecord record = findRecord(pk.empnum, system);
   return (pk);
}
```

ここで説明したメソッドも、インターフェース *EntityBean* に入れられます。 ここに示されていない、このほかのインターフェース・メソッドもあります。完全ソース・コードが必要な場合は、VSE コネクター・クライアント のオンライン・ドキュメンテーション (詳しくは 30 ページの『オンライン・ドキュメンテーション・オプションの使用』を参照) を参照してください。

注: *ejbRemove()* メソッドは、この EJB に関連した VSAM レコードのローカル・インスタンスのみを削除します。 変更内容をリモート・データベースにコミットするには、*ejbStore()* への呼び出しが必要です。

```
public void ejbRemove() throws RemoteException
{
   try {
      record.delete();
   }
   catch (IOException e)
   {
      throw new RemoteException("" + e);
   }
}

public void ejbActivate () throws RemoteException
{
   system = connectVSE();
```

```
      try {
         findRecord(empnum, system);
      }
      catch (FinderException e)
      {
         throw new RemoteException("FinderException");
      }
   }
```

ステップ 6.2: z/VSE ホストへのアクセスおよびデータベースにあるレコードの取得

このトピックでは、以下のことを行うためのメソッドがインプリメントされます。

- z/VSE ホストにアクセスする。
- データベースにあるレコードを取得する。

```
  /**
   * Create connection specification. The connection spec holds
   * information about the physical host connection.
   */
  public VSESystem connectVSE() throws RemoteException
  {
     VSEConnectionSpec spec;
     VSESystem system;
     try
     {
        spec = new VSEConnectionSpec(InetAddress.getByName("9.164.155.95"),
                                     2893, "sysa","mypassw");

        // This is application server dependent
        Properties p = new Properties();
        p.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY,
              "com.ibm.websphere.naming.WsnInitialContextFactory");
        p.put(javax.naming.Context.PROVIDER_URL, "iiop:///");

        Context ctx = new InitialContext(p);
        spec.setJNDIContext(ctx);
        spec.setJNDIName("eis/VSEConnector");

        /* Stay logged on with this user for lifetime of this connection */
        spec.setLogonMode(true);

        /* Create VSE system instance with this connection */
        system = new VSESystem(spec);
     }
     catch (java.net.UnknownHostException e)
     {
        throw new RemoteException("Unknown host");
     }
  }

  /**
   * find and load the record with pk empnum from VSAM.
   */
  protected VSEVsamRecord findRecord(int empnum, VSESystem system)
  throws FinderException
  {
     try {
        VSEVsamRecord rec = new VSEVsamRecord(system, "VSESP.USER.CATALOG",
                                   "EJB.VSAM.EXAMPLE", "EJBMAP");
        rec.setKeyField(EMPNUM_INDEX, new Integer(empnum));
        if (rec.isExistent())
        {
           rec.refresh();
           return rec;
```

```
        }
        else
        {
           throw new FinderException("Record not found.");
        }
     }
     catch (IOException e)
     {
        throw new FinderException("IOException");
     }
  }
```

ステップ 6.3: データ・フィールドにアクセスするための Set メソッドおよび Get メソッドのインプリメント

このトピックでは、データ・フィールド (列) にアクセスするための set メソッドおよび get メソッドがインプリメントされます。これらのメソッドは、EJB のカプセル化されたデータにアクセスするために EJB クライアントが使用するインターフェースです。

アクセスしたいそれぞれの列ごとに、*get* メソッドを提供する必要があります。

更新されるすべての列に対して、*set* メソッドを提供する必要があります。

注:

1. これらのメソッドがリモート・データベースにアクセスすることはありません。これらのメソッドは、EJB の内部データを戻すか変更するだけです。
2. 変更を永続的に 更新する必要がある場合は、インターフェース・メソッド *ejbStore()* を使用する必要があります。

```
  /**
   * returns the employee number for the record.
   */
  public int getEMPNum() throws RemoteException
  {
     try {
        return ((Integer)record.getField(EMPNUM_INDEX)).intValue();
     }
     catch (IOException e)
     {
        throw new RemoteException("IOException");
     }
  }


  /**
   * set the employee number for the current record
   */
  public void setEMPNum(int empnum) throws RemoteException
  {
     try
     {
        record.setField(EMPNUM_INDEX, new Integer(empnum));
     }
     catch (IOException e)
     {
        throw new RemoteException("IOException");
     }
     return;
  }
  ...
}
```

ステップ 7: Java ソース・ファイルのコンパイル

EJB サンプルをセットアップするには、このトピックで説明する Java ソース・コードをコンパイルする必要があります。 ディレクトリー **¥<install-directory>¥samples** からコンパイル・ジョブを実行します。

注: EJB に関連するクラスが入っている特定の JAR ファイルを、ご使用のローカル・クラスパスに組み込む必要があります。 詳しくは、以下を参照してください。

- VSE コネクター・クライアント のオンライン・ドキュメンテーション (詳しくは 30 ページの『オンライン・ドキュメンテーション・オプションの使用』を参照してください)。
- 25 ページの『WebSphere Application Serverのインストール』にある資料のリファレンス。

VSE コネクター・クライアント には、サンプル Java ソース・ファイルのすべてについて、すでにコンパイルされているクラス・ファイルが用意されています。 ただし、以下のソース・ファイルは、EJB の例に属します。

1. サンプル・データを作成し表示するためのユーティリティーをコンパイルするジョブ。

```
javac com¥ibm¥vse¥samples¥EJBPrepareData.java
javac com¥ibm¥vse¥samples¥EJBShowData.java
```

2. EJB ソース・ファイルをコンパイルするジョブ。

```
javac com¥ibm¥vse¥samples¥Employer.java
javac com¥ibm¥vse¥samples¥EmployerHome.java
javac com¥ibm¥vse¥samples¥EmployerBean.java
javac com¥ibm¥vse¥samples¥Employee.java
javac com¥ibm¥vse¥samples¥EmployeeHome.java
javac com¥ibm¥vse¥samples¥EmployeeBean.java
javac com¥ibm¥vse¥samples¥Record.java
javac com¥ibm¥vse¥samples¥RecordHome.java
javac com¥ibm¥vse¥samples¥RecordBean.java
```

3. EJB クライアントをコンパイルするジョブ。

```
javac com¥ibm¥vse¥samples¥EJBApplet.java
javac com¥ibm¥vse¥samples¥EJBClient.java
```

ステップ 8: EJB のデプロイ

EJB をデプロイするために使用するメソッドは、使用している WebSphere Application Server のバージョンによって異なります。 VSE コネクター・クライアント のオンライン・ドキュメンテーションには、WebSphere Application Server のさまざまなバージョンに対する EJB のデプロイ方法の詳細な説明があります (30 ページの『オンライン・ドキュメンテーション・オプションの使用』を参照)。

EJB のデプロイ方法の詳細情報を得るには、以下のようにします。

1. 30 ページの『オンライン・ドキュメンテーション・オプションの使用』に記載されているオンライン・ドキュメンテーションのメイン・ウィンドウを表示します。
2. 「**Programming Concepts (プログラミング概念)**」を選択します。
3. 「**EJBs**」を選択します。
4. 「**An example to represent VSAM records (VSAM レコード表現例)**」を選択します。

5. 以下のいずれかを選択します。
   - **WebSphere 3.x での EJB のデプロイ (Deploy the EJBs on WebSphere 3.x)**
   - **WebSphere 4.x での EJB のデプロイ (Deploy the EJBs on WebSphere 4.x)**

ステップ 9: EJB クライアントからの EJB へのアクセス

EJB クライアントから EJB にアクセスするための前提条件

作成した EJB に EJB クライアントからアクセスするには、以下の条件が満たされていなければなりません。

1. 284 ページの『ステップ 1: サンプルの VSAM クラスターの定義』に説明されているように、サンプルを実行するのに必要な VSAM クラスターが正常に作成されている。
2. 284 ページの『ステップ 2: 従業員用のレコード・レイアウトの作成』に説明されているように、例えば、ユーティリティー *EJBPrepareData* を使用して、上記のクラスターがサンプル・データで充てんされている。
3. (23 ページの『TCP/IP の構成およびアクティブ化』で説明されているように) TCP/IP for z/VSE が z/VSE ホストで実行されている。
4. (39 ページの『VSE コネクター・サーバー の始動』で説明されているように) VSE コネクター・サーバー が z/VSE ホストで実行されている。
5. (30 ページの『オンライン・ドキュメンテーション・オプションの使用』で説明されているように) 正しい IP アドレス、VSE ユーザー ID、およびパスワードが **RecordBean.java** に指定されている。
6. (23 ページの『VSE HTTP Server の構成およびアクティブ化』で説明されているように) IBM HTTP Server が物理/論理中間層で実行されている。
7. WebSphere Application Server が物理/論理中間層で実行されている (25 ページの『WebSphere Application Serverのインストール』にある資料のリファレンスを参照)。
8. (292 ページの『ステップ 8: EJB のデプロイ』で説明されているように) サンプルの EJB が実行されている。

EJB クライアントから EJB にアクセスする方法

EJB クライアントは、以下の一般的な方法を使用して、EJB の中のビジネス・ロジックにアクセスします。

1. EJB は命名サービスを使用して、EJB のホーム・インターフェースを見付けます。
2. 命名サービス (通常は JNDI (Java Naming and Directory Interface)) は、EJB のホーム・インターフェース (279 ページの『EJB クライアントが EJB にアクセスする方法』に説明があります) をインプリメントするオブジェクトにリファレンスを戻します。
3. EJB クライアントは EJB のホーム・インターフェースに呼び出しを行い、EJB のリモート・インターフェース (279 ページの『EJB クライアントが EJB にアクセスする方法』に説明があります) にアクセスします。

4. EJB クライアントは、リモート・インターフェースに照らして、EJB のビジネス・メソッドを呼び出します。

ただし、EJB クライアントから EJB にアクセスするために使用する実際のコードは、使用する Web Application Server のタイプに応じて異なります。

図 137 に、*EJBClient* のサンプル・アプリケーションが、ホーム・インターフェース (**H** として示す) およびリモート・インターフェース (**R** として示す) を使用して、EJB にアクセスする方法を示します。 EJB クライアントを使用して EJB にアクセスするための完全コードを参照したい場合は、VSE コネクター・クライアントのオンライン・ドキュメンテーションを参照してください。

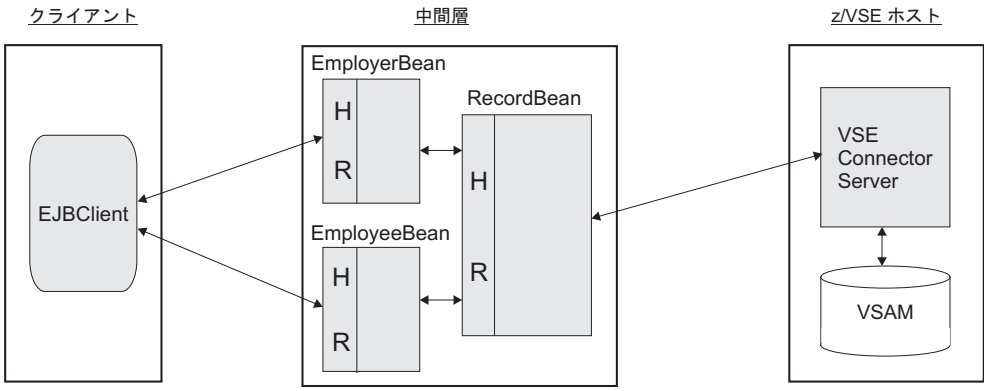


図 *137.* 提供されている例で *EJB* クライアントが *EJB* にアクセスする方法

2 つの Session Bean は、実際には、それ自身が EJB クライアントです。 これらの Bean は、*RecordBean* にアクセスするとき、EJB クライアントが Session Bean にアクセスするときに行うのと同じように、ホーム・インターフェースとリモート・インターフェースを検索するというプロセスを実行します。

Session Bean は、特定の列をビューに組み込むか組み込まないことによって、同じデータに対して 2 つの異なるビューをインプリメントします。 さらに、Session Bean は、特定の列に対するアクセス権限を指定します。 例えば、*EmployeeBean* は、特定の列を読み取ることしかできませんが、*EmployerBean* には、この列を更新できるようにします。

EJB クライアントから EJB にアクセスするための EJB クライアント・ソース・コードのサンプル

このトピックでは、VSE Connector Client の一部であるサンプルの EJBClient アプリケーションのソース・コードについて説明します。このコードは、Web アプリケーション・サーバーのタイプとしての *WebSphere Application Server* の使用を基にしています。

```
public class EJBClient
{
   public static void main(String[] argv)
   {
      try
      {
         EmployerHome employerh;
         Properties p = new Properties();  1
         p.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY,
              "com.ibm.websphere.naming.WsnInitialContextFactory");
         p.put(javax.naming.Context.PROVIDER_URL, "iiop:///");
         InitialContext ic = new InitialContext(p);  2
         Object obj = (Object) ic.lookup("EmployerBean");  3
         if (obj instanceof org.omg.CORBA.Object)  4
         {
            employerh =
                (EmployerHome)(javax.rmi.PortableRemoteObject.narrow(
                         (org.omg.CORBA.Object)obj, EmployerHome.class));
         }
         else
         {
            System.out.println(
                   "Did not get an org.omg.CORBA.Object from lookup().");
            return;
         }
         Employer employer = employerh.create(1003, "newpass3");  5
         System.out.println("employer.calcPayToDate() = " +  6
                                              employer.calcPayToDate());
         Vector v = employer.getEMPInfo();
         System.out.println("employer.getEMPInfo() :");
         for (int i=0;i<v.size();i++)
            System.out.println("  " + v.elementAt(i).toString());
      }
      catch (Exception e)
      {
         ...
      }
   }
}
```

図 *138. EJB* クライアント・コードの例

以下の番号は、図 138 の中の番号の説明です。

1 プロパティーを定義します。 PROVIDER_URL として **iiop:///** を指定することにより、EJB クライアントが、WebSphere Application Server のデフォルトであるポート番号 900 で listen するローカル・ホスト上にあるネーム・サーバーを検索します。 実際の環境では、次のように、完全な URL を指定します。

```
iiop://bankserver.mybank.com:9019
```

2 初期コンテキスト・オブジェクトがインスタンス化されます。

3 EmployerBean のホーム・インターフェースが検索されます。 EJB を識別するためにここで使用されているストリングは、EJB のデプロイメント記述子の中に保管されています。また、*WebSphere* 管理コンソール を使用することにより、EJB のデプロイメント・プロパティーを表示することもできます。

4 戻されたオブジェクトを、EJB のホーム・インターフェース・クラスにキャストします。 検索メソッドによってオブジェクトが戻されたら、静的メソ

ッド *PortableRemoteObject.narrow()* を使用して、指定した EJB の EJB ホーム・オブジェクトを取得する必要があります。

5 リモート・インターフェース・オブジェクトを作成して、EJB のビジネス・ロジック・メソッドにアクセスします。 リモート・インターフェース *create()* メソッドを呼び出すことによって、EJB の *ejbCreate()* メソッドが呼び出されます。

6 EJB のビジネス・ロジック・メソッドにアクセスします。 例えば、1003 という番号の雇用主のいくつかのプロパティーを入手します。 ただし、EJB は常にリモート・オブジェクト (この例では「Employer.class」) を使用してアクセスされることに注意してください。 EJB クライアントは、EJB (この例では **EmployerBean.class**) にあるメソッドを直接呼び出すことはできません。 上記の例の雇用主番号とパスワードは、VSE コネクター・クライアント で提供されている EJB サンプルからとられています (VSE コネクター・クライアント は、サンプルの VSAM クラスターをサンプル・データで充てんするユーティリティーも提供しています)。

第 21 章 Java ベース・コネクターの拡張

このトピックでは、ユーザー独自の「プラグイン」を作成して、2 層環境および 3 層環境内で Java ベース・コネクターを拡張する方法について説明します。この拡張では、z/VSE ホストの現行日時を取得して表示する「日時プラグイン」という例を頻繁に使用します。

ユーザー独自のプラグインを作成することにより、以下のものに追加アクセスを行えるようになります。

- リソース (CICS で現在オープンしている VSAM ファイルにアクセスするなど)。
- アプリケーション (CICS トランザクションを開始する、または、ベンダー・アプリケーションを開始するなど)。

ユーザーは、以下のものから成る独自のプラグインを作成できます。

- クライアント・プラグイン のみ。 これは、VSE Java Beans のクラス・ライブラリーを拡張することによって行います。 この場合、ユーザーのプラグインは JavaBean になり、これ自身で (既存の) JavaBeans を使用して、データにアクセスするか、サービスを提供します。 注**:** この JavaBean は、*VSEPlugin* をインプリメントしません (詳しくは以下を参照)。
- クライアント・プラグイン およびサーバー・プラグイン。 この場合、ユーザーのサーバー・プラグインは、ユーザーのクライアント・プラグインで必要な追加機能を提供することによって VSE コネクター・サーバー の機能を拡張します。

ユーザーのサーバー・プラグインは、VSE コネクター・サーバー に対するプラグインである VSE PHASE から成ります。この PHASE (これは、ユーザーが LE/VSE-C を使用して作成する) は、VSE コネクター・サーバー のスタートアップ時にロードされます。また、PHASE は、よく定義されたインターフェースを提供し、インプリメントされた関数を VSE コネクター・サーバー が呼び出せるようにしなければなりません。

ユーザーのクライアント・プラグインは、ユーザー自身が作成した 1 つ以上の JavaBeans から成ります。これらの JavaBeans は Java クラス *VSEPlugin* をインプリメントします (これは、パッケージ **com.ibm.vse.connector** に入れられます)。ユーザー独自の JavaBeans の作成方法の詳細については、オンライン・ドキュメンテーションを参照してください。

このトピックに含まれるのは次のとおりです。

サーバー・プラグインのインプリメント

サーバー・プラグインは、VSE コネクター・サーバー によって特定のシーケンスで呼び出される一連のコールバック関数 から成っています。 それぞれのサーバー・プラグインは、ユーザーが LE/VSE-C を使用して作成するフェーズから成り、このフェーズは VSE コネクター・サーバー のスタートアップ時にロードされます。ユーザーのプラグインは、よく定義されたインターフェースを提供し、ユーザーのサーバー・プラグインでインプリメントされている関数を VSE コネクター・サーバー が呼び出せるようにしなければなりません。 これらのコールバック関数 (PluginMainEntryPoint、SetupPlugin など) は、このトピックの後のほうで説明します。

VSE コネクター・サーバー によって関数が呼び出される方法の概要が図 139 に示されています。

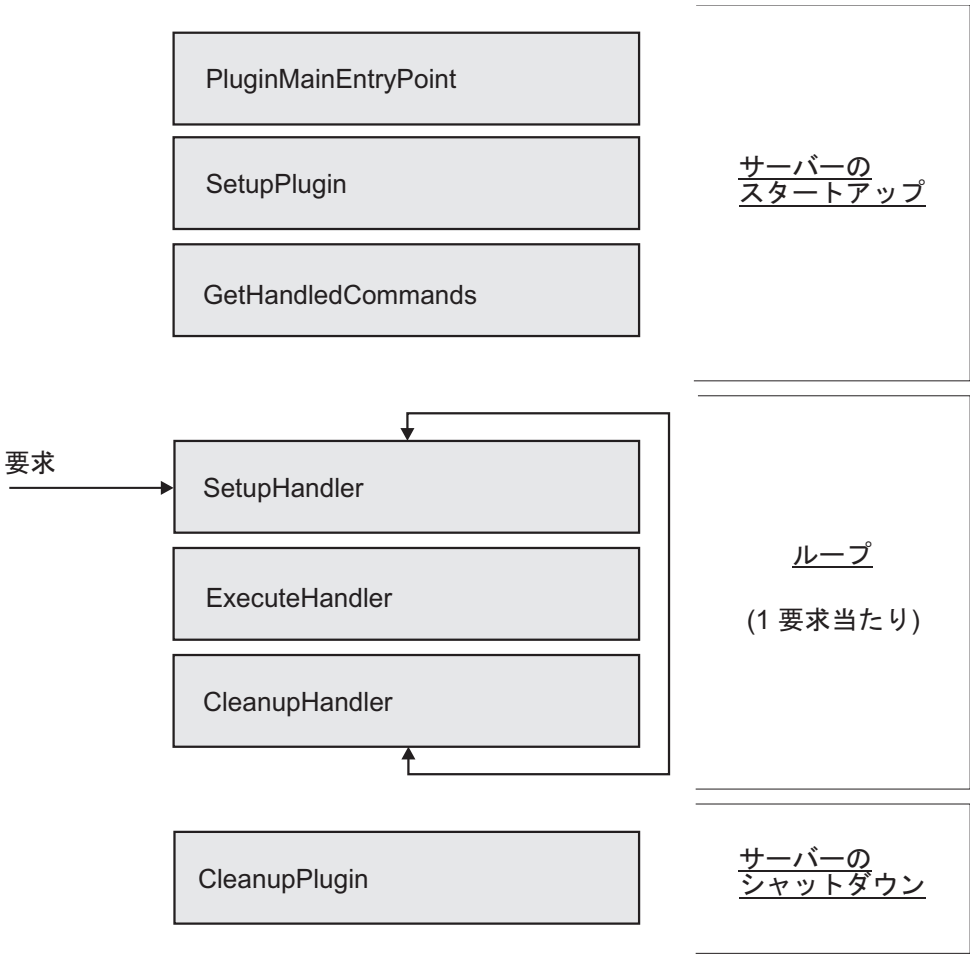


図 *139.* サーバー・プラグインの関数が呼び出される方法の概要

VSE コネクター・サーバー によって関数が呼び出される方法の典型的なシーケンス を、 299 ページの図 140、 300 ページの図 141、および 301 ページの図 142 に示します。

- 299 ページの図 140 には、VSE コネクター・サーバー のスタートアップ時に呼び出されるサーバー・プラグインの関数の説明があります。

- 300 ページの図 141 および 301 ページの図 142 に、VSE コネクター・サーバー のシャットダウン中に、それぞれ、VSE コネクター・サーバー がクライアントから要求を受け取ったとき、および、その要求の処理が終了したときに呼び出されるサーバー・プラグインの関数の説明があります。

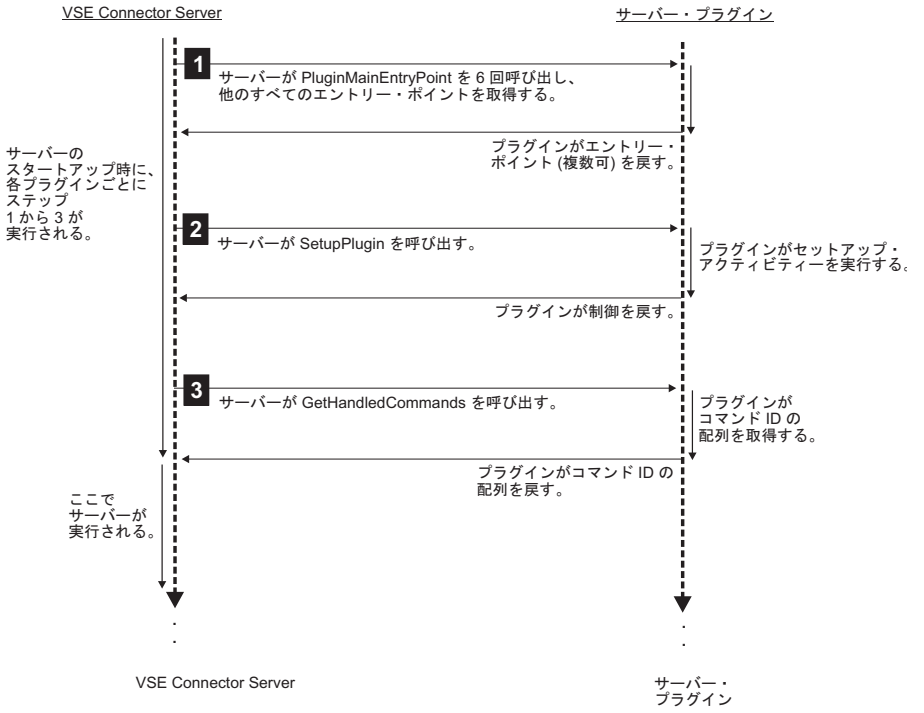


図 *140.* スタートアップ中にプラグインの関数が呼び出される方法

1 サーバー・プラグイン・フェーズ (CDLOAD) をロードすることにより、VSE コネクター・サーバー は、サーバー・プラグインのエントリー・ポイント (*PluginMainEntryPoint*) を取得します。 次に、その他のサーバー・プラグイン関数のエントリー・ポイントを取得するために、このエントリー・ポイントが 6 回呼び出されます。 それぞれの呼び出しごとに、パラメーター *iFuncID* によって指定された関数のエントリー・ポイントが戻されます。 このステップが完了すると、サーバーには、プラグイン・フェーズのエントリー・ポイントのリストができます。

2 *SetupPlugin* 関数は、サーバー・プラグインがロードされた後で VSE コネクター・サーバー が呼び出す最初の関数です。 この関数は、要求に依存しないすべてのリソース (ストレージの割り振り、データベース接続のオープンなど) を初期化するために使用されます。

3 VSE コネクター・サーバー は、*SetupPlugin* 関数を呼び出した直後に *GetHandledCommands* 関数を呼び出します。 *GetHandledCommands* は、サーバー・プラグインが受け入れた ID のリストを戻します。 VSE コネクター・サーバー は、特定のサーバー・プラグインに要求をダイレクトするために、あとでこの情報を使用します。 したがって、コマンド ID は、すべてのサーバー・プラグインで固有でなければなりません。 VSE コネクタ

ー・サーバー は、定義済みの範囲のコマンド ID を受け入れます。 詳細については、ヘッダー・ファイル IESPLGIN.H ファイルを参照してください。

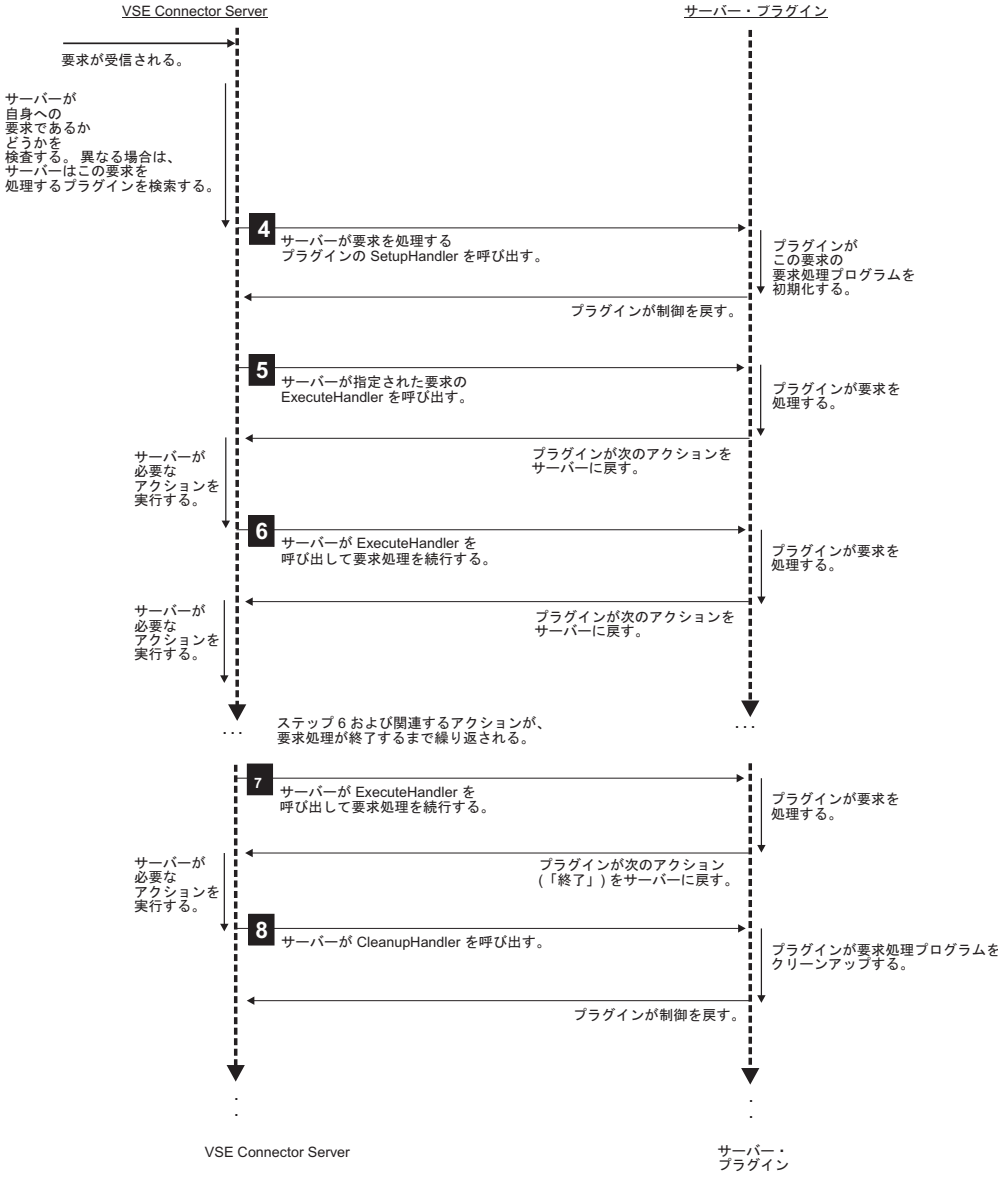


図 *141.* 要求を受け取ったときにプラグインの関数が呼び出される方法

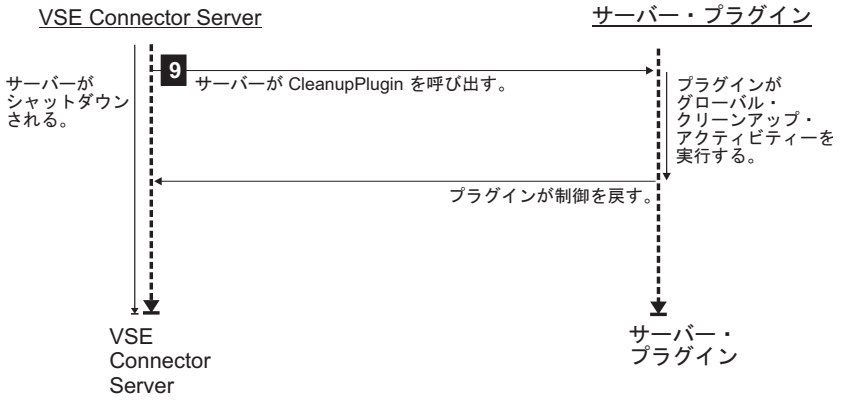


図 *142.* サーバーのシャットダウン中にプラグインの関数が呼び出される方法の概要

4 クライアントからの要求を受け取ると、VSE コネクター・サーバー は、まず、その要求をどのサーバー・プラグインが処理すべきかを判別し、選択されたサーバー・プラグインの *SetupHandler* 関数を呼び出します。 サーバー・プラグインは、要求を処理するために使用する制御ブロックから成る要求処理プログラム を、この要求のために割り振ります。制御ブロックは *SetupHandler* 関数によって割り振られ、1 つの要求に対して特定の制御ブロックになります。

5 特定の要求を処理するために、VSE コネクター・サーバー は、サーバー・プラグインの *ExecuteHandler* 関数を呼び出します。 サーバー・プラグインは、通常、このコードを小さなチャンクに分割し、それぞれのチャンクが、*ExecuteHandler* 関数への次の呼び出しの際に VSE コネクター・サーバーによって呼び出されます (以下のステップ 6 および 7 を参照)。 *SetupHandler* 関数によって割り振られた制御ブロックは、それぞれの呼び出しのたびに、*ExecuteHandler* 関数に受け渡されます。 この制御ブロックは、ローカル変数を保管するためにも使用できます。 次に、VSE コネクター・サーバー が、*ExecuteHandler* 関数によって戻されたアクションを処理しますが、その間、この関数は、データの送受信を行っていても、あるいは、タイマーまたは ECB を待っていてもかまいません。

6 VSE コネクター・サーバー は、再び、サーバー・プラグインの *ExecuteHandler* 関数を呼び出し、要求の処理を続けます。 *SetupHandler* 関数によって割り振られた制御ブロックは、再び、*ExecuteHandler* 関数に受け渡されます。 次に、VSE コネクター・サーバー が、*ExecuteHandler* 関数によって戻されたアクションを処理します。

7 VSE コネクター・サーバー が、サーバー・プラグインの *ExecuteHandler* 関数を呼び出し、要求の処理を続けます。 次に、VSE コネクター・サーバー が、*ExecuteHandler* 関数によって戻されたアクションを処理しますが、この場合、ExecuteHandler 関数は、サーバー・プラグインがすべての処理を終了した (すべてのアクションが完了した) ことを示します。

8 VSE コネクター・サーバー は、*CleanupHandler* 関数を呼び出し、この要求に割り振られていたすべてのリソースをサーバー・プラグインがクリーンアップできるようにします。 要求の処理中に予期しないエラー (例えばネッ

トワーク接続が切断される) が起きた場合、*CleanupHandler* 関数は、この時点までに使用されていたすべてのリソースをクリーンアップする必要があります。

9 VSE コネクター・サーバー がシャットダウンされ、ロードされていたすべてのサーバー・プラグインの *CleanupPlugin* 関数を呼び出します。

PluginMainEntryPoint 関数のインプリメント

VSE コネクター・サーバー は、この関数を呼び出して、プラグインのエントリー・ポイントを確立します。 *PluginMainEntryPoint* 関数は、このプラグインの関数の、このほかのすべてのエントリー・ポイントを取り出します。 VSE コネクター・サーバー は、 *PluginMainEntryPoint* を数回呼び出します。 それぞれの呼び出しごとに、*iFuncID* パラメーターを使用して指定されたプラグイン関数のエントリー・ポイントが戻されます。

注:

1. 関数 *PluginMainEntryPoint* は、名前変更しないでください。 これは、C ヘッダー・ファイル IESPLGIN.H に、この関数を**取り出し可能**として定義している `#pragma linkage` ステートメントが入っているからです。
2. この関数を名前変更することにした場合は、次のように、ユーザー自身の `#pragama linkage` ステートメントを指定する必要があります。

   ```
   #pragma linkage(MyEntryPointFunction,fetchable)
   ```

   このステートメントにより、*MyEntryPointFunction* が、ユーザーの PHASE のエントリー・ポイントとして使用されることが強制されます。

次に、*PluginMainEntryPoint* 関数のコーディングの例を示します。

```
/*********************************************************************
* Function:    PluginMainEntryPoint Main entry point of the Plugin     *
*                           PHASE.                                     *
* Parameters:  iFuncID      Function ID to get the entrypoint for      *
* Return:      Entrypoint of the requested Function or NULL            *
*********************************************************************/
FUNC_PTR PluginMainEntryPoint(int iFuncID)
{
  switch(iFuncID)
  {
    case PLGFUNC_SETUPPLUGIN:
      return((FUNC_PTR)fetchep((FETCH_PTR)SetupPlugin));
      break;
    case PLGFUNC_CLENUPPLUGIN:
      return((FUNC_PTR)fetchep((FETCH_PTR)CleanupPlugin));
      break;
    case PLGFUNC_GETHANDLEDCMDS:
      return((FUNC_PTR)fetchep((FETCH_PTR)GetHandledCommands));
      break;
    case PLGFUNC_SETUPHANLDER:
      return((FUNC_PTR)fetchep((FETCH_PTR)SetupHandler));
      break;
    case PLGFUNC_EXECUTEHANLDER:
      return((FUNC_PTR)fetchep((FETCH_PTR)ExecuteHandler));
      break;
    case PLGFUNC_CLEANUPHANLDER:
      return((FUNC_PTR)fetchep((FETCH_PTR)CleanupHandler));
      break;

    default:
      return(NULL);
      break;
  };
  return(NULL);
};
```

図 *143. PluginMainEntryPoint* 関数をインプリメントするためのサンプル・コード

SetupPlugin 関数のインプリメント

VSE コネクター・サーバー は、この関数を、サーバー・プラグインがロードされた後で 呼び出します。 *SetupPlugin* 関数は以下のことを実行します。

- プラグインに必要なすべての初期化ステップを実行します。
- 以下のことについての情報が入っている PLUGIN_INFO ブロックを指すポインターを取得します。
  - プラグイン構成メンバーに指定されている z/VSE システム・パラメーター。
  - VSE コネクター・サーバー が提供するユーティリティー関数のエントリー・ポイント。

また、ユーザーは、ユーザー独自のプラグイン専用制御ブロックを割り振り、このブロックのポインターを VSE コネクター・サーバー に戻すこともできます。 VSE コネクター・サーバー は、このポインターを、この時点から今後呼び出されるユーザーのプラグインに属す各関数に受け渡します。

注:

1. 1 つのプラグインを複数回定義した場合は、そのプラグインは複数回ロードされます。

2. *SetupPlugin* は、それぞれのプラグイン「インスタンス」ごとに別々に呼び出されます。
3. ユーザーが割り振るプラグイン専用制御ブロックのすべてのオカレンスは、相互に分離されます。

次に、*SetupPlugin* 関数のコーディングの例を示します。

```
/*********************************************************************
* Function:    SetupPlugin   Sets up the Plugin. The Plugin may alloc  *
*                            resources used for command processing.    *
*                            The Plugin can allocate a PluginPrivate   *
*                            Data Area, that is passed to each function*
* Parameters:  lplpPluginPrivate Pointer to a Pointer to the Plugins   *
*                            private Data area. This pointer should be *
*                            set by the Plugin.                        *
*              lpPluginInfo  Pointer to a struct PLUGIN_INFO containing*
*                            several information about the server      *
* Return:      Error values see PLGERR_xxx values                      *
*********************************************************************/
int SetupPlugin(void** lplpPluginPrivate,PLUGIN_INFO* lpPluginInfo)
{
  SAMPLE_PLUGIN*  lpSamplePlugin;

  /* Parameter checking ... */
  if(lplpPluginPrivate==NULL || lpPluginInfo==NULL)
    return(PLGERR_INVALID_PARAM);

  /* Allocate Plugin-Private Data area */
  *lplpPluginPrivate = malloc(sizeof(SAMPLE_PLUGIN));
  if(*lplpPluginPrivate==NULL)
    return(PLGERR_NULL_POINTER);

  /* Fill the Plugin-Info */
  lpPluginInfo->dwPluginVersion = PLUGIN_VERSION;
  strcpy(lpPluginInfo->szDescription,PLUGIN_DESCRIPTION);

  /* Initialize the Plugin ... */

  lpSamplePlugin = (SAMPLE_PLUGIN*)*lplpPluginPrivate;

  /* Open SYSLOG as trace output */
  lpSamplePlugin->lpOutput = fopen("DD:SYSLOG","w");
  if(lpSamplePlugin->lpOutput==NULL)
    return(PLGERR_NULL_POINTER);

  fprintf(lpSamplePlugin->lpOutput,"SetupPlugin Called¥n");

  return(PLGERR_NO_ERROR);
};
```

図 *144. SetupPlugin* 関数をインプリメントするためのサンプル・コード

CleanupPlugin 関数のインプリメント

VSE コネクター・サーバー は、この関数を、サーバー・プラグインがアンロードされる前に 呼び出します。 これは、*SetupPlugin* 関数の相手側の関数です。

SetupPlugin 関数は、必要なすべてのクリーンアップ・ステップを実行します。 これにより、プラグインは、前に割り振られたプラグイン専用制御ブロックを割り振り解放します。

次に、*CleanupPlugin* 関数のコーディングの例を示します。

```
/*********************************************************************
* Function:    CleanupPlugin Cleans up the Plugin.                   *
* Parameters:  lpPluginPrivate Pointer to the Plugins private data area*
* Return:      Error values see PLGERR_xxx values                    *
*********************************************************************/
int CleanupPlugin(void* lpPluginPrivate)
{
  SAMPLE_PLUGIN*  lpSamplePlugin;

  /* Parameter checking... */
  if(lpPluginPrivate==NULL)
    return(PLGERR_INVALID_PARAM);

  /* Cleanup the Plugin ... */

  lpSamplePlugin = (SAMPLE_PLUGIN*)lpPluginPrivate;

  fprintf(lpSamplePlugin->lpOutput,"CleanupPlugin Called¥n");

  /* Close trace output */
  fclose(lpSamplePlugin->lpOutput);

  /* Free the Plugin Private Data */
  free(lpPluginPrivate);

  return(PLGERR_NO_ERROR);
};
```

図 *145. CleanupPlugin* 関数をインプリメントするためのサンプル・コード

GetHandledCommands 関数のインプリメント

VSE コネクター・サーバー は、この関数を、*SetupPlugin* が完了した直後に呼び出します。 この関数は、あるサーバー・プラグインによって、どのコマンド (要求) が、および、どれだけ多くのコマンド (要求) が現在処理されているかを判別するために使用されます。

GetHandledCommands は、command-Ids の配列を VSE コネクター・サーバー に戻します。

次に、*GetHandledCommands* 関数のコーディングの例を示します。

```
/*********************************************************************
* Function:    GetHandledCommands Returns a list of CommandIDs that are*
*                          handled by this Plugin.                   *
* Parameters:  lpPluginPrivate Pointer to the Plugins private data area*
*              lpiNumCommands Pointer to a int that should be set by   *
*                          Plugin to the number of Commands          *
*              lpCommandIDs  Pointer to a array of ints. Each element  *
*                          defines one CommandID                     *
* Return:      Error values see PLGERR_xxx values                     *
*********************************************************************/
int GetHandledCommands(void* lpPluginPrivate,
                       int* lpiNumCommands,
                       int** lpCommandIDs)
{
  SAMPLE_PLUGIN*  lpSamplePlugin;

  /* Parameter Checking ... */
  if(lpPluginPrivate==NULL ||
     lpiNumCommands==NULL ||
     lpCommandIDs==NULL)
    return(PLGERR_INVALID_PARAM);

  lpSamplePlugin = (SAMPLE_PLUGIN*)lpPluginPrivate;

  fprintf(lpSamplePlugin->lpOutput,"GetHandledCommands Called¥n");

  /* There are 2 Command sHandled by the Sample */
  *lpiNumCommands = 2;

  /* Return a Pointer to the List of Command-IDs */
  *lpCommandIDs = &HandledCommandIDs[0];

  return(PLGERR_NO_ERROR);
};
```

図 *146. GetHandledCommands* 関数をインプリメントするためのサンプル・コード

HandledCommandIDs 配列は、次のように定義されます。

```
int   HandledCommandIDs[2] = { CMD_SAMPLE_TIME,
                               CMD_SAMPLE_DATE };
```

SetupHandler 関数のインプリメント

VSE コネクター・サーバー は、サーバー・プラグインによって処理されなければならない要求が受け取られるたびに、この関数を呼び出します。

1. この関数は、その要求のための要求処理プログラムを初期化します。
2. VSE コネクター・サーバー は、CMD_INFO 制御ブロックを指すポインターをこの関数に受け渡します。
3. CMD_INFO 制御ブロックには、処理するコマンドに関する情報 (例えば、要求のコマンド ID) が入ります。 ユーザーは、ハンドラーの専用制御ブロックを割り振り、このブロックを指すポインターをサーバーに戻すことができます。 サーバー・エンジンは、このポインターを、同じ要求に属す各関数呼び出しに受け渡します。

注: 1 つの要求を複数回実行することが可能です。 要求のそれぞれの「インスタンス」ごとに、関数 *SetupHandler* が別々に呼び出されます。 つまり、ユーザーが割り振るハンドラーの専用制御ブロックは相互に分離されます (ユーザーのコーディングでは、このことを考慮する必要があります)。

また、ハンドラーは、必ず、再入可能 としてインプリメントする必要があります。

次に、*SetupHandler* 関数のコーディングの例を示します。

```
/*********************************************************************
* Function:    SetupHandler  Sets up a Command Handler.               *
* Parameters:  lpPluginPrivate Pointer to the Plugins private data area*
*              lplpCommandPrivate Pointer to a Pointer to the Handlers *
*                           private data area                          *
*              lpCmdInfo     pointer to a struct CMD_INFO defining the *
*                           actual command                             *
* Return:      Error values see PLGERR_xxx values                     *
*********************************************************************/
int SetupHandler(void* lpPluginPrivate,
                 void** lplpCommandPrivate,
                 CMD_INFO* lpCmdInfo)
{
  SAMPLE_PLUGIN*  lpSamplePlugin;

  /* parameter Checking ... */
  if(lpPluginPrivate==NULL ||
     lplpCommandPrivate==NULL ||
     lpCmdInfo==NULL)
    return(PLGERR_INVALID_PARAM);

  lpSamplePlugin = (SAMPLE_PLUGIN*)lpPluginPrivate;

  /* Initialize the Handler... */
  fprintf(lpSamplePlugin->lpOutput,"SetupHandler Called¥n");

  /* check which Command we should handle */
  switch(lpCmdInfo->Command.dwCommand)
  {
    case CMD_SAMPLE_TIME:  /* We are Handling Time-Command */
      /* allocate Command Private datat area */
      ...
      break;

    case CMD_SAMPLE_DATE:   /* We are Handling Date Command */
      /* allocate Command Private datat area */
      ...
      break;
  };
  return(PLGERR_NO_ERROR);
};
```

図 *147. SetupHandler* 関数をインプリメントするためのサンプル・コード

ExecuteHandler 関数のインプリメント

VSE コネクター・サーバー は、ハンドラーの処理を実行するために、この関数を呼び出します。 VSE コネクター・サーバー は、プラグインの専用制御ブロックおよびハンドラーの専用制御ブロックを、この関数に受け渡します。 また、処理するコマンドに関する情報が入っている CMD_INFO ブロックを指すポインターも受け渡されます。 またこのブロックは、VSE コネクター・サーバーに、*ExecuteHandler* 関数がそれに制御を返した後でどのようなアクションをとるべきかを指示するためにも使用されます。

VSE コネクター・サーバー は、単一タスクでのみ実行されます。 したがって、ユーザーのサーバー・プラグインも、1 つの (メイン) タスクで実行されます。 したがって、*ExecuteHandler* 関数 (およびその他のすべての関数) が、制御を、VSE コネクター・サーバー にできるだけ早急に戻すようにしなければなりません。 これを確実に行うようにするには、要求処理を複数の小さなチャンクに分割し、それぞれのチャンクが最小の時間で実行できるようにしなければなりません。 要求処理を複数のチャンクに分割する方法の例については、「日時サンプル・プラグイン」 (313 ページの『IBM 提供のサーバー・プラグインの使用の例』に説明があります) を参照してください。

以下の場合には、ユーザーの要求処理プログラムは、アクション・コマンドを CMD_INFO ブロックにセットアップし、制御を VSE コネクター・サーバー に戻すことができます。

- ECB (イベント制御ブロック) を待つ必要がある場合
- タイマーを待つ必要がある場合
- クライアントからのデータの受け取りを待つ必要がある場合
- クライアントへのデータの送信を待つ必要がある場合

VSE コネクター・サーバー は、ユーザーが指定したイベントを待ち、制御をユーザーのプラグインに受け渡し、さらに、イベントが行われた後で制御をユーザーのプラグインに受け渡します。 したがって、ユーザーのプラグイン自体が待つ必要はありません。これは、プラグインが待つことにより、VSE コネクター・サーバー によって実行中の、並行して実行されるタスクのすべてがブロックされることになるからです。

また、ネットワークを使用してデータの送受信を行いたい場合は、アクション・コマンドをセットアップすることもできます。 ユーザーのプラグインは、ネットワーク・サービスに直接アクセスできません。ユーザーのプラグインに代わって、VSE コネクター・サーバー が、ネットワークを使用したデータの送受信を処理します。

次に、*ExecuteHandler* 関数のコーディングの例を示します。

```
...
/* Check which Command to handle */
switch(lpCmdInfo->Command.dwCommand)
{
  case CMD_SAMPLE_TIME:  /* Time Command */
    ...
    break;

  case CMD_SAMPLE_DATE:  /* Date Command */
    ...
    break;

  ...
};
```

図 *148. ExecuteHandler* 関数をインプリメントするためのサンプル・コード

309 ページの図 149 のコードのフラグメントは、1 つのプラグイン内の複数の要求を区別する方法を示しています。 要求の処理をいくつかの小さなチャンクに分割するために、CMD_INFO ブロックには、ハンドラーの実際の状態を保管するため

に使用できる *iState* というフィールドが入っています。

```
/* Check in which state we are (0 for first call) */
switch(lpCmdInfo->iState)
{
  case 0:  /* Initial State */
    /* Verify that the Command is correctly sent */
    if(lpCmdInfo->Command.dwDataSize!=0)
      return(PLGERR_PROTOCOL_ERROR);

    /* OK, start processing */
    ...
    /* return from handler without any special action */
     lpCmdInfo->iAction = PLGACT_NOTHING;
     lpCmdInfo->iState = 1;
     break;

   case 1: /* Send the Rresponse */
     /* Send the Response Command */
     /* Setup the COMMAND-Struct for Response */
     memset(&lpCmdInfo->Response,0,sizeof(COMMAND));
     lpCmdInfo->Response.dwCommand = RES_SAMPLE_TIME;
     lpCmdInfo->Response.dwDataSize = sizeof(SAMPLE_TIME);

     /* Setup Action and State */
     lpCmdInfo->iAction = PLGACT_SEND_RESP;
     lpCmdInfo->iState = 2; /* State after Send-Command */
     break;

   case 2: /* state after send reponse */
     ...

}
```

図 *149.* プラグイン内の複数の要求を区別するためのサンプル・コード

要求が受け取られると、VSE コネクター・サーバー は、フィールド *iState* をゼロにセットします。 *ExecuteHandler* 関数は、このフィールドに、その実際の状態を表す値をセットすることができます。 *iState* フィールドは、次の呼び出しで、そのまま変更されずに、*ExecuteHandler* 関数に受け渡されます。 VSE コネクター・サーバー は、アクション PLGACT_FINISH が戻されるまで、クライアントがディスパッチされるたびに、*ExecuteHandler* 関数を呼び出し続けます。

CleanupHandler 関数のインプリメント

VSE コネクター・サーバー は、この関数を、要求が実行された後で呼び出します。 これは、*SetupHandler* 関数に対する対照関数です。 この関数はハンドラーをクリーンアップし、ハンドラーの専用制御ブロックを解放します (*SetupHandler* 関数の実行中に割り振られている場合)。

CleanupHandler 関数は、通常、要求の処理が完了した (すなわち、*ExecuteHandler* 関数がアクションを PLGACT_FINISH にセットした) 後で呼び出されます。 ネットワーキング・エラー (例えば、接続が切断される) が発生した場合、VSE コネクター・サーバー は *CleanupHandler* 関数を呼び出します。 *CleanupHandler* 関数は、常に、ハンドラーをクリーンアップできなければなりません。 ハンドラーの状態を検査し、さらに、ハンドラーによって割り振られていたすべてのリソースをクリーンアップしなければなりません。

ユーザー独自のプラグイン・コールバック関数の作成

前に説明したプラグイン・コールバック関数は、ユーザーのサーバー・プラグインがインプリメントしなければならない関数で、VSE コネクター・サーバー によって呼び出される関数です。 ただし、ユーザーは、コードを、さらにプラグイン・コールバック関数に分割できます。また、コードをいくつかのモジュールに分割することもできます。

前に述べたように、VSE サーバー・プラグインは、LE/VSE-C でインプリメントされるように設計されています。 これを C でインプリメントしない場合は、以下のいずれかの方法をとることができます。

- 完全なプラグインを別のプログラム言語 (PL1 またはアセンブラーなど) でインプリメントする。 ただし、作成されたコードが LE/VSE に準拠し、さらに、LE/VSE で使用される呼び出し規則をサポートする必要があります。
- スケルトンを、別のプログラム言語でインプリメントされた関数を呼び出す小さなスタブ・コードとして使用する。 これは、推奨されるメソッドです。

図 150 は、2 番目のメソッド (スケルトンを小さなスタブ・コードとして使用する) を示すコードのチャンクです。 この例では、*MyASMExecuteFunction* という関数が (シンボル MYASMEXE を使用して) 呼び出されます。呼び出し規則のオペレーティング・システムが、必要なパラメーターを受け渡します。

```
/* define the ASM function */
#pragma map(MyASMExecuteFunction,"MYASMEXE")
#pragma linkage(MyASMExecuteFunction,OS)

int ExecuteHandler(void* lpPluginPrivate,
                   void* lpCommandPrivate,
                   CMD_INFO* lpCmdInfo)
{
  int rc;

  rc = MyASMExecuteFunction(lpCmdInfo);
  /* TODO errro checking ... */

  return(PLGERR_NO_ERROR);
};
```

図 *150. C* 以外の言語で作成されたスタブ・コードを呼び出す例

VSE コネクター・サーバー でサポートされているアクション・コード

ExecuteHandler 関数から戻るときには、CMD_INFO ブロックのフィールド *iAction* に、アクション・コードをセットアップする必要があります。 このアクション・コードによって、VSE コネクター・サーバー に、次の反復では何をすべきかを伝えます。可能なアクション・コードは、以下のとおりです。

PLGACT_NOTHING
: 特別なアクションではなく、ハンドラーは、できるだけ早く再び呼び出されることを要求しています。

PLGACT_SEND

ハンドラーはクライアントにデータを送信する必要があります。 ハンドラーは、以下のフィールドを CMD_INFO ブロックにセットアップする必要があります。

lpData

バッファーを指すポインター

dwDataSize

送信するデータのサイズ (バイト単位)

注: バッファーは、ハンドラーの専用制御ブロックの中になければなりません。 これはローカル変数であってはなりません。ローカル変数は関数内でのみ有効であるからです。 ハンドラーは、すべてのデータが送信されたあとで、制御を取得します。

PLGACT_RECEIVE

ハンドラーは、クライアントから、データを受け取る必要があります。 ハンドラーは、以下のフィールドを CMD_INFO ブロックにセットアップする必要があります。

lpData

バッファーを指すポインター

dwDataSize

受け取るデータのサイズ (バイト単位)

注: バッファーは、ハンドラーの専用制御ブロックの中になければなりません。 これはローカル変数であってはなりません。ローカル変数は関数内でのみ有効であるからです。 ハンドラーは、すべてのデータが受け取られたあとで、制御を取得します。

PLGACT_SEND_RESP

ハンドラーはクライアントに応答ヘッダーを送信する必要があります。 ハンドラーは、Response というフィールドを CMD_INFO ブロックにセットアップする必要があります。 ハンドラーは、応答ヘッダーが送信されたあとで、制御を取得します。

PLGACT_FINISH

ハンドラーは、VSE コネクター・サーバー に、要求の処理が完了したことを伝えます。 VSE コネクター・サーバー は、このアクションの結果、*CleanupHandler* 関数を呼び出しますが、この要求のために ExecuteHandler 関数を呼び出すことはありません。

PLGACT_CHECKCANCEL

このアクションは PLGACT_NOTHING に似ていますが、VSE コネクター・サーバー は、取り消しバイトが受け取られるようになっているかどうかを検査します。 フラグ FLG_CANCANCEL がコマンド・ヘッダーに指定されている場合は、要求は取り消し可能です。 VSE コネクター・サーバー は、取り消しバイトが受け取られた場合、フィールド bWasCanceled に TRUE をセットします。

PLGACT_WAIT

ハンドラーは、ECB (イベント制御ブロック) を待つ必要があります。 ハン

ドラーは、フィールド *lpECB* を、ECB を指すポインターにセットアップする必要があります。ハンドラーは、ECB がポストされた後で、制御を取得します。

PLGACT_WAITRECV
このアクションは PLGACT_WAIT に似ていますが、受け取るデータが使用可能になるのを待ちます。 すなわち、このハンドラーは、ECB がポストされた後か、あるいは、受け取るデータが使用可能になった後で、制御を取得します。 注: このアクションでは、受け取られるデータはありません。

PLGACT_WAITTIMER
ハンドラーは、指定した時間だけ待つ必要があります。 ハンドラーは、待つ秒数を、CMD_INFO ブロック内のフィールド *iTimer* にセットアップする必要があります。 ハンドラーは、指定した時間が経過した後で、制御を取得します。

VSE コネクター・サーバー でサポートされているユーティリティー関数

VSE コネクター・サーバー は、サーバー・プラグイン内から呼び出すことができるいくつかのユーティリティー関数 をサポートしています。 プラグインは、以下のことを行います。

- *SetupPlugin* 関数に受け渡される PLUGIN_INFO 制御ブロック内にある関数のエントリー・ポイントを取得します。
- プラグインが必要とする関数のエントリー・ポイントを、そのプラグインの専用制御ブロックに保管します。

以下のユーティリティー関数がサポートされています。

StrToAscii
VSE コネクター・サーバー の構成メンバーの中に構成されているコード・ページを使用して EBCDIC ストリングを ASCII に変換します。

StrToEbcdic
VSE コネクター・サーバー の構成メンバーに構成されているコード・ページを使用して ASCII ストリングを EBCDIC に変換します。

StrToUpcase
ストリングを大文字に変換します。

StrTrim
先頭ブランクと末尾ブランクをストリングから除去します。

StrReplace
ストリングの中の任意の文字を、別の文字で置き換えます。

MatchWildCards
文字列がワイルドカード・フィルター (「*」および「?」がある) にマッチングするか検査します。

EncryptData
ランダム XOR アルゴリズムを使用して、データのバッファーを暗号化します。

Decryptdata
ランダム XOR アルゴリズムを使用して、データのバッファーを復号します。

CheckLibrSecurity
Libr リソースへのアクセスが保証されているか検査します。

CheckPOWERSecurity
POWER リソースへのアクセスが保証されているか検査します。

CheckVSAMSecurity
VSAM リソースへのアクセスが保証されているか検査します。

CheckConsoleSecurity
Console リソースへのアクセスが保証されているか検査します。

CheckICCFSecurity
ICCF リソースへのアクセスが保証されているか検査します。

IBM 提供のサーバー・プラグインの使用の例

IESPLGIN.H および IESPLGSK.C は、VSE コネクター・クライアント で提供されているメンバーであり、「日時プラグイン」の例のサーバー・プラグイン を作成するために、これらのメンバーが使用されています。 ユーザー独自のサーバー・プラグインを作成する際には、次に説明する、これらのメンバーの内容を使用することができます。

- IESPLGIN.H は、プラグインのプログラミング・インターフェースを定義するために使用される C ヘッダー・ファイルです。
- IESPLGSK.C は、プラグインの基本になるスケルトンの C プログラムです。 このスケルトンには、ユーザー独自のサーバー・プラグインを作成するときに必要なすべての関数が入っています。 また、このスケルトンには、プラグインを作成するときに役立つコメントも入っています。

サーバー・プラグインの登録およびコンパイル

1. ユーザーのサーバー・プラグインは、VSE コネクター・サーバーのプラグイン構成メンバー (通常は IESPLGIN.L) に登録する必要があります。このメンバーは、1 つ以上のテキスト行から成り、それぞれの行で 1 つのプラグインを定義します。 各行の構文は、以下のようになります。

   ```
   PLUGIN=<phase name>,PARM=<any kind of parameters>
   ```

   キーワードは以下のように使用します。

   - キーワード PLUGIN は、サーバー・プラグインの名前を入力するために使用します。
   - キーワード PARM は、サーバー・プラグインのパラメーター・リストを受け渡すために使用します。 このパラメーター・リストを使用して、ユーザーのサーバー・プラグインを構成することができます。 注: キーワード PARM は、プラグインにパラメーターを受け渡さない場合でも、使用しなければなりません。
2. VSE コネクター・サーバー スタートアップ・ジョブの LIBDEF チェーンの中にフェーズ名 を入力してください (詳しくは 32 ページの『ジョブ SKVCSSTJ - スタート・アップ・ジョブ』を参照)。

3. サーバー・プラグインをコンパイルするには、次に示すようなジョブを使用してください。プラグインは、再入可能オプション (RENT) を使用してコンパイルする必要があります。

```
* $$ JOB JNM=COMPILE,CLASS=4,DISP=L
* $$ LST DISP=D,CLASS=A,RBS=100
// JOB COMPILE AND LINK VSE CONNECTOR SEVRER PLUGIN
// LIBDEF *,SEARCH=(YOURLIB.YOURSLIB,PRD2.SCEEBASE,PRD2.PROD,          X
               PRD2.DBASE)
// LIBDEF PHASE,CATALOG=YOURLIB.YOURSLIB
// OPTION ERRS,SXREF,SYM,NODECK,CATAL,LISTX
   PHASE YOURNAME,*,SVA
// EXEC EDCCOMP,SIZE=EDCCOMP,PARM='LONGNAME RENT SS SOURCE             X
               INFILE(DD:YOURLIB.YOURSLIB(YOURNAME.C))'
/*
// EXEC EDCPRLK,SIZE=EDCPRLK,PARM='UPCASE MAP'
/*
// EXEC LNKEDT,SIZE=512K
/&
* $$ EOJ
```

4. これで、VSE コネクター・サーバー を再始動することができます。再始動すると、次のメッセージがシステム・コンソールに書き込まれます。

```
Fn nnnn LOADING PLUGIN:  <your plugin>
```

クライアント・プラグインのインプリメント

インプリメントするそれぞれのサーバー・プラグイン (298 ページの『サーバー・プラグインのインプリメント』に説明があります) ごとに、クライアント・プラグイン もインプリメントする必要があります。 クライアント・プラグインは、サーバー・プラグインがサポートするそれぞれの要求ごとに 1 つの JavaBean で構成されます。

次に、クライアント・プラグインを開発するための要件を示します。 以下が必要です。

- JDK (Java Development Kit) がインストールされている (バージョン 1.1.6 以上)。
- VSE コネクター・クライアント がインストールされている。 少なくとも、JAR ファイル **VSEConnector.jar** が、クラスパスに入っていなければなりません。
- VSE Java Beans の JavaDoc を API リファレンスとして使用する。

ユーザーがインプリメントする JavaBeans では、VSE Java Beans で提供される通信サービスを使用します。 したがって、ユーザーの JavaBeans で、VSE コネクター・サーバー への接続を確立する必要はありません。これは、ユーザーのプラグインが使用する VSE Java Beans のそれぞれが、*VSEResource* クラスのサブクラスであるからです。 ユーザーの JavaBeans は、クラス *VSESystem* のインスタンスを使用して z/VSE ホストを識別します。

VSESystem のセットアップは、その他のすべての VSE Java Beans (例えば、*VSELibrarian* Bean) の場合と同じです。 つまり、

1. *VSEConnectionSpec* を作成し、必要なプロパティーをセットします。
2. *VSESystem* を作成し、*VSEConnectionSpec* を使用してターゲットの z/VSE ホストを識別します。

3. ユーザーの Plugin Bean のインスタンスを作成します (*VSESystem* オブジェクトがある場合)。

次に、VSESystem をセットアップするためのコードのアウトラインを示します。

```
VSEConnectionSpec spec =  new VSEConnectionSpec(...);
...
VSESystem system = new VSESystem(spec);
...
MyPluginBean myPlugin = new MyPluginBean(system);
```

VSEPlugin クラスの使用

ユーザーが作成するクライアント・プラグインである JavaBean は、いずれも、*VSEPlugin* のサブクラスでなければなりません。 *VSEPlugin* は抽象クラスであり、それ自体が *VSEResource* のサブクラスです。 また、VSE Java Beans の JavaDoc でおわかりのように、*VSEResource* は次のメソッドをインプリメントしています。

setVSESystem
使用する *VSESystem* をセットします。

getVSESystem
現在使用されている *VSESystem* を取得します。

addVSEResourceListener
Bean に *VSEResourceListener* を追加します。

removeVSEREsourceListener
VSEResourceListener を除去します。

notifyListStarted
登録されたすべての *VSEResourceListeners* に通知します。

notifyListAdded
登録されたすべての *VSEResourceListeners* に通知します。

notifyListEnded
登録されたすべての *VSEResourceListeners* に通知します。

setCancel
現在の要求を取り消します。

getCancel
現在のキャンセル状態を取得します。

isExistent
リソースが存在するかどうか検査します。

toString
ストリング表記を戻します。

さらに、ユーザー独自のメソッドをインプリメントしたり、ご使用の JavaBean にプロパティーを追加することができます。

クライアント・プラグイン JavaBean の要求を活動化するために、メソッド *execute()* を呼び出します。 このメソッドは、クラスの内側からのみ呼び出されなければならないので、「保護される」として宣言されます。 例えば、メソッド *refresh* をオーバーライドして *execute* を呼び出し、JavaBeans プロパティーの最新表示を

要求できます。 メソッド *execute* は要求の実行をトリガーし、要求が処理されると制御を戻します。 要求の処理中に、以下のメソッドが呼び出されます。

- *getRequestID*
- *getRequestSubcommand*
- *getRequestFlags*
- *getRequestError*
- *getRequestDataSize*
- *sendRequestdata*
- *getResponseID*
- *getResponseDataSize*
- *receiveResponseData*

サーバー・プラグインとクライアント・プラグインが一緒に作業できるようにするには、次のことが必要です。

- *getRequestID* メソッドおよび *getResponseID* メソッドによって戻された Command Ids は、サーバー・プラグインで使用された CommandIDs に一致しなければなりません。
- 要求または応答のために転送されるデータ・フォーマットは、サーバー・プラグインでサポートされているデータ・フォーマットに一致しなければなりません。
- プラグイン JavaBean は、プラグイン・サーバーとプラグイン・クライアントとの間の通信に使用されるプロトコルを使用しなければなりません。

getRequestDataSize メソッドおよび *getResponseDataSize* メソッドは、次のように、転送されるデータのサイズ (バイト単位) を戻す必要があります。

- データが転送されない場合は、このメソッドはゼロを戻す必要があります。
- 要求または応答が DATAEX フォーマットを使用する場合は、メソッドは定数 SIZE_UNKNOWN を戻す必要があります。 これは、コマンドに、DATAEX コマンドとしてフラグを立てます。
- また、*sendRequestData* メソッドまたは *receiveResponseData* メソッドは、それぞれ、送信されたデータのサイズ、または、受信されたデータのサイズを戻す必要があります。

さらに、

- DATAEX コマンドの場合、メソッドは、コマンドが終了するまで制御を保持する必要があります。 データ転送が完了した後で、メソッドは制御を戻してコマンドの終了を示します。
- DATAEX 以外のコマンドの場合、*sendRequestData* メソッドまたは *receiveResponseData* メソッドの戻り値は、それぞれ、*getRequestDataSize* または *getResponseDataSize* によって戻された値に等しくなければなりません。

クライアント・プラグインは、以下のメソッドを、

- *getResponseSubCommand*
- *getResponseFlags*
- *getResponseError*

メソッド *receiveResponseData()* の中、あるいは、メソッド *execute()* から制御が戻った後で、呼び出すことができます。 これらのメソッドを呼び出すことによって、クライアント・プラグインは、応答の中で送信されたサブコマンド、フラグ、およびエラー値を取得することができます。

これにより、クライアント・プラグインは、これらの値を使用して、エラー検査とエラー・ハンドリングを実行できます。

プラグインを設計するときの一般的な考慮事項

プラグインを設計するときは、以下の点を考慮してください。

- VSE コネクター・サーバー とサーバー・プラグインとの間のプロトコルはどのように定義すべきか?
- z/VSE 上のリソースまたはアプリケーションにはどのようにアクセスできるか?
- どの種類のデータにアクセスする必要があるか?
- どの要求あるいは関数を許可すべきか?
- データ・フォーマットは、ネットワーク上でどのように転送すべきか?
- JavaBean インターフェースを定義するときに、クライアント・プラグイン用のデータのビューをどのように構成すべきか?

VSE コネクター・サーバー とプラグインとの間のプロトコルの指定

Java ベース・コネクターは、その独自のプロトコル を通信目的で使用します。 ただし、ユーザーは、このプロトコルを、ユーザー独自のコマンドで拡張することができます。 Java ベース・コネクターで使用するこのプロトコルの特性については、このトピックで説明します。

Java ベース・コネクターで使用されるプロトコルは、TCP/IP 接続を基にしています。このプロトコルは、データを転送するために、接続指向の ストリームを使用します。 (エンドレス) ストリームは、ストリーム・コマンド と呼ばれる小さな部分に分けられます。 ストリーム・コマンドは、以下のもので構成されます。

- コマンド・ヘッダー
- データ部分 (オプショナル)

さまざまなストリーム・コマンドは、4 バイトの固有の command-Id で識別され、コマンド・ヘッダーの中に保管されます。 コマンド・ヘッダーには、以下のオプショナル・フィールドがあります。

フィールド
: 説明

Flags ビット・フラグ (4 バイト)。 詳しくは、事前定義 Flag 値を参照。

Error エラー・コード (4 バイト)。 詳しくは、事前定義エラー・コードを参照。

SubCommand
: SubCommand ID (4 バイト)。 将来の利用に限られます。

コマンド・ヘッダーの直後には、ストリーム・コマンドに付随するデータが続きます。 ストリーム・コマンドにデータがない場合は、次のストリーム・コマンドのコマンド・ヘッダーが続きます。

ストリーム・コマンドに付随するデータは、次の 2 つの方法で転送されます。

- 固定長データ
- 可変長データ (DATAEX)

データが固定長 として転送されるときは、データ・バイトの数がコマンド・ヘッダーに定義されます。 このバイト数のあとに、次のコマンド・ヘッダーが始まります。

データが可変長 (「DATAEX」メソッド) として転送されるときは、データの長さは直接指定されませんが、データ内に暗黙的に指定されます。 したがって、ユーザー・プラグインを開発するときは、転送されるデータの中に、データの終わりをマークする任意の標識を定義することができます。 データの送信側と受信側の両方が、この標識を認識できなければなりません。 可変長データの終わりが来たあとに、次のコマンド・ヘッダーが続きます。

プラグインを開発するときに、ユーザー独自の要求を使用して、一連の既存の要求を拡張できます。 このためには、以下を定義する必要があります。

- ユーザーの要求と応答で使用するコマンド ID。
- コマンドに付随するデータのフォーマット。

データ/アプリケーションにアクセスするためのアクセス方式の選択

まず、プラグインが必要なデータまたはアプリケーションにアクセスするために、z/VSE で使用するアクセス方式を選択する必要があります。 VSE コネクター・サーバー は、バッチ 区画 (静的または動的) で実行されるので、ご使用のプラグインも、同じバッチ環境で実行されます。 したがって、選択するアクセス方式は、データまたはアプリケーションに、バッチ ・プログラム内からアクセスできなければなりません。

プラグインは、LE/VSE 環境内で作成する必要があります。 VSE コネクター・サーバー のプラグインは、VSE コネクター・サーバー 自体が LE/VSE C でインプリメントされているので、LE/VSE C でインプリメントする必要があります。ただし、このプラグイン内から、アセンブラー・モジュールまたは PL/1 モジュールを呼び出すことができます。

また、プラグインを作成するときに VSE マクロ (例えば GETVIS) を使用することができますが、LE/VSE のレジスターの使用は、ユーザー自身でプログラムしなければなりません。

ASCII/EBCDIC およびビッグ/リトル・エンディアンに関する考慮事項

どの種類のデータにも、アクセスすることができます。 テキスト・データにアクセスすることにした場合は、ユーティリティー関数を使用して ASCII と EBCDIC の

間の変換を行ってから、データをネットワークで転送しなければなりません。 必要な変換は、ユーザーのプラグインで実行する必要があります。

ASCII と EBCDIC の間の変換はクライアント・サイドで実行できますが、この変換は、サーバー・サイドで実行することをお勧めします。 ASCII から EBCDIC への (およびその逆の) 変換を行うユーティリティー関数は、変換を行うために構成されたコード・ページを使用します。 整数値のようなバイナリー・データを転送する場合は、通常、ビッグ・エンディアン・フォーマットまたはリトル・エンディアン・フォーマットに関して気遣う必要はありません。

z/VSE (すなわち System z プラットフォーム) は、整数値についてはビッグ・エンディアン・フォーマットを使用します。Java も、Intel (リトル・エンディアン) プラットフォームで実行している場合であっても、ビッグ・エンディアン・フォーマットを使用します。 したがって、整数値は、転送する前に変換する必要はありません。

どの要求/関数をサポートすべきかについての決定

ユーザーのプラグインで、どの種類のアクセス要求あるいは関数をサポートすべきかを決定しなければなりません。 1 つの VSE コネクター・サーバー プラグインで、1 つまたは複数の異なる要求をサポートできます。

1 つの種類のアクセス (読み取り、書き込み、更新、...) のそれぞれに 1 つの要求を使用すること、あるいは、1 つの要求だけを使用して、さまざまな種類のアクセスを実行できるようにすることもできます。 後者の場合、書き込みまたは読み取り操作が要求されたことをプラグインに通知するパラメーターを定義できます。

ネットワークを使用したデータの転送

また、ユーザーは、プラグインのサーバー部分とクライアント部分の間のインターフェースを定義しなければなりません。 つまり、データを転送するために使用するプロトコル を定義する必要があります。 よく定義されたプロトコルが、VSE コネクター・クライアント と VSE コネクター・サーバー との間の通信に使用されています。 したがって、ユーザーのプラグインのプロトコルも、このプロトコルをサポートしなければなりません。

VSE コネクター・クライアント と VSE コネクター・サーバー との間の通信に使用されるプロトコルは、要求 から成っています。 各要求は、

- ほかの要求から独立しています。
- ネットワークを使用して、ブロックとして転送されます。
- 4 バイトの番号 *command-Id* によって識別されます。

ユーザーのプラグインは、特定の範囲内の、独自の command-Ids を使用できます。 ユーザーのプラグインは、データを転送するためには事前定義されたプロトコルを使用しなければなりませんが、データのフォーマット自体は、ユーザーが定義することができます。

クライアント・プラグインのビューの構造化

また、ユーザーは、データまたはアプリケーションのクライアント部分のビューをどのように構造化すべきかについて考慮しなければなりません。 Java ベース・コ

ネクターを使用すると、Java ベース・コネクターによって提供される通信方式を使用するユーザー独自の JavaBeans プラグインをインプリメントできます。

それぞれの要求ごとに、クライアント・サイドで、対応する JavaBean をインプリメントしなければなりません。 要求に付随するデータの送受信は、この JavaBean が行います。 したがって、この JavaBean は、転送されるデータのフォーマットをよく知っていなければなりません。

ユーザーが開発するプラグイン JavaBeans の外部インターフェースは、ユーザーが設計できます。

第 22 章 データにアクセスするための データベース呼び出しレベル・インターフェース の使用

このトピックでは、アプリケーション・プログラムでデータベース呼び出しレベル・インターフェース (DBCLI) を使用して、適切なデータベース・サーバー上のリレーショナル・データベースにアクセスする方法について説明します。データベース・サーバーは、通常、非 z/VSE プラットフォームで稼働しています。

このトピックに含まれるのは次のとおりです。

- 『DBCLI プログラミングの概念』
- 326 ページの『プログラミングの制限と要件』
- 327 ページの『DBCLI を COBOL で使用』
- 328 ページの『DBCLI を PL/I で使用』
- 328 ページの『DBCLI を C で使用』
- 329 ページの『DBCLI をアセンブラーで使用』
- 330 ページの『DBCLI を REXX で使用 (バッチ)』
- 331 ページの『DBCLI を REXX/CICS で使用』
- 332 ページの『DBCLI 関数呼び出しの構文とパラメーター』
- 333 ページの『DBCLI 関数 (参照情報)』
- 436 ページの『DBCLI 使用時のパフォーマンスに関する考慮事項』
- 437 ページの『DBCLI 使用時のエラー原因の調査』
- 438 ページの『DBCLI で使用される戻りコード』
- 439 ページの『バッチ照会ツール』
- 444 ページの『対話式照会ツール』

関連トピック:

- 85 ページの『第 9 章 データベース呼び出しレベル・インターフェースのインストール』(データベース呼び出しレベル・インターフェースの概説 が含まれています)。
- COBOL の例 **COBSAMPL** (DBCLI サーバー・パッケージの一部として出荷されます)。この例では、COBOL プログラム内でデータベース呼び出しレベル・インターフェースを使用する方法を示しています。

DBCLI プログラミングの概念

API 環境の初期化と終了

DBCLI クライアントで提供される API を使用する場合は、以下の作業を実行する必要があります。

1. 他の関数を呼び出す前に、INITENV 関数を呼び出して API 環境を初期化します。 INITENV 関数は、後続のすべての関数に渡す必要がある環境ハンドル を割り振ります。プログラムで設定できるのは、一度に *1* つのアクティブ環境だけです。
2. TERMENV 関数を呼び出して、(プログラムの最後で) API 環境を終了します。TERMENV 関数は、DBCLI コードで割り振ったすべてのリソースを解放します。TERMENV 関数はまた、「残された」接続またはステートメントもすべて閉じます。 TERMENV 関数の実行後、環境ハンドルは無効になります。

お客様は、環境レベルで各種の属性を設定および取得できます。そのためには、SETENVATTR 関数または GETENVATTR 関数を呼び出します。

DBCLI サーバー およびベンダー・データベースとの接続および切断

特定のデータベースを使用する前に、*DBCLI* サーバー (DBCliServer) およびベンダー・データベースに接続する必要があります。

- DBCLI サーバー (DBCliServer) およびベンダー・データベースには、CONNECT 関数を呼び出して接続します。以下を入力する必要があります。
  - DBCliServer の IP アドレスまたはホスト名。
  - 接続先にするデータベースの別名。
- CONNECT 関数は、接続を必要とする後続のすべての関数に渡す必要がある接続ハンドル を割り振ります。
- どの時点でも、同じまたは異なる DBCLI サーバーおよびデータベースへの複数の接続 を持つことができます。
- 各接続は、その独自の接続ハンドルによって表現されます。必要な接続ハンドルの後続の関数への受け渡しは、アプリケーションが受け持つ作業です。

データベースでの作業を終了した場合は、DISCONNECT 関数を呼び出して、データベースと DBCLI サーバー (DBCliServer) から切断します。

- DISCONNECT 関数は、この接続を使用して割り当てた接続ハンドルとすべての残留ステートメント (もしあれば) を解放します。
- DISCONNECT 関数の後は、接続ハンドルは有効ではなくなります。

接続レベルでいくつかの属性を設定および取得できます。これは SETCONNATTR または GETCONNATTR 関数を呼び出すことによって実行できます。

準備する SQL ステートメントは、作成に使用した接続と強力にバインドされます。ある接続で作成されたステートメントを他の接続で使用することはできません。そのようにすると、エラーを受け取ります。

作業論理単位 (トランザクション)

デフォルトでは、接続はトランザクション・モード で動作します。

- 接続によって実行するデータベース・アップデートはいずれも、作業論理単位に含まれています。
- 作業論理単位を終了するには、COMMIT 関数または ROLLBACK 関数を呼び出します。

- COMMIT 関数は、その作業論理単位の開始以降に加えられた変更をすべてコミットして、新しい作業論理単位を開始します。
- ROLLBACK 関数は、その作業論理単位の開始以降、または保存ポイントまでに行われた変更をすべてロールバックします (元に戻します)。

通常、COMMIT 関数はプログラムの最後で明示的に呼び出す必要があります。

- COMMIT 関数を呼び出さない場合、DISCONNECT 関数を呼び出して正常に接続を閉じると、DBCliServer はすべての変更を自動的にコミットします。
- 接続が切断されると (例えばプログラムの異常終了など)、DBCLI サーバー (DBCliServer) は、最後の作業論理単位の開始以降に加えられた変更をすべてロールバックします。

接続を自動コミット・モード に設定することができます。

- 自動コミット・モードでは、SQL ステートメントはそれぞれ独自の作業論理単位として扱われ、ステートメントの実行が完了すると自動的にコミットされます。そのため、COMMIT 関数や ROLLBACK 関数を呼び出す必要はありません。
- 接続を自動コミット・モードに設定するには、SETCONNATTR 関数を呼び出して CONNATTR-AUTO-COMMIT 属性を TRUE に設定します。

SQL ステートメントの実行

SQL ステートメントを実行するには、まず、*SQL* ステートメントの準備 を行う必要があります。

- 通常、この準備中に、データベースは SQL ステートメントをプリコンパイルし、そのステートメントのアクセス・プラン を作成します。
- このアクセス・プランは、ステートメントが存在する限り保持されます。
- これで、ステートメントを必要に応じて何度でも実行できます。

PREPARESTATEMENT 関数は、SQL ステートメントの実行準備を行います。これは、ステートメントを表すステートメント・ハンドル を割り振ります。

- アプリケーションは、ステートメントを必要とする後続のすべての関数にステートメント・ハンドルを受け渡す必要があります。
- アプリケーションは一度に複数のステートメントを準備できます。
- アプリケーションは、必要なステートメント・ハンドルを後続の関数で確実に使用する必要があります。

PREPARECALL 関数は、ストアード・プロシージャー呼び出しステートメントの実行準備を行います。

SQL ステートメントには、実行時に評価されるパラメーター が含まれることがあります。

- SQL ステートメント内では、パラメーターは疑問符 (?) でマーク付けされます。
- パラメーターには、出現する順番で 1 から番号が付けられます。
- 準備が完了すると、アプリケーションは BINDPARAMETER 関数を使用してホスト変数をパラメーターにバインドできます。後でステートメントが実行されると、ホスト変数の内容が使用されてデータベースに送信されます。

次の SELECT 照会の例には 1 つのパラメーターが含まれます。

```
SELECT * FROM EMPLOYEE WHERE SALARY>?
```

ステートメントのパラメーターに関する詳細情報を取得するには、GETNUMPARAMETERS および GETPARAMETERINFO 関数を使用できます。

ステートメントを実行するには、EXECUTE 関数を呼び出す必要があります。

- ステートメントが更新 SQL ステートメントであった場合は、GETUPDATECOUNT 関数を使用するか、EXECUTE 関数の UPDATE-COUNT パラメーターを使用して、更新された行数を取得できます。
- ステートメントが SQL 照会であった場合は、カーソルを使用して、結果の行と列を取得する (取り出す) ことができます。
- 1 つのステートメントで複数の結果を提供することができます。
  - 追加の結果を取得するには、GETMORERESULTS 関数を呼び出す必要があります。
  - GETMORERESULTS 関数は、次に使用可能なカーソルまたは更新カウントに移動します。
  - ステートメントがストアード・プロシージャー呼び出しであった場合、結果を取得できなくなるまで GETMORERESULTS 関数を呼び出す必要があります。これによってのみ、ストアード・プロシージャーによって返されたデータで出力パラメーターが更新されます。

複数の属性の設定および取得をステートメント・レベルで行うことができます。これを行うには、SETSTMTATTR または GETSTMTATTR 関数を呼び出します。

ステートメントが不要になった場合は、CLOSESTATEMENT 関数を呼び出してステートメントを閉じる必要があります。

- CLOSESTATEMENT 関数は、ステートメント・ハンドルを解放するとともに、ステートメントの最後の実行から依然として開かれている可能性があるすべてのカーソル (ある場合) を閉じます。
- CLOSESTATEMENT 関数を実行すると、ステートメント・ハンドルは有効でなくなります。

カーソル

SQL 照会を実行すると、結果はカーソル という形式で返されます。

- カーソルにより、結果の行および列を取得する (取り出す) ことができます。
- GETNUMCOLUMNS 関数および GETCOLUMNINFO 関数を使用して、カーソルの列に関する詳細情報を取得できます。
- 列は、表示の順序で、1 から始まる番号で番号付けされます。

カーソルを使用して結果行を取り出すには、まずホスト変数を関心対象の列にバインド する必要があります。

- ホスト変数を関心対象の列にバインドするには、BINDCOLUMN 関数を呼び出します。
- FETCH 関数が後で呼び出されると、ホスト変数は、取り出された行内の列の内容で更新されます。

- デフォルトでは、FETCH 関数は最初から最後までカーソルを処理します。

以下の場合は、カーソルを位置変更 できます。

- データベースによりサポートされている。
- 適切なタイプ (CURSOR-TYPE-SCROLL-INSENSITIVE または CURSOR-TYPE-SCROLL-SENSITIVE) を使用してステートメントを作成済みである。

位置変更は、以下のいずれかを使用して実行できます。

- FETCH-PREVIOUS、FETCH-FIRST、FETCH-LAST、FETCH-ABSOLUTE、または FETCH-RELATIVE 操作が指定された FETCH 関数。
- SETPOS 関数。

以下の場合は、カーソル内 で行を更新、削除、または挿入することもできます。

- データベースによりサポートされている。
- 適切なタイプ (CURSOR-TYPE-SCROLL-INSENSITIVE または CURSOR-TYPE-SCROLL-SENSITIVE) および並行性 (CURSOR-CONCUR-UPDATABLE) を使用してステートメントを作成済みである。

更新、削除、または挿入は、SETPOS-UPDATE、SETPOS-DELETE、または SETPOS-INSERT 操作を使用する SETPOS 関数を呼び出すことによって実行されます。

カーソルを必要としない場合、CLOSECURSOR 関数を呼び出してクローズする必要があります。 CLOSECURSOR 関数の後に、行を取得するために FETCH 関数を使用することはできなくなりました。

データベース・メタデータ

DBCLI インターフェースは、データベースからメタデータ を取得できます。これには、表のリスト、索引、キー、表の列などを取得する関数が含まれます。この情報は一般に、データベースのシステム・カタログ表に保管されます。通常の SELECT ステートメントをシステム・カタログ表に対して実行することもできます。しかしこれには、使用中のデータベース・システムとベンダーについてユーザーが把握していることが必要です。システム・カタログ表は、ベンダー固有であり、データベース固有でもあります。

DBCLI インターフェースは、メタデータ情報を取得するための、一連のデータベース非依存関数を提供します。これらの関数には、接頭部「DB」が付きます。例えば、関数 DBTABLES は、データベースで使用できる表のリストを再試行します。

データベースによっては、一部のメタデータ関数をサポートしない場合があることに注意してください。

接続プーリング

接続プーリング では、CICS Transaction Server for z/VSE V1.1 下で実行されている DBCLI アプリケーション用の既存の データベース接続が保持および再利用されます。これを実装するには、 91 ページの『接続プーリング・マネージャーの構成と開始/停止』で説明されているように接続プーリング・マネージャー を構成します。

デフォルトでは、接続プーリングは使用されません。したがって、既存の DBCLI アプリケーションは引き続きそのままの状態で動作します (接続プーリングを使用しません)。

接続プーリングは、*CICS* で のみ使用できます。バッチ・アプリケーションで接続プーリングを使用することはできません。バッチでの実行中に接続プーリングの使用を試みると、拒否されます。

z/VSE と DBCLIServer 間の接続には、DBCLI アプリケーションで SSL/TLS (Secure Socket Layer/Transport Layer Security) を使用できます。ただし、

- SSL/TLS 接続のプーリングはサポートされません。
- SSL/TLS 接続を使用して接続プーリングを使用する試みは拒否されます。

DBCLI アプリケーションで接続プーリングの使用を要求するには、CONNECT 関数を呼び出す前に、環境変数 ENVATTR-USE-CONNPOOL を TRUE に設定します。この場合、

- これ以上のアプリケーションの変更は不要です。
- 接続プーリングの使用は DBCLI アプリケーションに認識されません。

アプリケーションによって接続プーリングが有効にされた場合、以下の処理が行われます。

1. CONNECT 関数によって、接続プーリング内に一致する 接続 (つまり、同じホスト名/IP アドレス、ポート、データベース名、ユーザー ID、およびパスワードを持つ接続) があるかどうかが確認されます。
   a. 一致する接続がある 場合、その一致する接続がプールから取得され、再利用されます。
   b. その接続がアクティブでなくなっている場合や、一致する接続をプールから取得できない場合は、新しい接続が確立されます。
2. アプリケーションは接続の使用を終えると、DISCONNECT 関数を呼び出します。DISCONNECT 処理中に、接続は接続プーリングに返されるので、その接続を後で再利用できます。
3. 接続プーリングが使用されているかどうかに関係なく、正常な DISCONNECT 呼び出しによって変更がデータベースにコミットされます。
   a. アプリケーション・プログラムが異常終了した場合や、アプリケーション・プログラムがデータベースから正常に切断されずに 終了した場合は、変更は自動的にロールバックされます。
   b. この場合、接続はプールに返されずに閉じられます。

関連トピック:

- 86 ページの『接続プーリングの概要』
- 91 ページの『接続プーリング・マネージャーの構成と開始/停止』

プログラミングの制限と要件

DBCLI プログラミング・インターフェースには、以下の制限があります。

- DBCLI コードが CICS に対応している。 CICS で実行されている場合、メモリー割り振りは、GETVIS マクロではなく EXEC CICS GETMAIN を使用して実行されます。
- DBCLI API は、ICCF 疑似パーティションで実行されているプログラムでは使用できない。
- これらの DBCLI 呼び出しを実行する際には、ロックを保持しない。
- CICS がストレージ保護で作動しているときに CICS トランザクションで DBCLI API を使用する場合は、DBCLI API を使用するすべてのプログラムを EXECKEY(CICS) で定義する必要がある。これは、これらのプログラムにリンクするプログラムにも当てはまります。トランザクション定義用の TASKDATAKEY(CICS) は不要です。
- CICS トランザクションで DBCLI API を使用する場合は、そのトランザクションを実行する前に、EZA の「task-related-user-exit」(TRUE) をアクティブにしておく必要があります。この TRUE をアクティブにする方法について詳しくは、「*IBM z/VSE TCP/IP* サポート」の『EZA インターフェースに関する CICS の考慮事項』を参照してください。
- ほとんどの JDBC ドライバーは、完全な SQL ステートメントしか許容しません。 Db2 Server for VSE アプリケーションで使用される SQL プリプロセッサー・ステートメントは受け入れません。
- 接続プーリングは CICS でのみ使用できます。バッチ・アプリケーションは接続プーリングを使用できません。バッチ実行中に接続プーリングを使用しようとしても拒否されます。
- DBCLI では、z/VSE と DBCLIServer 間の接続に /TLS (Secure Socket Layer/Transport Layer Security) を使用することができます。ただし、SSL/TLS 接続のプーリングはサポートしていません。SSL/TLS 接続を使用して接続プーリングを使用する試みは拒否されます。

DBCLI を COBOL で使用

COBOL プログラム内で DBCLI を使用する場合には、以下のような一般的な考慮事項があります。

すべてのパラメーターは参照による受け渡しを行う必要があります。 COBOL ではこれがデフォルトです。

IESDBCLI に対する呼び出しは静的呼び出しでなければなりません。 NODYNAM コンパイラー・オプションを使用してご使用のプログラムをコンパイルする必要があります。

FUNCTION

16 バイトの文字フィールドであり、左寄せにして、右にブランクが埋められます。呼び出しの名前に設定されます。FUNCTION は大/小文字を特定します。大文字である必要があります。関数は常に最初のパラメーターである必要があります。

ENV-HANDLE
環境ハンドルを含むフルワード・バイナリー変数。ENV-HANDLE パラメーターは、DBCLI への呼び出しごとに、2 番目のパラメーターとして指定する必要があります。

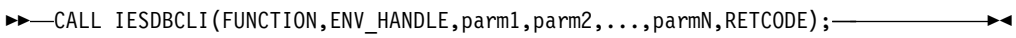
parmn
タイプ呼び出しに応じたパラメーターの変数番号。

RETCODE
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

COBOL コピーブック IESDBCOB には、共通の宣言が含まれています。

DBCLI を PL/I で使用

PL/I プログラム内で DBCLI を使用する場合には、以下のような一般的な考慮事項があります。



すべてのパラメーターは参照による受け渡しを行う必要があります。 PL/I ではこれがデフォルトです。

FUNCTION
16 バイトの文字フィールドであり、左寄せにして、右にブランクが埋められます。呼び出しの名前に設定されます。FUNCTION は大/小文字を特定します。大文字である必要があります。関数は常に最初のパラメーターである必要があります。

ENV_HANDLE
環境ハンドルを含むフルワード・バイナリー変数。ENV-HANDLE パラメーターは、DBCLI への呼び出しごとに、2 番目のパラメーターとして指定する必要があります。

parmn
タイプ呼び出しに応じたパラメーターの変数番号。

RETCODE
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

PL/I コピーブック IESDBPL1 には、共通の宣言が含まれています。以下のステートメントも含まれています。

```
DCL IESDBCLI ENTRY OPTIONS(RETCODE,ASM,INTER) EXT;
```

DBCLI を C で使用

C プログラム内で DBCLI を使用する場合には、以下のような一般的な考慮事項があります。

```
char function[16];
void* env_handle;
int retcode;
```

```
►►—IESDBCLI(function,&env_handle,&parm1,&parm2,...,parmN,&retcode);————►◄
```

すべてのパラメーターは参照による受け渡しを行う必要があります。つまり、パラメーターのアドレスを (& 演算子を使用して) IESDBCLI 関数に明示的に提供する必要があります。 C ストリングでは、& 演算子を使用する必要はありません。C ストリングは文字の配列であり、いずれにしろ参照によって渡されるからです。

DBCLI は、C のゼロ終端 はサポートしていません。

- 入力パラメーターでは、ブランクを埋め込んで、必要な長さの C ストリングを提供する必要があります。
- その後にゼロ終端文字を追加できますが、それは DBCLI に無視されます。
- 出力パラメーターの場合、DBCLI はストリングの終了にゼロ文字を使用しません。
- 必要なら、適切なゼロ終端を確保することはできます。

function
16 バイトの文字フィールドであり、左寄せにして、右にブランクが埋められます。呼び出しの名前に設定されます。FUNCTION は大/小文字を特定します。大文字である必要があります。関数は常に最初のパラメーターである必要があります。

env_handle
環境ハンドルを含むフルワード・バイナリー変数。ENV-HANDLE パラメーターは、DBCLI への呼び出しごとに、2 番目のパラメーターとして指定する必要があります。

parmn
タイプ呼び出しに応じたパラメーターの変数番号。

retcode
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

C ヘッダー・ファイル IESDBC.h には、共通の宣言が含まれます。また、以下のような C 関数プロトタイプと #pragma ステートメントも含まれます。

```
int IESDBCLI(char* function, void** env_handle, ...);
#pragma map(IESDBCLI,"IESDBCLI")
#pragma linkage(IESDBCLI,OS)
```

DBCLI をアセンブラーで使用

アセンブラー・プログラム内で DBCLI を使用する場合には、以下のような一般的な考慮事項があります。

```
►►—CALL IESDBCLI,(FUNCTION,ENV_HANDLE,parm1,parm2,...,parmN,RETCODE),VL————►◄
```

再入可能プログラミングには以下の呼び出しフォーマットを使用できます。

```
►►—CALL IESDBCLI,(FUNCTION,ENV_HANDLE,parm1,parm2,...,parmN,RETCODE),VL,MF=(E,list-addr)————►◄
```

すべてのパラメーターは参照による受け渡しを行う必要があります。つまり、レジスター 1 が指すパラメーター・リストにはパラメーターのアドレスが含まれている必要があります。パラメーター・リストの最後の項目には最上位ビットを入れて、最後のパラメーターであることを示す必要があります。推奨されているとおり VL オプションを使用する場合、この点については CALL マクロによって既に対応されています。

FUNCTION
16 バイトの文字フィールドであり、左寄せにして、右にブランクが埋められます。呼び出しの名前に設定されます。FUNCTION は大/小文字を特定します。大文字である必要があります。関数は常に最初のパラメーターである必要があります。

ENV_HANDLE
環境ハンドルを含むフルワード・バイナリー変数。ENV-HANDLE パラメーターは、DBCLI への呼び出しごとに、2 番目のパラメーターとして指定する必要があります。

parmn
タイプ呼び出しに応じたパラメーターの変数番号。

RETCODE
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

以下のレジスター規則が適用されます。

- レジスター 0、1、14、および 15 はインターフェースによって使用され、必要に応じて、呼び出す前に保存しなければなりません。
- レジスター 13 は、呼び出し元が提供する 72 バイトの保存域を指す必要があります。

アセンブラー・マクロ IESDBASM には、共通の宣言が含まれます。

DBCLI を REXX で使用 (バッチ)

バッチ REXX プログラム内で DBCLI を使用するには、ADDRESS LINKPGM ホスト・コマンド環境を使用して外部プログラム IESDBCLA を呼び出す必要があります。

```
►►—ADDRESS LINKPGM "IESDBCLA FUNCTION ENV_HANDLE parm1 parm2 ... parmN RETCODE"————►◄
```

DBCLI API を呼び出す前に、すべてのパラメーターを適切な長さの値で初期化する必要があります。これは、出力パラメーターには特に必要です。フルワード・バイナリー変数は、4 バイトを含むように初期化する必要があります (例えば、VARIABLE = D2C(0,4))。この変数にはバイナリー表記の値が含まれると想定されているため、値を REXX ストリング表記からバイナリー表記に、あるいはその逆に、変換する必要があります。これには REXX 関数 C2S および D2C を使用します。

FUNCTION
16 バイトの文字フィールドであり、左寄せにして、右にブランクが埋められま

す。呼び出しの名前に設定されます。FUNCTION は大/小文字を特定します。大文字である必要があります。関数は常に最初のパラメーターである必要があります。

ENV_HANDLE
環境ハンドルを含むフルワード・バイナリー変数。ENV-HANDLE パラメーターは、DBCLI への呼び出しごとに、2 番目のパラメーターとして指定する必要があります。

parmn
タイプ呼び出しに応じたパラメーターの変数番号。

RETCODE
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

DBCLI を REXX/CICS で使用

DBCLI を CICS 下で実行される REXX/CICS プログラム内で使用するには、REXX/CICS ユーザーが定義可能なコマンドを使用します。ユーザー定義可能なコマンドについて詳しくは、「*CICS Transaction Server for VSE/ESA, REXX Guide*」(SC34-5764) のトピック『REXX/CICS Command Definition』を参照してください。 DBCLI を使用する REXX/CICS プログラムは、次のようにプログラムの始めに DBCLI コマンドを定義する必要があります。

```
'DEFCMD DBCLI CALL = CALL IESDBCIR (CICSLOAD'
```

これにより、コマンド名が「CALL」の、外部および内部のターゲット環境「DBCLI」が定義されます。このコマンドは、DBCLI REXX/CICS サポート・ルーチン IESDBCIR をロードします。 (CICSLOAD を指定すると、REXX/CICS は EXEC CICS LOAD によって IESDBCIR をロードし、それを呼び出すために分岐リンク・インターフェースを使用します。あるいは、(CICSLINK を指定する方法もあります。これは、EXEC CICS LINK を使用して IESDBCIR を呼び出します。両方のオプションがサポートされています。分岐リンク・インターフェースはオーバーヘッドが少ない一方、LINK インターフェースは (CEDF などによる) トレースが容易です。

DBCLI コマンドを定義すると、次のように DBCLI 呼び出しを実行できます。

```
►►—'DBCLI CALL FUNCTION ENV_HANDLE parm1 parm2 ... parmN RETCODE'————————►◄
```

DBCLI API を呼び出す前に、すべてのパラメーターを適切な長さの値で初期化する必要があります。これは、出力パラメーターには特に必要です。フルワード・バイナリー変数は、4 バイトを含むように初期化する必要があります (例えば、VARIABLE = D2C(0,4))。この変数にはバイナリー表記の値が含まれると想定されているため、値を REXX ストリング表記からバイナリー表記に、あるいはその逆に、変換する必要があります。これには REXX 関数 C2S および D2C を使用します。

FUNCTION
16 バイトの文字フィールドであり、左寄せにして、右にブランクが埋められま

す。呼び出しの名前に設定されます。FUNCTION は大/小文字を特定します。大文字である必要があります。関数は常に最初のパラメーターである必要があります。

ENV_HANDLE
環境ハンドルを含むフルワード・バイナリー変数。ENV-HANDLE パラメーターは、DBCLI への呼び出しごとに、2 番目のパラメーターとして指定する必要があります。

parmn
タイプ呼び出しに応じたパラメーターの変数番号。

RETCODE
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、最後のパラメーターでなければなりません。

DBCLI 関数呼び出しの構文とパラメーター

DBCLI が提供するプログラミング・インターフェースでは、ご使用のアプリケーションが、COBOL、PL/1、C、アセンブラー、または REXX で作成されたバッチまたは CICS TS アプリケーションから DBCLI クライアントを呼び出すことができます。

このトピックのパラメーター記述は COBOL for VSE 言語の構文および規則に従って書かれていますが、お客様はご自身の使用する言語に適した構文や規則に従ってください。

以下は、COBOL、PL/I、C、およびアセンブラー言語プログラムでの、ストレージ定義ステートメントの例です。

```
COBOL PIC
  PIC S9(4) BINARY                                HALFWORD BINARY VALUE
  PIC S9(8) BINARY                                FULLWORD BINARY VALUE
  PIC     X(n)                                    CHARACTER FIELD OF N BYTES
PL/I DECLARE STATEMENT
  DCL     HALF         FIXED BIN(15),             HALFWORD BINARY VALUE
  DCL     FULL         FIXED BIN(31),             FULLWORD BINARY VALUE
  DCL     CHARACTER    CHAR(n)                    CHARACTER FIELD OF n BYTES
C DECLARATION
  short   half;                                   HALFWORD BINARY VALUE
  int     full;                                   FULLWORD BINARY VALUE
  char    string[n];                              CHARACTER FIELD OF n BYTES
ASSEMBLER DECLARATION
  DS      H                                       HALFWORD BINARY VALUE
  DS      F                                       FULLWORD BINARY VALUE
  DS      CLn                                     CHARACTER FIELD OF n BYTES
```

一部の関数ではオプション・パラメーターが使用されます。オプション・パラメーターは、次の例のように大括弧で表示されます。

```
►►—CALL 'IESDBCLI' USING ENV-HANDLE FUNCTION—parm1 [optparm2] ... [optparmN]———►

►—RETCODE.———————————————————————————————————————————————►◄
```

DBCLI 関数 (参照情報)

DBCLI 関数使用箇所のロードマップ

表 8 は、必要な関数を探す際に役立つ「ロードマップ」です。

表 *8. DBCLI* 関数使用箇所のロードマップ

カテゴリー	関数	
環境関数	424 ページの『INITENV』	413 ページの『GETENVATTR』
	436 ページの『TERMENV』	416 ページの『GETLASTERROR』
	432 ページの『SETENVATTR』	425 ページの『INITSSL』
接続関数	343 ページの『CONNECT』	342 ページの『COMMIT』
	344 ページの『CONNECTSSL』	430 ページの『ROLLBACK』
	392 ページの『DISCONNECT』	434 ページの『SETSAVEPOINT』
	431 ページの『SETCONNATTR』	430 ページの『RELEASESAVEPOINT』
	396 ページの『GETCONNATTR』	
ステートメント関数	428 ページの『PREPARESTATEMENT』	435 ページの『SETSTMTATTR』
	426 ページの『PREPARECALL』	421 ページの『GETSTMTATTR』
	341 ページの『CLOSESTATEMENT』	347 ページの『CREATESTATEMENT』
ステートメント・パラメーター関数	418 ページの『GETNUMPARAMETERS』	338 ページの『BINDPARAMETER』
	419 ページの『GETPARAMETERINFO』	
ステートメント実行関数	392 ページの『EXECUTE』	423 ページの『GETUPDATECOUNT』
カーソル関数	417 ページの『GETNUMCOLUMNS』	393 ページの『FETCH』
	395 ページの『GETCOLUMNINFO』	433 ページの『SETPOS』
	334 ページの『BINDCOLUMN』	417 ページの『GETMORERESULTS』
	420 ページの『GETROWNUMBER』	341 ページの『CLOSECURSOR』

表 8. *DBCLI* 関数使用箇所のロードマップ (続き)

カテゴリー	関数	
データベース・メタデータ関数	349 ページの『DBATTRIBUTES』	374 ページの『DBPROCEDURES』
	352 ページの『DBBESTROWIDENT』	376 ページの『DBSCHEMAS』
	354 ページの『DBCATALOGS』	377 ページの『DBSUPERTABLES』
	355 ページの『DBCOLUMNPRIV』	379 ページの『DBSUPERTYPES』
	357 ページの『DBCOLUMNS』	381 ページの『DBTABLEPRIV』
	359 ページの『DBCROSSREFERENCE』	383 ページの『DBTABLES』
	363 ページの『DBEXPORTEDKEYS』	385 ページの『DBTABLETYPES』
	365 ページの『DBIMPORTEDKEYS』	386 ページの『DBTYPEINFO』
	368 ページの『DBINDEXINFO』	388 ページの『DBUDTS』
	370 ページの『DBPRIMARYKEYS』	390 ページの『DBVERSIONCOLS』
	372 ページの『DBPROCEDURECOLS』	

BINDCOLUMN

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

BINDCOLUMN 関数により、アプリケーションはホスト変数を、ステートメントが実行された後にカーソルに入る列にバインドできます。 FETCH 関数が後で呼び出されると、ホスト変数が列の値に設定されます。

関連する関数:

- 395 ページの『GETCOLUMNINFO』
- 417 ページの『GETNUMCOLUMNS』

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE           PIC S9(8) BINARY.
   01 STMT-HANDLE          PIC S9(8) BINARY.
   01 COL-INDEX            PIC S9(8) BINARY.
   01 NATIVE-TYPE          PIC S9(8) BINARY.
   01 VAR                  PIC .....
   01 VAR-LEN              PIC S9(8) BINARY.
   01 VAR-IND              PIC S9(8) BINARY.
   01 OPTION               PIC ....
   01 OPT-LEN              PIC S9(8) BINARY.
   01 RETCODE              PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-BINDCOLUMN ENV-HANDLE
        STMT-HANDLE COL-INDEX NATIVE-TYPE VAR VAR-LEN VAR-IND
        [OPTION [OPT-LEN]] RETCODE.
```

FUNCTION (入力)

BINDCOLUMN を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (入力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。

COL-INDEX (入力)
バインドする列の数を含むフルワード・バイナリー変数。列番号は 1 で始まります。列番号 0 は、現在行を含む特殊列です。使用する場合は、ネイティブ・タイプが NATIVE-TYPE-INTEGER-UNSIGNED のフルワード・バイナリー変数にバインドする必要があります。

NATIVE-TYPE (入力)
列にバインドされるホスト変数のネイティブ・タイプを含むフルワード・バイナリー変数。ネイティブ・タイプの値が 0 の場合、列はアンバインドされます。以下のネイティブ・タイプを使用できます。

NATIVE-TYPE-STRING (1)
テキスト・ストリング。

NATIVE-TYPE-INTEGER-SIGNED (2)
符号付き 2 進整数。

NATIVE-TYPE-INTEGER-UNSIGNED (3)
符号なし 2 進整数。

NATIVE-TYPE-PACKED-SIGNED (4)
符号付きパック 10 進数 (COMP-3)。

NATIVE-TYPE-PACKED-UNSIGNED (5)
符号なしパック 10 進数。

NATIVE-TYPE-ZONED-SIGNED (6)
符号付きゾーン 10 進数。

NATIVE-TYPE-ZONED-UNSIGNED (7)
符号なしゾーン 10 進数。

NATIVE-TYPE-C-STRING (8)
ゼロ終了ストリング (C 言語ストリング)。

NATIVE-TYPE-FLOAT_BFP (9)
IEEE 2 進数浮動小数点表記 (BFP) の浮動小数点数。

NATIVE-TYPE-FLOAT_HFP (10)
16 進数浮動小数点表記 (HFP) の浮動小数点数。

NATIVE-TYPE-TOD (11)
8 バイトの時刻クロック (TOD) 値。

NATIVE-TYPE-BOOLEAN (12)
ブール値。つまり TRUE (1) または FALSE (0) であるビット。

NATIVE-TYPE-BINARY (13)
2 進数フィールド。

NATIVE-TYPE-DATE_PACKED (32)
パック 10 進数フォーマットの日付値。パターン・オプションを使用して、日付パターンを指定します。

NATIVE-TYPE-TIME_PACKED (33)
パック 10 進数フォーマットの時刻値。パターン・オプションを使用して、時間パターンを指定します。

NATIVE-TYPE-TIMESTAMP_PACKED (34)
パック 10 進数フォーマットの日時値。パターン・オプションを使用して、日時パターンを指定します。

NATIVE-TYPE-DATE_STRING (35)
テキスト・フォーマットの日付値。パターン・オプションを使用して、日付パターンを指定します。

NATIVE-TYPE-TIME_STRING (36)
テキスト・フォーマットの時刻値。パターン・オプションを使用して、時間パターンを指定します。

NATIVE-TYPE-TIMESTAMP_STRING (37)
テキスト・フォーマットの日時値。パターン・オプションを使用して、日時パターンを指定します。

VAR (据え置き入出力)
列にバインドされるホスト変数。 FETCH 関数が後で呼び出されると、ホスト変数が列の値に設定されます。パラメーター VAR-LEN は、ホスト変数の長さを指定します (バイト単位)。 COBOL の用語 'LENGTH OF' を使用して変数のバイト長さを決定します。

REXX を使用する場合、このパラメーターはブランクを埋め込んだ 32 文字のテキスト・ストリングとしてバインド変数名を含む変数であることが必要です。この目的で、REXX 関数 LEFT を以下のように使用できます。

```
LEFT('variable-name',32)
```

VAR-LEN (入力)
ホスト変数の長さ (バイト単位) を含むフルワード・バイナリー変数。 COBOL の用語 'LENGTH OF' を使用して変数のバイト長さを決定します。

VAR-IND (据え置き入出力)
列にバインドされたホスト変数の標識として機能するフルワード・バイナリー変数。 FETCH 関数が後で呼び出されると、標識変数が設定されます。その内容は、変数値が NULL であるかどうか、または行が更新、削除、または挿入されているかどうかを示します。更新可能カーソルの場合、標識変数はさらに、SETPOS-UPDATE 操作が指定された SETPOS 関数の使用時に、どの列を更新するかを指定するためにも使用されます。

REXX を使用する場合、このパラメーターはブランクを埋め込んだ 32 文字のテキスト・ストリングとしてバインド変数名を含む変数であることが必要です。この目的で、REXX 関数 LEFT を以下のように使用できます。

```
LEFT('variable-name',32)
```

標識変数には、以下の値を含めることができます。

INDICATE-NOTNULL (0)
変数値は非 NULL です。列にバインドされたホスト変数の内容が設定されます。

INDICATE-NULL (1)
変数値は NULL です。

INDICATE-IGNOREUPDATE (2)
SETPOS-UPDATE 操作が指定された SETPOS 関数を使用して行の更新を実行する際には、この列は無視してください。

INDICATE-DELETED (16)
現在行が削除されたことを示します。

INDICATE-INSERTED (32)
現在行が挿入されたことを示します。この値は INDICATE-NULL と結合できます。

INDICATE-UPDATED (64)
現在行が更新されたことを示します。この値は INDICATE-NULL と結合できます。

OPTION (入力、オプション)
ホスト変数のオプション値を指定するために使用される変数。このパラメーターはオプションです。以下のネイティブ・タイプは、オプション値を受け入れます。

NATIVE-TYPE-STRING および NATIVE-TYPE-C-STRING
オプション値は、ホスト変数の内容を EBCDIC から ASCII に (またはその逆に) 変換するために使用する EBCDIC コード・ページを指定するテキスト・フィールドです。 OPT-LEN パラメーターを指定する必要があり、テキスト・フィールドの長さを含む必要があります。

すべての数値タイプ (PACKED、ZONED、INTEGER)
オプション値は、数値データに使用される暗黙の小数点位を含むフルワード・バイナリー変数です。 OPT-LEN パラメーターはこのオプションでは必須ではありません。

NATIVE-TYPE-BOOLEAN
オプション値は、右からカウントされたビット位置を含むフルワード・バイナリー変数です。デフォルトはビット・ゼロ (右端ビット) です。
OPT-LEN パラメーターはこのオプションでは必須ではありません。

すべての DATE、TIME、および TIMESAMP タイプ
オプション値は、ホスト変数の内容をデータベース DATE、TIME、または TIMESTAMP に (またはその逆に) 変換するために使用するパターンを指定するテキスト・フィールドです。サンプルのパターン・フォーマット「yyyy.MM.dd hh:mm:ss」は、「1996.07.10 15:08:56」などの入力を受け入れます。 OPT-LEN パラメーターを指定する必要があり、テキスト・フィールドの長さを含む必要があります。

OPT-LEN (入力、オプション)
OPTION がテキスト・フィールドである場合、OPTION パラメーターの長さを含むフルワード・バイナリー変数。このパラメーターはオプションです。これが指定されている場合、OPTION パラメーターも指定する必要があります。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

BINDPARAMETER

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

BINDPARAMETER 関数により、アプリケーションはホスト変数をステートメント・パラメーターにバインドできます。ステートメントを後で実行する場合、パラメーターにバインドされたホスト変数の内容は、データベースに送信されます。

関連セクション:

- 418 ページの『GETNUMPARAMETERS』
- 419 ページの『GETPARAMETERINFO』

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE          PIC S9(8) BINARY.
   01 STMT-HANDLE         PIC S9(8) BINARY.
   01 PARAM-INDEX         PIC S9(8) BINARY.
   01 NATIVE-TYPE         PIC S9(8) BINARY.
   01 VAR                 PIC .....
   01 VAR-LEN             PIC S9(8) BINARY.
   01 VAR-IND             PIC S9(8) BINARY.
   01 OPTION              PIC ....
   01 OPT-LEN             PIC S9(8) BINARY.
   01 RETCODE             PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-BINDPARAMETER ENV-HANDLE
        STMT-HANDLE PARM-INDEX NATIVE-TYPE VAR VAR-LEN VAR-IND
        [OPTION [OPT-LEN]] RETCODE.
```

FUNCTION (入力)
BINDPARAMETER を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (入力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。

PARAM-INDEX (入力)
バインドするステートメント・パラメーターの番号を含むフルワード・バイナリー変数。パラメーター番号は、1 から始まります。

NATIVE-TYPE (入力)
ステートメント・パラメーターにバインドされるホスト変数のネイティブ・タイプを含むフルワード・バイナリー変数。ネイティブ・タイプの値が 0 の場合、パラメーターはアンバインドされます。以下のネイティブ・タイプを使用できます。

NATIVE-TYPE-STRING (1)
テキスト・ストリング。

NATIVE-TYPE-INTEGER-SIGNED (2)
符号付き 2 進整数。

NATIVE-TYPE-INTEGER-UNSIGNED (3)
符号なし 2 進整数。

NATIVE-TYPE-PACKED-SIGNED (4)
符号付きパック 10 進数 (COMP-3)。

NATIVE-TYPE-PACKED-UNSIGNED (5)
符号なしパック 10 進数。

NATIVE-TYPE-ZONED-SIGNED (6)
符号付きゾーン 10 進数。

NATIVE-TYPE-ZONED-UNSIGNED (7)
符号なしゾーン 10 進数。

NATIVE-TYPE-C-STRING (8)
ゼロ終了ストリング (C 言語ストリング)。

NATIVE-TYPE-FLOAT_BFP (9)
IEEE 2 進数浮動小数点表記 (BFP) の浮動小数点数。

NATIVE-TYPE-FLOAT_HFP (10)
16 進数浮動小数点表記 (HFP) の浮動小数点数。

NATIVE-TYPE-TOD (11)
8 バイトの時刻クロック (TOD) 値。

NATIVE-TYPE-BOOLEAN (12)
ブール値。つまり TRUE (1) または FALSE (0) であるビット。

NATIVE-TYPE-BINARY (13)
2 進数フィールド。

NATIVE-TYPE-DATE_PACKED (32)
パック 10 進数フォーマットの日付値。パターン・オプションを使用して、日付パターンを指定します。

NATIVE-TYPE-TIME_PACKED (33)
パック 10 進数フォーマットの時刻値。パターン・オプションを使用して、時間パターンを指定します。

NATIVE-TYPE-TIMESTAMP_PACKED (34)
パック 10 進数フォーマットの日時値。パターン・オプションを使用して、日時パターンを指定します。

NATIVE-TYPE-DATE_STRING (35)
テキスト・フォーマットの日付値。パターン・オプションを使用して、日付パターンを指定します。

NATIVE-TYPE-TIME_STRING (36)
テキスト・フォーマットの時刻値。パターン・オプションを使用して、時間パターンを指定します。

NATIVE-TYPE-TIMESTAMP_STRING (37)
テキスト・フォーマットの日時値。パターン・オプションを使用して、日時パターンを指定します。

VAR (据え置き入出力)
ステートメント・パラメーターにバインドされるホスト変数。ステートメントを実行する場合、ホスト変数の内容は、データベースに送信されます。ステートメント・パラメーターが出力パラメーターを示す場合、ストアード・プロシージャ

ーの実行によりホスト変数を設定することもできます。パラメーター VAR-LEN は、ホスト変数の長さを指定します (バイト単位)。 COBOL の用語 'LENGTH OF' を使用して変数のバイト長さを決定します。

REXX を使用する場合、このパラメーターはブランクを埋め込んだ 32 文字のテキスト・ストリングとしてバインド変数名を含む変数であることが必要です。この目的で、REXX 関数 LEFT を以下のように使用できます。

```
LEFT('variable-name',32)
```

VAR-LEN (入力)
ホスト変数の長さ (バイト単位) を含むフルワード・バイナリー変数。 COBOL の用語 'LENGTH OF' を使用して変数のバイト長さを決定します。

VAR-IND (据え置き入出力)
ステートメント・パラメーターにバインドされたホスト変数の標識として機能するフルワード・バイナリー変数。標識変数の内容は、ステートメントの実行時に決定されます。これは、変数値が NULL であるかどうかを示します。パラメーターが出力パラメーターである場合、ストアード・プロシージャーが実行されると標識変数が更新されることがあります。

REXX を使用する場合、このパラメーターはブランクを埋め込んだ 32 文字のテキスト・ストリングとしてバインド変数名を含む変数であることが必要です。この目的で、REXX 関数 LEFT を以下のように使用できます。

```
LEFT('variable-name',32)
```

標識変数には、以下の値を含めることができます。

INDICATE-NOTNULL (0)
変数値は非 NULL です。パラメーターにバインドされたホスト変数の内容が使用されます。

INDICATE-NULL (1)
変数値は NULL です。

OPTION (入力、オプション)
ホスト変数のオプション値を指定するために使用される変数。このパラメーターはオプションです。以下のネイティブ・タイプは、オプション値を受け入れます。

NATIVE-TYPE-STRING および NATIVE-TYPE-C-STRING
オプション値は、ホスト変数の内容を EBCDIC から ASCII に (またはその逆に) 変換するために使用する EBCDIC コード・ページを指定するテキスト・フィールドです。 OPT-LEN パラメーターを指定する必要があり、テキスト・フィールドの長さを含む必要があります。

すべての数値タイプ (PACKED、ZONED、INTEGER)
オプション値は、数値データに使用される暗黙の小数点位を含むフルワード・バイナリー変数です。 OPT-LEN パラメーターはこのオプションでは必須ではありません。

NATIVE-TYPE-BOOLEAN
オプション値は、右からカウントされたビット位置を含むフルワード・バイナリー変数です。デフォルトはビット・ゼロ (右端ビット) です。
OPT-LEN パラメーターはこのオプションでは必須ではありません。

すべての **DATE**、**TIME**、および **TIMESAMP** タイプ
オプション値は、ホスト変数の内容をデータベース DATE、TIME、または TIMESTAMP に (またはその逆に) 変換するために使用するパターンを指定するテキスト・フィールドです。サンプルのパターン・フォーマット「yyyy.MM.dd hh:mm:ss」は、「1996.07.10 15:08:56」などの入力を受け入れます。 OPT-LEN パラメーターを指定する必要があり、テキスト・フィールドの長さを含む必要があります。

OPT-LEN (入力、オプション)
OPTION がテキスト・フィールドである場合、OPTION パラメーターの長さを含むフルワード・バイナリー変数。このパラメーターはオプションです。これが指定されている場合、OPTION パラメーターも指定する必要があります。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

CLOSECURSOR

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

CLOSECURSOR 関数は、オープン・カーソルを閉じます。カーソルが閉じられると、FETCH 関数を使用してカーソルからデータを取り出すことはできなくなります。クローズ・カーソルはまた、プリフェッチ・バッファーに割り振られたメモリーを解放します。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE            PIC S9(8) BINARY.
   01 STMT-HANDLE           PIC S9(8) BINARY.
   01 RETCODE               PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-CLOSECURSOR ENV-HANDLE
        STMT-HANDLE RETCODE.
```

FUNCTION (入力)
CLOSECURSOR を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (入力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

CLOSESTATEMENT

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

CLOSESTATEMENT 関数は、指定されたステートメントを閉じます。 CLOSESTATEMENT 関数を正常に呼び出した後は、ステートメント・ハンドルは無効になります。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE         PIC S9(8) BINARY.
   01 STMT-HANDLE        PIC S9(8) BINARY.
   01 RETCODE            PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-CLOSESTATEMENT ENV-HANDLE
        STMT-HANDLE RETCODE.
```

FUNCTION (入力)
CLOSESTATEMENT を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (入力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

COMMIT

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

COMMIT 関数は、作業論理単位を終了させ、この接続でその時点まで、または前回のコミット以降にデータベース内で実行されたすべての変更をコミットします。その保持能力によっては、コミット呼び出しがステートメントのカーソルを閉じることがあります (ステートメントの準備時の HOLD-CURSORS-OVER-COMMIT または CLOSE-CURSORS-AT-COMMIT 値を参照してください)。

関連する関数: 430 ページの『ROLLBACK』

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE         PIC S9(8) BINARY.
   01 CON-HANDLE         PIC S9(8) BINARY.
   01 RETCODE            PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-COMMIT ENV-HANDLE CON-HANDLE
        RETCODE.
```

FUNCTION (入力)
COMMIT を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

CON-HANDLE (入力)
接続ハンドルを含むフルワード・バイナリー変数。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

CONNECT

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

CONNECT 関数は、DBCliServer およびデータベースへの接続を確立します。これは接続を表す接続ハンドルを割り振り、CON-HANDLE パラメーターを設定します。アプリケーションは、接続を必要とする後続のすべての関数に接続ハンドルを受け渡す必要があります。アプリケーションは一度に複数の接続を確立できます。アプリケーションは、必要な接続ハンドルを後続の関数で確実に使用する必要があります。

関連する関数:

- 344 ページの『CONNECTSSL』
- 392 ページの『DISCONNECT』

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE           PIC S9(8) BINARY.
   01 CON-HANDLE           PIC S9(8) BINARY.
   01 SERVER               PIC X(nnn) VALUE IS 'myhost.domain.com'.
   01 SERVER-LEN           PIC S9(8) BINARY VALUE IS 17.
   01 PORT                 PIC S9(8) BINARY VALUE IS 16178.
   01 DBNAME               PIC X(nnn) VALUE IS 'SAMPLE'.
   01 DBNAME-LEN           PIC S9(8) BINARY VALUE IS 6.
   01 USERID               PIC X(nnn) VALUE IS 'DBUSER'.
   01 USERID-LEN           PIC S9(8) BINARY VALUE IS 6.
   01 PASSWD               PIC X(nnn) VALUE IS 'password'.
   01 PASSWD-LEN           PIC S9(8) BINARY VALUE IS 8.
   01 RETCODE              PIC S9(8) BINARY.
PROCEDURE
    CALL 'IESDBCLI' USING FUNC-CONNECT ENV-HANDLE CON-HANDLE
         SERVER SERVER-LEN PORT
         DBNAME DBNAME-LEN [USERID USERID-LEN PASSWD PASSWD-LEN]
         RETCODE.
```

FUNCTION (入力)
CONNECT を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

CON-HANDLE (出力)
接続ハンドルを含むフルワード・バイナリー変数。CONNECT 呼び出しにより設定された CON-HANDLE 値は、接続を必要とする他の DBCLI 関数への後続の呼び出しで指定される必要があります。

SERVER (入力)
接続先の DBCliServer のホスト名、IPv4 または IPv6 アドレスを含む文字フィールド。このフィールドは SERVER-LEN パラメーターで指定されている長さまで、右側にブランクを埋め込む必要があります。

SERVER-LEN (入力)
SERVER パラメーターで指定されたホスト名、IPv4 または IPv6 アドレスの長さを含む、フルワード・バイナリー変数。

PORT (入力)
DBCliServer に接続するために使用するポート番号を含むフルワード・バイナリ

ー変数。有効なポート番号は 1 から 65535 です。 0 を指定すると、デフォルトのポート番号が使用されます。デフォルトのポート番号は、環境属性 ENVATTR-DEFAULT-PORT を使用して設定できます。

DBNAME (入力)
接続先のデータベース名を含む文字フィールド。データベース名は DBCliServer サイドで構成されたデータベースを指します。データベース名の先頭が jdbc である場合、それは使用する JDBC URL を示します。このフィールドは DBNAME-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。

DBNAME-LEN (入力)
DBNAME パラメーターで指定されたデータベース名の長さを含む、フルワード・バイナリー変数。

USERID (入力、オプション)
データベースへの接続時に使用されるユーザー ID を含む、オプションの文字フィールド。このフィールドは USERID-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。このパラメーターを指定した場合は、USERID-LEN、PASSWD、および PASSWD-LEN の各パラメーターも指定する必要があります。

USERID-LEN (入力、オプション)
USERID パラメーターで指定されたユーザー ID の長さを含む、フルワード・バイナリー変数。

PASSWD (入力、オプション)
データベースへの接続時に使用されるパスワードを含む、オプションの文字フィールド。このフィールドは PASSWD-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。このパラメーターを指定した場合は、USERID、USERID-LEN、および PASSWD-LEN の各パラメーターも指定する必要があります。

PASSWD-LEN (入力、オプション)
PASSWD パラメーターで指定されたパスワードの長さを含む、フルワード・バイナリー変数。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

CONNECTSSL

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

CONNECTSSL 関数は、DBCLI サーバー およびデータベースへの接続を確立します。これは接続を表す接続ハンドルを割り振り、CON-HANDLE パラメーターを設定します。アプリケーションは、接続を必要とする後続のすべての関数に接続ハンドルを受け渡す必要があります。アプリケーションは一度に複数の接続を確立できます。アプリケーションは、必要な接続ハンドルを後続の関数で確実に使用する必要があります。 CONNECTSSL は、SSL または TLS プロトコルを使用して、DBCLI サ

ーバー (DBCliServer) への暗号化接続 を確立します。 SSL を使用する場合、CONNECTSSL 関数を呼び出す前に INITSSL 関数を呼び出す必要があります。

関連する関数:

- 343 ページの『CONNECT』
- 392 ページの『DISCONNECT』

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE           PIC S9(8) BINARY.
   01 CON-HANDLE           PIC S9(8) BINARY.
   01 SERVER               PIC X(nnn) VALUE IS 'myhost.domain.com'.
   01 SERVER-LEN           PIC S9(8) BINARY VALUE IS 17.
   01 PORT                 PIC S9(8) BINARY VALUE IS 16178.
   01 DBNAME               PIC X(nnn) VALUE IS 'SAMPLE'.
   01 DBNAME-LEN           PIC S9(8) BINARY VALUE IS 6.
   01 USERID               PIC X(nnn) VALUE IS 'DBUSER'.
   01 USERID-LEN           PIC S9(8) BINARY VALUE IS 6.
   01 PASSWD               PIC X(nnn) VALUE IS 'password'.
   01 PASSWD-LEN           PIC S9(8) BINARY VALUE IS 8.
   01 KEYNAME              PIC X(nnn) VALUE IS 'SAMPLE'.
   01 KEYNAME-LEN          PIC S9(8) BINARY VALUE IS 6.
   01 CIPHER               PIC X(nnn) VALUE IS '352F'.
   01 CIPHER-LEN           PIC S9(8) BINARY VALUE IS 4.
   01 RETCODE              PIC S9(8) BINARY.
PROCEDURE
    CALL 'IESDBCLI' USING FUNC-CONNECTSSL ENV-HANDLE CON-HANDLE
         SERVER SERVER-LEN PORT
         KEYNAME KEYNAME-LEN CIPHER CIPHER-LEN
         DBNAME DBNAME-LEN [USERID USERID-LEN PASSWD PASSWD-LEN]
         RETCODE.
```

FUNCTION (入力)
CONNECTSSL を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

CON-HANDLE (出力)
接続ハンドルを含むフルワード・バイナリー変数。CONNECTSSL 呼び出しにより設定された CON-HANDLE 値は、接続を必要とする他の DBCLI 関数への後続の呼び出しで指定される必要があります。

SERVER (入力)
接続先の DBCliServer のホスト名、IPv4 または IPv6 アドレスを含む文字フィールド。このフィールドは SERVER-LEN パラメーターで指定されている長さまで、右側にブランクを埋め込む必要があります。

SERVER-LEN (入力)
SERVER パラメーターで指定されたホスト名、IPv4 または IPv6 アドレスの長さを含む、フルワード・バイナリー変数。

PORT (入力)
DBCliServer に接続するために使用するポート番号を含むフルワード・バイナリー変数。有効なポート番号は 1 から 65535 です。 0 を指定すると、デフォルトのポート番号が使用されます。デフォルトのポート番号は、環境属性 ENVATTR-DEFAULT-PORT を使用して設定できます。

DBNAME (入力)
接続先のデータベース名を含む文字フィールド。データベース名は DBCliServer サイドで構成されたデータベースを指します。データベース名の先頭が jdbc である場合、それは使用する JDBC URL を示します。このフィールドは DBNAME-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。

DBNAME-LEN (入力)
DBNAME パラメーターで指定されたデータベース名の長さを含む、フルワード・バイナリー変数。

USERID (入力、オプション)
データベースへの接続時に使用されるユーザー ID を含む、オプションの文字フィールド。このフィールドは USERID-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。このパラメーターを指定した場合は、USERID-LEN、PASSWD、および PASSWD-LEN の各パラメーターも指定する必要があります。

USERID-LEN (入力、オプション)
USERID パラメーターで指定されたユーザー ID の長さを含む、フルワード・バイナリー変数。

PASSWD (入力、オプション)
データベースへの接続時に使用されるパスワードを含む、オプションの文字フィールド。このフィールドは PASSWD-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。このパラメーターを指定した場合は、USERID、USERID-LEN、および PASSWD-LEN の各パラメーターも指定する必要があります。

PASSWD-LEN (入力、オプション)
PASSWD パラメーターで指定されたパスワードの長さを含む、フルワード・バイナリー変数。

KEYNAME (入力)
この接続で使用する SSL/TLS 鍵リング名を含む、文字フィールド。このフィールドは KEYNAME-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。

KEYNAME-LEN (入力)
KEYNAME パラメーターで指定されたキー名の長さを含む、フルワード・バイナリー変数。

CIPHER (入力)
この接続で用いられる使用優先度順に、SSL/TLS 暗号が含まれる文字フィールド。このフィールドは CIPHER-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。TCP/IP for z/VSE および OpenSSL によりサポートされる値は以下のとおりです。

```
C027 for TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 OPENSSL ONLY
C014 for TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA  - OPENSSL ONLY
C013 for TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA  - OPENSSL ONLY
C012 for TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA - OPENSSL ONLY
  6B for TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 - OPENSSL ONLY
  67 for TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 - OPENSSL ONLY
  39 for TLS_DHE_RSA_WITH_AES_256_CBC_SHA    - OPENSSL ONLY
  33 for TLS_DHE_RSA_WITH_AES_128_CBC_SHA    - OPENSSL ONLY
```

```
16 for SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA   - OPENSSL ONLY
3D for TLS_RSA_WITH_AES_256_CBC_SHA256     - OPENSSL ONLY
3C for TLS_RSA_WITH_AES_128_CBC_SHA256     - OPENSSL ONLY
3B for TLS_RSA_WITH_NULL_SHA256 - DEPRECATED,OPENSSL ONLY
35 for TLS_RSA_WITH_AES_256_CBC_SHA
2F for TLS_RSA_WITH_AES_128_CBC_SHA
0A for RSA1024_3DESCBC_SHA                 - DEPRECATED
09 for RSA1024_DESCBC_SHA                  - DEPRECATED
08 for RSA512_DES40CBC_SHA                 - DEPRECATED
02 for RSA512_NULL_SHA                     - DEPRECATED
01 for RSA512_NULL_                        - DEPRECATED
```

これらの値は任意の組み合わせにより、任意の順序で使用できます。

CIPHER-LEN (入力)
CIPHER パラメーターで指定された暗号の長さを含む、フルワード・バイナリー変数。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

CREATESTATEMENT

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

CREATESTATEMENT 関数は、実行用の SQL ステートメントを作成します。このステートメントを表すステートメント・ハンドルを割り振り、STMT-HANDLE パラメーターを設定します。アプリケーションは、ステートメントを必要とする後続のすべての関数にステートメント・ハンドルを受け渡す必要があります。アプリケーションは、一度に複数のステートメントを作成できます。アプリケーションは、必要なステートメント・ハンドルを後続の関数で確実に使用する必要があります。

CREATESTATEMENT で作成されたステートメントは、疑問符 (?) のマークが付いたパラメーターを含むことができません。パラメーターを使用したい場合は、ステートメントを準備するために PREPARESTATEMENT を代わりに使用する必要があります。

関連する関数:

- 341 ページの『CLOSESTATEMENT』
- 426 ページの『PREPARECALL』
- 428 ページの『PREPARESTATEMENT』

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE           PIC S9(8) BINARY.
   01 CON-HANDLE           PIC S9(8) BINARY.
   01 STMT-HANDLE          PIC S9(8) BINARY.
   01 SQL                  PIC X(nnn) VALUE IS 'SELECT * FROM TABLE WHERE A=5'.
   01 SQL-LEN              PIC S9(8) BINARY VALUE IS 30.
   01 CURSOR-TYPE          PIC S9(8) BINARY VALUE IS 1003.
   01 CONCURRENCY          PIC S9(8) BINARY VALUE IS 1007.
   01 HOLDABILITY          PIC S9(8) BINARY VALUE IS 1.
   01 RETCODE              PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-CREATESTATEMENT ENV-HANDLE CON-HANDLE
        STMT-HANDLE SQL SQL-LEN [CURSOR-TYPE CONCURRENCY [HOLDABILITY]]
        RETCODE.
```

FUNCTION (入力)
PREPARESTATEMENT を含む 16 バイトの文字フィールド。このフィールドは左寄せされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

CON-HANDLE (入力)
接続ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (出力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。
CREATESTATEMENT 呼び出しにより設定された STMT-HANDLE 値は、ステートメントを必要とする他の DBCLI 関数への後続の呼び出しで指定される必要があります。

SQL (入力)
準備対象の SQL ステートメントを含む文字フィールド。このフィールドは SQL-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。

SQL-LEN (入力)
SQL パラメーターで指定された SQL ステートメントの長さを含む、フルワード・バイナリー変数。

CURSOR-TYPE (入力、オプション)
カーソル・タイプを含むフルワード・バイナリー変数。このパラメーターはオプションです。これを指定する場合は、CONCURRENCY パラメーターも指定する必要があります。カーソル・タイプは、カーソルが前方スクロールのみか、通常のスクロールに使用できるかを制御します。以下の値が使用できます。

CURSOR-TYPE-FORWARD-ONLY (1003):
カーソルは、前方にのみ移動できます。

CURSOR-TYPE-SCROLL-INSENSITIVE (1004):
カーソルはスクロール可能ですが、一般に、他者による変更は認識しません。

CURSOR-TYPE-SCROLL-SENSITIVE (1005):
カーソルはスクロール可能で、一般に、他者による変更も認識します。

CONCURRENCY (入力、オプション)
並行性モードを含むフルワード・バイナリー変数。このパラメーターはオプションです。これを指定する場合は、CURSOR-TYPE パラメーターも指定する必要があります。並行性モードは、カーソルが読み取り専用か、更新可能かを制御します。以下の値が使用できます。

CURSOR-CONCUR-READ-ONLY (1007):
カーソルは読み取り専用です。

CURSOR-CONCUR-UPDATABLE (1008):
カーソルは更新可能です。

HOLDABILITY (入力、オプション)
保持能力モードを含むフルワード・バイナリー変数。このパラメーターはオプションです。これを指定する場合は、CURSOR-TYPE パラメーターと

CONCURRENCY パラメーターも指定する必要があります。値は以下のいずれかを含むフルワード・バイナリー変数です。

HOLD-CURSORS-OVER-COMMIT (1):
COMMIT が実行されても、オープン・カーソルは開いたままです。

CLOSE-CURSORS-AT-COMMIT (2):
COMMIT が実行されると、すべてのオープン・カーソルが閉じられます。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

DBATTRIBUTES

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

DBATTRIBUTES 関数は、特定のスキーマとカタログで使用可能な、ユーザー定義タイプ (UDT) の特定のタイプの特定の属性の説明を取得します。これはステートメント・ハンドルを割り振り、 STMT-HANDLE パラメーターを設定します。このステートメントは、BINDCOLUMN および FETCH などの関数を使用して戻された情報を検索するのに使用される必要がある、オープン・カーソルを含みます。カーソルは CLOSECURSOR 関数を使用してクローズされなければならず、ステートメントは CLOSESTATEMENT 関数を使用してクローズされなければなりません。

説明は、カタログ、スキーマ、タイプ、および属性名の基準と一致する UDT の属性に対してのみ返されます。それらは TYPE_SCHEM、TYPE_NAME、および ORDINAL_POSITION により配列されます。この説明には、継承された属性は含まれません。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE            PIC S9(8) BINARY.
   01 CON-HANDLE            PIC S9(8) BINARY.
   01 STMT-HANDLE           PIC S9(8) BINARY.
   01 CATALOG               PIC X(nnn).
   01 CATALOG-LEN           PIC S9(8) BINARY.
   01 SCHEMAPATT            PIC X(nnn).
   01 SCHEMAPATT-LEN        PIC S9(8) BINARY.
   01 TYPENAMEPATT          PIC X(nnn).
   01 TYPENAMEPATT-LEN      PIC S9(8) BINARY.
   01 ATTRNAMEPATT          PIC X(nnn).
   01 ATTRNAMEPATT-LEN      PIC S9(8) BINARY.
   01 RETCODE               PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-DBATTRIBUTES ENV-HANDLE CON-HANDLE
        STMT-HANDLE CATALOG CATALOG-LEN SCHEMAPATT SCHEMAPATT-LEN
        TYPENAMEPATT TYPENAMEPATT-LEN ATTRNAMEPATT ATTRNAMEPATT-LEN
        RETCODE.
```

FUNCTION (入力)
DBATTRIBUTES を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

CON-HANDLE (入力)
接続ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (出力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。この関数呼び出しにより設定された STMT-HANDLE 値は、ステートメントを必要とする他の DBCLI 関数への後続の呼び出しで指定される必要があります。

CATALOG (入力)
カタログ名を含む文字フィールド。このフィールドは CATALOG-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。CATALOG がブランクで構成されている場合 (しかし CATALOG-LEN がゼロより大きい場合)、関数はカタログのないものを検索します。
CATALOG-LEN がゼロの場合、カタログ名は検索の絞り込みに使用されません。

CATALOG-LEN (入力)
CATALOG パラメーターで指定されたカタログ名の長さを含む、フルワード・バイナリー変数。

SCHEMAPATT (入力)
スキーマ名パターンを含む文字フィールド。このフィールドは SCHEMAPATT-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。SCHEMAPATT がブランクで構成されている場合 (しかし SCHEMAPATT-LEN がゼロより大きい場合)、関数はスキーマのないものを検索します。 SCHEMAPATT-LEN がゼロの場合、スキーマ名パターンは検索の絞り込みに使用されません。このパターン内では以下のとおりです。

- '%' は 0 文字以上の任意のサブストリングと一致することを意味します。
- '_' は任意の 1 文字と一致することを意味します。

SCHEMAPATT-LEN (入力)
SCHEMAPATT パラメーターで指定されたスキーマ名パターンの長さを含む、フルワード・バイナリー変数。

TYPENAMEPATT (入力)
タイプ名パターンを含む文字フィールド。このフィールドは TYPENAMEPATT-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。TYPENAMEPATT-LEN がゼロの場合、タイプ名パターンは検索の絞り込みに使用されません。このパターン内では以下のとおりです。

- '%' は 0 文字以上の任意のサブストリングと一致することを意味します。
- '_' は任意の 1 文字と一致することを意味します。

TYPENAMEPATT-LEN (入力)
TYPENAMEPATT パラメーターで指定されたタイプ名パターンの長さを含む、フルワード・バイナリー変数。

ATTRNAMEPATT (入力)
属性名パターンを含む文字フィールド。このフィールドは ATTRNAMEPATT-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。ATTRNAMEPATT-LEN がゼロの場合、属性名パターンは検索の絞り込みに使用されません。このパターン内では以下のとおりです。

- '%' は 0 文字以上の任意のサブストリングと一致することを意味します。
- '_' は任意の 1 文字と一致することを意味します。

ATTRNAMEPATT-LEN (入力)
ATTRNAMEPATT パラメーターで指定された属性名パターンの長さを含む、フルワード・バイナリー変数。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

戻されたカーソルは、以下の列を含みます。

1. TYPE_CAT (String): タイプ・カタログ (NULL の場合もあります)
2. TYPE_SCHEM (String): タイプ・スキーマ (NULL の場合もあります)
3. TYPE_NAME (String): タイプ名
4. ATTR_NAME (String): 属性名
5. DATA_TYPE (int): 属性タイプの SQL タイプ
6. ATTR_TYPE_NAME (String): データ・ソース依存のタイプ名。 UDT の場合、タイプ名は完全修飾です。 REF の場合、タイプ名は完全修飾であり、参照タイプのターゲット・タイプを表します。
7. ATTR_SIZE (int): 列のサイズ。文字タイプまたは日付タイプの場合、これは文字の最大数です。数字タイプまたは 10 進数タイプの場合、これは精度です。
8. DECIMAL_DIGITS (int): 小数桁の数
9. NUM_PREC_RADIX (int): 基数 (一般的には 10 または 2)
10. NULLABLE (int): NULL が許可されるかどうか
    - NULLABLE-NONULLS (0): NULL 値を許可しない場合がある
    - NULLABLE-NULLABLE(1): NULL 値を例外なく許可する
    - NULLABLE-UNKNOWN(2): NULL 可能性は不明
11. REMARKS (String): 列を説明するコメント (NULL の場合もあります)
12. ATTR_DEF (String): デフォルト値 (NULL の場合もあります)
13. SQL_DATA_TYPE (int):未使用
14. SQL_DATETIME_SUB (int): 未使用
15. CHAR_OCTET_LENGTH (int): 文字タイプの場合、列内のバイトの最大数
16. ORDINAL_POSITION (int): 表内の列の索引 (1 で始まる)
17. IS_NULLABLE (String):「NO」は、列が NULL 値を例外なく許可しないことを意味します。「YES」は、列が NULL 値を許可する場合があることを意味します。空のストリングは不明を意味します。
18. SCOPE_CATALOG (String): 参照属性の有効範囲である表のカタログ (DATA_TYPE が REF でない場合は NULL)
19. SCOPE_SCHEMA (String): 参照属性の有効範囲である表のスキーマ (DATA_TYPE が REF でない場合は NULL)
20. SCOPE_TABLE (String): 参照属性の有効範囲である表名 (DATA_TYPE が REF でない場合は NULL)

21. SOURCE_DATA_TYPE (short): 特殊タイプまたはユーザー生成 Ref タイプのソース・タイプ、java.sql.Types からの SQL タイプ (DATA_TYPE が DISTINCT またはユーザー生成 REF でない場合は NULL)

DBBESTROWIDENT

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

DBBESTROWIDENT 関数は、行を一意的に識別する、表の最適化された列セットの説明を取得します。それらは SCOPE に従って配列されます。これはステートメント・ハンドルを割り振り、 STMT-HANDLE パラメーターを設定します。このステートメントは、BINDCOLUMN および FETCH などの関数を使用して戻された情報を検索するのに使用される必要がある、オープン・カーソルを含みます。カーソルは CLOSECURSOR 関数を使用してクローズされなければならず、ステートメントは CLOSESTATEMENT 関数を使用してクローズされなければなりません。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE         PIC S9(8) BINARY.
   01 CON-HANDLE         PIC S9(8) BINARY.
   01 STMT-HANDLE        PIC S9(8) BINARY.
   01 CATALOG            PIC X(64).
   01 CATALOG-LEN        PIC S9(8) BINARY.
   01 SCHEMA             PIC X(nnn).
   01 SCHEMA-LEN         PIC S9(8) BINARY.
   01 TABLENAME          PIC X(nnn).
   01 TABLENAME-LEN      PIC S9(8) BINARY.
   01 SCOPE              PIC S9(8) BINARY.
   01 NULLABLE           PIC S9(8) BINARY.
   01 RETCODE            PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-DBBESTROWIDENT ENV-HANDLE CON-HANDLE
        STMT-HANDLE CATALOG CATALOG-LEN SCHEMA SCHEMA-LEN
        TABLENAME TABLENAME-LEN SCOPE NULLABLE RETCODE.
```

FUNCTION (入力)
DBBESTROWIDENT を含む 16 バイトの文字フィールド。このフィールドは左寄せされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

CON-HANDLE (入力)
接続ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (出力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。この関数呼び出しにより設定された STMT-HANDLE 値は、ステートメントを必要とする他の DBCLI 関数への後続の呼び出しで指定される必要があります。

CATALOG (入力)
カタログ名を含む文字フィールド。このフィールドは CATALOG-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。CATALOG がブランクで構成されている場合 (しかし CATALOG-LEN がゼロより大きい場合)、関数はカタログのないものを検索します。
CATALOG-LEN がゼロの場合、カタログ名は検索の絞り込みに使用されません。

CATALOG-LEN (入力)
CATALOG パラメーターで指定されたカタログ名の長さを含む、フルワード・バイナリー変数。

SCHEMA (入力)
スキーマ名を含む文字フィールド。このフィールドは SCHEMA-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。SCHEMA がブランクで構成されている場合 (しかし SCHEMA-LEN がゼロより大きい場合)、関数はスキーマのないものを検索します。 SCHEMA-LEN がゼロの場合、スキーマ名は検索の絞り込みに使用されません。

SCHEMA-LEN (入力)
SCHEMA パラメーターで指定されたスキーマ名の長さを含む、フルワード・バイナリー変数。

TABLENAME (入力)
テーブル名を含む文字フィールド。このフィールドは TABLENAME-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。

TABLENAME-LEN (入力)
TABLENAME パラメーターで指定されたテーブル名の長さを含む、フルワード・バイナリー変数。

SCOPE (入力)
関心対象の有効範囲が含まれるフルワード・バイナリー変数。可能な値は、以下のとおりです。

1. BESTROW-TEMPORARY (0) - 行を使用している間のみ、極めて一時的に有効
2. BESTROW-TRANSACTION (1) - 現行トランザクションの残りに対して有効
3. BESTROW-SESSION (2) - 現行セッションの残りに対して有効

NULLABLE (入力)
BOOLEAN-FALSE (0) または BOOLEAN-TRUE (1) のいずれかを含むフルワード・バイナリー変数。このパラメーターは、NULL 可能な列が含まれるかどうかを判別します。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

戻されたカーソルは、以下の列を含みます。

1. SCOPE (short): 結果の実際の有効範囲
   - BESTROW-TEMPORARY (0) - 最良行 ID の有効範囲がごく一時的で、行の使用中にのみ存続することを示す
   - BESTROW-TRANSACTION (1) - 現行トランザクションの残りに対して有効
   - BESTROW-SESSION (2) - 現行セッションの残りに対して有効
2. COLUMN_NAME (ストリング): 列名
3. DATA_TYPE (int): java.sql.Types からの SQL データ・タイプ

4. TYPE_NAME (String): データ・ソース依存のタイプ名。UDT の場合はこのタイプ名は完全修飾です。
5. COLUMN_SIZE (int): 精度
6. BUFFER_LENGTH (int): 使用されない
7. DECIMAL_DIGITS (short): スケール
8. PSEUDO_COLUMN (short): Oracle ROWID などの疑似列
   - BESTROW-UNKNOWN (0) - 疑似列の場合もそうでない場合もある
   - BESTROW-NOTPSEUDO (1) - 疑似列ではない
   - BESTROW-PSEUDO (2) - 疑似列

DBCATALOGS

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

DBCATALOGS 関数は、このデータベースで使用可能なカタログ名のリストを取得します。これはステートメント・ハンドルを割り振り、 STMT-HANDLE パラメーターを設定します。このステートメントは、BINDCOLUMN および FETCH などの関数を使用して戻された情報を検索するのに使用される必要がある、オープン・カーソルを含みます。カーソルは CLOSECURSOR 関数を使用してクローズされなければならず、ステートメントは CLOSESTATEMENT 関数を使用してクローズされなければなりません。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE         PIC S9(8) BINARY.
   01 CON-HANDLE         PIC S9(8) BINARY.
   01 STMT-HANDLE        PIC S9(8) BINARY.
   01 RETCODE            PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-DBCATALOGS ENV-HANDLE CON-HANDLE
        STMT-HANDLE RETCODE.
```

FUNCTION (入力)
DBCATALOGS を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

CON-HANDLE (入力)
接続ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (出力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。この関数呼び出しにより設定された STMT-HANDLE 値は、ステートメントを必要とする他の DBCLI 関数への後続の呼び出しで指定される必要があります。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

戻されたカーソルは、以下の列を含みます。

- TABLE_CAT (String): カタログ名

DBCOLUMNPRIV

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

DBCOLUMNPRIV 関数は、表の列のアクセス権の記述を取得します。これはステートメント・ハンドルを割り振り、 STMT-HANDLE パラメーターを設定します。このステートメントは、BINDCOLUMN および FETCH などの関数を使用して戻された情報を検索するのに使用される必要がある、オープン・カーソルを含みます。カーソルは CLOSECURSOR 関数を使用してクローズされなければならず、ステートメントは CLOSESTATEMENT 関数を使用してクローズされなければなりません。列名の基準に一致する特権のみが返されます。これらは、COLUMN_NAME、PRIVILEGE の順に並べ替えられます。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE         PIC S9(8) BINARY.
   01 CON-HANDLE         PIC S9(8) BINARY.
   01 STMT-HANDLE        PIC S9(8) BINARY.
   01 CATALOG            PIC X(nnn).
   01 CATALOG-LEN        PIC S9(8) BINARY.
   01 SCHEMA             PIC X(nnn).
   01 SCHEMA-LEN         PIC S9(8) BINARY.
   01 TABLENAME          PIC X(nnn).
   01 TABLENAME-LEN      PIC S9(8) BINARY.
   01 COLNAMEPATT        PIC X(nnn).
   01 COLNAMEPATT-LEN    PIC S9(8) BINARY.
   01 RETCODE            PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-DBCOLUMNPRIV ENV-HANDLE CON-HANDLE
        STMT-HANDLE CATALOG CATALOG-LEN SCHEMA SCHEMA-LEN
        TABLENAME TABLENAME-LEN COLNAMEPATT COLNAMEPATT-LEN RETCODE.
```

FUNCTION (入力)
DBCOLUMNPRIV を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

CON-HANDLE (入力)
接続ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (出力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。この関数呼び出しにより設定された STMT-HANDLE 値は、ステートメントを必要とする他の DBCLI 関数への後続の呼び出しで指定される必要があります。

CATALOG (入力)
カタログ名を含む文字フィールド。このフィールドは CATALOG-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。CATALOG がブランクで構成されている場合 (しかし CATALOG-LEN がゼロより大きい場合)、関数はカタログのないものを検索します。
CATALOG-LEN がゼロの場合、カタログ名は検索の絞り込みに使用されません。

CATALOG-LEN (入力)
CATALOG パラメーターで指定されたカタログ名の長さを含む、フルワード・バイナリー変数。

SCHEMA (入力)
スキーマ名を含む文字フィールド。このフィールドは SCHEMA-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。SCHEMA がブランクで構成されている場合 (しかし SCHEMA-LEN がゼロより大きい場合)、関数はスキーマのないものを検索します。 SCHEMA-LEN がゼロの場合、スキーマ名は検索の絞り込みに使用されません。

SCHEMA-LEN (入力)
SCHEMA パラメーターで指定されたスキーマ名の長さを含む、フルワード・バイナリー変数。

TABLENAME (入力)
テーブル名を含む文字フィールド。このフィールドは TABLENAME-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。

TABLENAME-LEN (入力)
TABLENAME パラメーターで指定されたテーブル名の長さを含む、フルワード・バイナリー変数。

COLNAMEPATT (入力)
列名パターンを含む文字フィールド。このフィールドは COLNAMEPATT-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。COLNAMEPATT-LEN がゼロの場合、列名パターンは検索の絞り込みに使用されません。このパターン内では以下のとおりです。

- '%' は 0 文字以上の任意のサブストリングと一致することを意味します。
- '_' は任意の 1 文字と一致することを意味します。

COLNAMEPATT-LEN (入力)
COLNAMEPATT パラメーターで指定された列名パターンの長さを含む、フルワード・バイナリー変数。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

戻されたカーソルは、以下の列を含みます。

1. TABLE_CAT (ストリング): テーブル・カタログ (NULL の場合もあります)
2. TABLE_SCHEM (ストリング): テーブル・スキーマ (NULL の場合もあります)
3. TABLE_NAME (ストリング): テーブル名
4. COLUMN_NAME (ストリング): 列名
5. GRANTOR (String): アクセス権限の許可者 (NULL の場合もあります)
6. GRANTEE (String): アクセス権限の被許可者
7. PRIVILEGE (String): アクセス権限の名前
(SELECT、INSERT、UPDATE、REFRENCES など)
8. IS_GRANTABLE (String): 被許可者が他のユーザーに権限を付与することが許可される場合は「YES」、それ以外の場合は「NO」、不明の場合は NULL

DBCOLUMNS

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

DBCOLUMNS 関数は、指定されたカタログ内にある表の列の記述を取得します。これはステートメント・ハンドルを割り振り、 STMT-HANDLE パラメーターを設定します。このステートメントは、BINDCOLUMN および FETCH などの関数を使用して戻された情報を検索するのに使用される必要がある、オープン・カーソルを含みます。カーソルは CLOSECURSOR 関数を使用してクローズされなければならず、ステートメントは CLOSESTATEMENT 関数を使用してクローズされなければなりません。カタログ、スキーマ、表、および列名の基準に一致する列の記述のみが返されます。これらは、TABLE_SCHEM、TABLE_NAME、ORDINAL_POSITION の順に並べ替えられます。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE         PIC S9(8) BINARY.
   01 CON-HANDLE         PIC S9(8) BINARY.
   01 STMT-HANDLE        PIC S9(8) BINARY.
   01 CATALOG            PIC X(nnn).
   01 CATALOG-LEN        PIC S9(8) BINARY.
   01 SCHEMAPATT         PIC X(nnn).
   01 SCHEMAPATT-LEN     PIC S9(8) BINARY.
   01 TABLEPATT          PIC X(nnn).
   01 TABLEPATT-LEN      PIC S9(8) BINARY.
   01 COLNAMEPATT        PIC X(nnn).
   01 COLNAMEPATT-LEN    PIC S9(8) BINARY.
   01 RETCODE            PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-DBCOLUMNS ENV-HANDLE CON-HANDLE
        STMT-HANDLE CATALOG CATALOG-LEN SCHEMAPATT SCHEMAPATT-LEN
        TABLEPATT TABLEPATT-LEN COLNAMEPATT COLNAMEPATT-LEN RETCODE.
```

FUNCTION (入力)
DBCOLUMNS を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

CON-HANDLE (入力)
接続ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (出力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。この関数呼び出しにより設定された STMT-HANDLE 値は、ステートメントを必要とする他の DBCLI 関数への後続の呼び出しで指定される必要があります。

CATALOG (入力)
カタログ名を含む文字フィールド。このフィールドは CATALOG-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。CATALOG がブランクで構成されている場合 (しかし CATALOG-LEN がゼロより大きい場合)、関数はカタログのないものを検索します。
CATALOG-LEN がゼロの場合、カタログ名は検索の絞り込みに使用されません。

CATALOG-LEN (入力)
CATALOG パラメーターで指定されたカタログ名の長さを含む、フルワード・バイナリー変数。

SCHEMAPATT (入力)
スキーマ名パターンを含む文字フィールド。このフィールドは SCHEMAPATT-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。SCHEMAPATT がブランクで構成されている場合 (しかし SCHEMAPATT-LEN がゼロより大きい場合)、関数はスキーマのないものを検索します。 SCHEMAPATT-LEN がゼロの場合、スキーマ名パターンは検索の絞り込みに使用されません。このパターン内では以下のとおりです。

- '%' は 0 文字以上の任意のサブストリングと一致することを意味します。
- '_' は任意の 1 文字と一致することを意味します。

SCHEMAPATT-LEN (入力)
SCHEMAPATT パラメーターで指定されたスキーマ名パターンの長さを含む、フルワード・バイナリー変数。

TABLEPATT (入力)
テーブル名を含む文字フィールド。このフィールドは TABLEPATT-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。TABLEPATT-LEN がゼロの場合、テーブル名は検索の絞り込みに使用されません。このパターン内では以下のとおりです。

- '%' は 0 文字以上の任意のサブストリングと一致することを意味します。
- '_' は任意の 1 文字と一致することを意味します。

TABLEPATT-LEN (入力)
TABLEPATT パラメーターで指定されたテーブル名の長さを含む、フルワード・バイナリー変数。

COLNAMEPATT (入力)
列名パターンを含む文字フィールド。このフィールドは COLNAMEPATT-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。COLNAMEPATT-LEN がゼロの場合、列名パターンは検索の絞り込みに使用されません。このパターン内では以下のとおりです。

- '%' は 0 文字以上の任意のサブストリングと一致することを意味します。
- '_' は任意の 1 文字と一致することを意味します。

COLNAMEPATT-LEN (入力)
COLNAMEPATT パラメーターで指定された列名パターンの長さを含む、フルワード・バイナリー変数。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

戻されたカーソルは、以下の列を含みます。

1. TABLE_CAT (ストリング): テーブル・カタログ (NULL の場合もあります)
2. TABLE_SCHEM (ストリング): テーブル・スキーマ (NULL の場合もあります)
3. TABLE_NAME (ストリング): テーブル名
4. COLUMN_NAME (ストリング): 列名

5. DATA_TYPE (int): データ・タイプ SQL タイプ
6. TYPE_NAME (String): データ・ソースに依存するタイプ名。UDT の場合、タイプ名は完全修飾です。
7. COLUMN_SIZE (int): 列サイズ。文字タイプまたは日付タイプの場合、これは文字の最大数です。数字タイプまたは 10 進数タイプの場合、これは精度です。
8. BUFFER_LENGTH は使用されません。
9. DECIMAL_DIGITS (int): 小数桁の数
10. NUM_PREC_RADIX (int): 基数 (一般的には 10 または 2)
11. NULLABLE (int): NULL が許可されるかどうか
    - NULLABLE-NONULLS (0): NULL 値を許可しない場合がある
    - NULLABLE-NULLABLE(1): NULL 値を例外なく許可する
    - NULLABLE-UNKNOWN(2): NULL 可能性は不明
12. REMARKS (String): 列を説明するコメント (NULL の場合もあります)
13. COLUMN_DEF (String): デフォルト値 (NULL の場合もあります)
14. SQL_DATA_TYPE (int):未使用
15. SQL_DATETIME_SUB (int): 未使用
16. CHAR_OCTET_LENGTH (int): 文字タイプの場合、列内のバイトの最大数
17. ORDINAL_POSITION (int): 表内の列の索引 (1 で始まる)
18. IS_NULLABLE (String):「NO」は、列が NULL 値を例外なく許可しないことを意味します。「YES」は、列が NULL 値を許可する場合があることを意味します。空のストリングは不明を意味します。
19. SCOPE_CATALOG (String): 参照属性の有効範囲である表のカタログ (DATA_TYPE が REF でない場合は NULL)
20. SCOPE_SCHEMA (String): 参照属性の有効範囲である表のスキーマ (DATA_TYPE が REF でない場合は NULL)
21. SCOPE_TABLE (String): 参照属性の有効範囲である表名 (DATA_TYPE が REF でない場合は NULL)
22. SOURCE_DATA_TYPE (short): 特殊タイプまたはユーザー生成 Ref タイプのソース・タイプ、java.sql.Types からの SQL タイプ (DATA_TYPE が DISTINCT またはユーザー生成 REF でない場合は NULL)

DBCROSSREFERENCE

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

DBCROSSREFERENCE 関数は、特定の主キー表の主キー列を参照する特定の外部キー表内の外部キー列の記述を取得します (1 つの表が別の表のキーをどのようにインポートするかを記述します)。これはステートメント・ハンドルを割り振り、STMT-HANDLE パラメーターを設定します。このステートメントは、BINDCOLUMN および FETCH などの関数を使用して戻された情報を検索するのに使用される必要がある、オープン・カーソルを含みます。カーソルは CLOSECURSOR 関数を使用してクローズされなければならず、ステートメントは CLOSESTATEMENT 関数を使用してクローズされなければなりません。ほとんどの表が外部キーを 1 つのテーブルから 1 回のみインポートするため、通常、この関数は外部キー/主キーのペアを 1 組のみ

返します。これらは、
FKTABLE_CAT、FKTABLE_SCHEM、FKTABLE_NAME、KEY_SEQ の順に並べ替えられます。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE         PIC S9(8) BINARY.
   01 CON-HANDLE         PIC S9(8) BINARY.
   01 STMT-HANDLE        PIC S9(8) BINARY.
   01 PRIMCATALOG        PIC X(nnn).
   01 PRIMCATALOG-LEN    PIC S9(8) BINARY.
   01 PRIMSCHEMA         PIC X(nnn).
   01 PRIMSCHEMA-LEN     PIC S9(8) BINARY.
   01 PRIMTABLE          PIC X(nnn).
   01 PRIMTABLE-LEN      PIC S9(8) BINARY.
   01 FORCATALOG         PIC X(nnn).
   01 FORCATALOG-LEN     PIC S9(8) BINARY.
   01 FORSCHEMA          PIC X(nnn).
   01 FORSCHEMA-LEN      PIC S9(8) BINARY.
   01 FORTABLE           PIC X(nnn).
   01 FORTABLE-LEN       PIC S9(8) BINARY.
   01 RETCODE            PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-DBCROSSREFERENCE ENV-HANDLE CON-HANDLE
        STMT-HANDLE PRIMCATALOG PRIMCATALOG-LEN PRIMSCHEMA PRIMSCHEMA-LEN
        PRIMTABLE PRIMTABLE-LEN FORCATALOG FORCATALOG-LEN
        FORSCHEMA FORSCHEMA-LEN FORTABLE FORTABLE-LEN RETCODE.
```

FUNCTION (入力)
DBCROSSREFERENCE を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

CON-HANDLE (入力)
接続ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (出力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。この関数呼び出しにより設定された STMT-HANDLE 値は、ステートメントを必要とする他の DBCLI 関数への後続の呼び出しで指定される必要があります。

PRIMCATALOG (入力)
プライマリー・カタログ名を含む文字フィールド。このフィールドは PRIMCATALOG-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。PRIMCATALOG がブランクで構成されている場合 (しかし PRIMCATALOG-LEN がゼロより大きい場合)、関数はカタログのないものを検索します。PRIMCATALOG-LEN がゼロの場合、プライマリー・カタログ名は検索の絞り込みに使用されません。

PRIMCATALOG-LEN (入力)
PRIMCATALOG パラメーターで指定されたカタログ名の長さを含む、フルワード・バイナリー変数。

PRIMSCHEMA (入力)
プライマリー・スキーマ名を含む文字フィールド。このフィールドは PRIMSCHEMA-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。PRIMSCHEMA がブランクで構成されている場合 (しかし PRIMSCHEMA-LEN がゼロより大きい場合)、関数はスキーマのな

いものを検索します。PRIMSCHEMA-LEN がゼロの場合、プライマリー・スキーマ名は検索の絞り込みに使用されません。

PRIMSCHEMA-LEN (入力)
PRIMSCHEMA パラメーターで指定されたプライマリー・スキーマ名の長さを含む、フルワード・バイナリー変数。

PRIMTABLE (入力)
プライマリー表名を含む文字フィールド。このフィールドは PRIMTABLE-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。

PRIMTABLE-LEN (入力)
PRIMTABLE パラメーターで指定されたプライマリー表名の長さを含む、フルワード・バイナリー変数。

FORCATALOG (入力)
外部カタログ名を含む文字フィールド。このフィールドは FORCATALOG-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。FORCATALOG がブランクで構成されている場合 (しかし FORCATALOG-LEN がゼロより大きい場合)、関数はカタログのないものを検索します。FORCATALOG-LEN がゼロの場合、カタログ名は検索の絞り込みに使用されません。

FORCATALOG-LEN (入力)
FORCATALOG パラメーターで指定された外部カタログ名の長さを含む、フルワード・バイナリー変数。

FORSCHEMA (入力)
外部スキーマ名を含む文字フィールド。このフィールドは FORSCHEMA-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。FORSCHEMA がブランクで構成されている場合 (しかし FORSCHEMA-LEN がゼロより大きい場合)、関数はスキーマのないものを検索します。FORSCHEMA-LEN がゼロの場合、外部スキーマ名は検索の絞り込みに使用されません。

FORSCHEMA-LEN (入力)
FORSCHEMA パラメーターで指定された外部スキーマ名の長さを含む、フルワード・バイナリー変数。

FORTABLE (入力)
外部表名を含む文字フィールド。このフィールドは FORTABLE-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。

FORTABLE-LEN (入力)
FORTABLE パラメーターで指定された外部表名の長さを含む、フルワード・バイナリー変数。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

戻されたカーソルは、以下の列を含みます。

1. PKTABLE_CAT (ストリング): 主キー・テーブル・カタログ (NULL の場合もあります)

2. PKTABLE_SCHEM (ストリング): 主キー・テーブル・スキーマ (NULL の場合もあります)
3. PKTABLE_NAME (ストリング): 主キー・テーブル名
4. PKCOLUMN_NAME (ストリング): 主キー列名
5. FKTABLE_CAT (String): エクスポートされる外部キー表カタログ (NULL の場合もあります)
6. FKTABLE_SCHEM (String): エクスポートされる外部キー表スキーマ (NULL の場合もあります)
7. FKTABLE_NAME (String): エクスポートされる外部キー表名
8. FKCOLUMN_NAME (String): エクスポートされる外部キー列名
9. KEY_SEQ (short): 外部キー内の順序番号
10. UPDATE_RULE (short): 主キーが更新された場合の外部キーに対する処理。
    - IMPORTEDKEY-CASCADE (0) - 主キーの更新と一致するように、インポートされたキーを変更します。
    - IMPORTEDKEY-RESTRICT (1) - IMPORTEDKEY-NOACTION と同じ (ODBC 2.x 互換性のため)
    - IMPORTEDKEY-SETNULL (2) - 主キーが更新された場合、インポートされたキーを NULL に変更します
    - IMPORTEDKEY-NOACTION (3) - 主キーがインポートされている場合、その主キーの更新を許可しません
    - IMPORTEDKEY-SETDEFAULT (4) - 主キーが更新された場合、インポートされたキーをデフォルト値に変更します
11. DELETE_RULE (short): 主キーが削除された場合の外部キーに対する処理。
    - IMPORTEDKEY-CASCADE (0) - 削除されたキーをインポートする行を削除します
    - IMPORTEDKEY-RESTRICT (1) - IMPORTEDKEY-NOACTION と同じ (ODBC 2.x 互換性のため)
    - IMPORTEDKEY-SETNULL (2) - 主キーが削除された場合、インポートされたキーを NULL に変更します
    - IMPORTEDKEY-NOACTION (3) - 主キーがインポートされている場合、その主キーの削除を許可しません
    - IMPORTEDKEY-SETDEFAULT (4) - 主キーが削除された場合、インポートされたキーをデフォルト値に変更します
12. FK_NAME (String): 外部キー名 (NULL の場合もあります)
13. PK_NAME (String): 主キー名 (NULL の場合もあります)
14. DEFERRABILITY (short): コミットまで、外部キーの制約の評価を据え置くことができるかどうか
    - IMPORTEDKEY-INITIALLYDEFERRED (5) - 定義については SQL92 を参照してください
    - IMPORTEDKEY-INITIALLYIMMEDIATE (6) - 定義については SQL92 を参照してください
    - IMPORTEDKEY-NOTDEFERRABLE (7) - 定義については SQL92 を参照してください

DBEXPORTEDKEYS

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

DBEXPORTEDKEYS 関数は、特定の表の主キー列を参照する外部キー列の記述を取得します (表によってエクスポートされた外部キー)。これはステートメント・ハンドルを割り振り、 STMT-HANDLE パラメーターを設定します。このステートメントは、BINDCOLUMN および FETCH などの関数を使用して戻された情報を検索するのに使用される必要がある、オープン・カーソルを含みます。カーソルは CLOSECURSOR 関数を使用してクローズされなければならず、ステートメントは CLOSESTATEMENT 関数を使用してクローズされなければなりません。これらは、FKTABLE_CAT、FKTABLE_SCHEM、FKTABLE_NAME、KEY_SEQ の順に並べ替えられます。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE            PIC S9(8) BINARY.
   01 CON-HANDLE            PIC S9(8) BINARY.
   01 STMT-HANDLE           PIC S9(8) BINARY.
   01 CATALOG               PIC X(nnn).
   01 CATALOG-LEN           PIC S9(8) BINARY.
   01 SCHEMA                PIC X(nnn).
   01 SCHEMA-LEN            PIC S9(8) BINARY.
   01 TABLENAME             PIC X(nnn).
   01 TABLENAME-LEN         PIC S9(8) BINARY.
   01 RETCODE               PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-DBEXPORTEDKEYS ENV-HANDLE CON-HANDLE
        STMT-HANDLE CATALOG CATALOG-LEN SCHEMA SCHEMA-LEN
        TABLENAME TABLENAME-LEN RETCODE.
```

FUNCTION (入力)
DBEXPORTEDKEYS を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

CON-HANDLE (入力)
接続ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (出力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。この関数呼び出しにより設定された STMT-HANDLE 値は、ステートメントを必要とする他の DBCLI 関数への後続の呼び出しで指定される必要があります。

CATALOG (入力)
カタログ名を含む文字フィールド。このフィールドは CATALOG-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。CATALOG がブランクで構成されている場合 (しかし CATALOG-LEN がゼロより大きい場合)、関数はカタログのないものを検索します。
CATALOG-LEN がゼロの場合、カタログ名は検索の絞り込みに使用されません。

CATALOG-LEN (入力)
CATALOG パラメーターで指定されたカタログ名の長さを含む、フルワード・バイナリー変数。

SCHEMA (入力)
スキーマ名を含む文字フィールド。このフィールドは SCHEMA-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。SCHEMA がブランクで構成されている場合 (しかし SCHEMA-LEN がゼロより大きい場合)、関数はスキーマのないものを検索します。 SCHEMA-LEN がゼロの場合、スキーマ名は検索の絞り込みに使用されません。

SCHEMA-LEN (入力)
SCHEMA パラメーターで指定されたスキーマ名の長さを含む、フルワード・バイナリー変数。

TABLENAME (入力)
テーブル名を含む文字フィールド。このフィールドは TABLENAME-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。

TABLENAME-LEN (入力)
TABLENAME パラメーターで指定されたテーブル名の長さを含む、フルワード・バイナリー変数。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

戻されたカーソルは、以下の列を含みます。

1. PKTABLE_CAT (ストリング): 主キー・テーブル・カタログ (NULL の場合もあります)
2. PKTABLE_SCHEM (ストリング): 主キー・テーブル・スキーマ (NULL の場合もあります)
3. PKTABLE_NAME (ストリング): 主キー・テーブル名
4. PKCOLUMN_NAME (ストリング): 主キー列名
5. FKTABLE_CAT (String): エクスポートされる外部キー表カタログ (NULL の場合もあります)
6. FKTABLE_SCHEM (String): エクスポートされる外部キー表スキーマ (NULL の場合もあります)
7. FKTABLE_NAME (String): エクスポートされる外部キー表名
8. FKCOLUMN_NAME (String): エクスポートされる外部キー列名
9. KEY_SEQ (short): 外部キー内の順序番号
10. UPDATE_RULE (short): 主キーが更新された場合の外部キーに対する処理。
    - IMPORTEDKEY-CASCADE (0) - 主キーの更新と一致するように、インポートされたキーを変更します。
    - IMPORTEDKEY-RESTRICT (1) - IMPORTEDKEY-NOACTION と同じ (ODBC 2.x 互換性のため)
    - IMPORTEDKEY-SETNULL (2) - 主キーが更新された場合、インポートされたキーを NULL に変更します
    - IMPORTEDKEY-NOACTION (3) - 主キーがインポートされている場合、その主キーの更新を許可しません

- IMPORTEDKEY-SETDEFAULT (4) - 主キーが更新された場合、インポートされたキーをデフォルト値に変更します

11. DELETE_RULE (short): 主キーが削除された場合の外部キーに対する処理。
    - IMPORTEDKEY-CASCADE (0) - 削除されたキーをインポートする行を削除します
    - IMPORTEDKEY-RESTRICT (1) - IMPORTEDKEY-NOACTION と同じ (ODBC 2.x 互換性のため)
    - IMPORTEDKEY-SETNULL (2) - 主キーが削除された場合、インポートされたキーを NULL に変更します
    - IMPORTEDKEY-NOACTION (3) - 主キーがインポートされている場合、その主キーの削除を許可しません
    - IMPORTEDKEY-SETDEFAULT (4) - 主キーが削除された場合、インポートされたキーをデフォルト値に変更します
12. FK_NAME (String): 外部キー名 (NULL の場合もあります)
13. PK_NAME (String): 主キー名 (NULL の場合もあります)
14. DEFERRABILITY (short): コミットまで、外部キーの制約の評価を据え置くことができるかどうか
    - IMPORTEDKEY-INITIALLYDEFERRED (5) - 定義については SQL92 を参照してください
    - IMPORTEDKEY-INITIALLYIMMEDIATE (6) - 定義については SQL92 を参照してください
    - IMPORTEDKEY-NOTDEFERRABLE (7) - 定義については SQL92 を参照してください

DBIMPORTEDKEYS

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

DBIMPORTEDKEYS 関数は、表の外部キー列によって参照される主キー列の記述を取得します (表によってインポートされる主キー)。これはステートメント・ハンドルを割り振り、 STMT-HANDLE パラメーターを設定します。このステートメントは、BINDCOLUMN および FETCH などの関数を使用して戻された情報を検索するのに使用される必要がある、オープン・カーソルを含みます。カーソルは CLOSECURSOR 関数を使用してクローズされなければならず、ステートメントは CLOSESTATEMENT 関数を使用してクローズされなければなりません。これらは、FKTABLE_CAT、FKTABLE_SCHEM、FKTABLE_NAME、KEY_SEQ の順に並べ替えられます。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE         PIC S9(8) BINARY.
   01 CON-HANDLE         PIC S9(8) BINARY.
   01 STMT-HANDLE        PIC S9(8) BINARY.
   01 CATALOG            PIC X(nnn).
   01 CATALOG-LEN        PIC S9(8) BINARY.
   01 SCHEMA             PIC X(nnn).
   01 SCHEMA-LEN         PIC S9(8) BINARY.
   01 TABLENAME          PIC X(nnn).
   01 TABLENAME-LEN      PIC S9(8) BINARY.
```

```
   01 RETCODE            PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-DBIMPORTEDKEYS ENV-HANDLE CON-HANDLE
        STMT-HANDLE CATALOG CATALOG-LEN SCHEMA SCHEMA-LEN
        TABLENAME TABLENAME-LEN RETCODE.
```

FUNCTION (入力)
DBIMPORTEDKEYS を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

CON-HANDLE (入力)
接続ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (出力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。この関数呼び出しにより設定された STMT-HANDLE 値は、ステートメントを必要とする他の DBCLI 関数への後続の呼び出しで指定される必要があります。

CATALOG (入力)
カタログ名を含む文字フィールド。このフィールドは CATALOG-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。CATALOG がブランクで構成されている場合 (しかし CATALOG-LEN がゼロより大きい場合)、関数はカタログのないものを検索します。
CATALOG-LEN がゼロの場合、カタログ名は検索の絞り込みに使用されません。

CATALOG-LEN (入力)
CATALOG パラメーターで指定されたカタログ名の長さを含む、フルワード・バイナリー変数。

SCHEMA (入力)
スキーマ名を含む文字フィールド。このフィールドは SCHEMA-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。SCHEMA がブランクで構成されている場合 (しかし SCHEMA-LEN がゼロより大きい場合)、関数はスキーマのないものを検索します。 SCHEMA-LEN がゼロの場合、スキーマ名は検索の絞り込みに使用されません。

SCHEMA-LEN (入力)
SCHEMA パラメーターで指定されたスキーマ名の長さを含む、フルワード・バイナリー変数。

TABLENAME (入力)
テーブル名を含む文字フィールド。このフィールドは TABLENAME-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。

TABLENAME-LEN (入力)
TABLENAME パラメーターで指定されたテーブル名の長さを含む、フルワード・バイナリー変数。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

戻されたカーソルは、以下の列を含みます。

1. PKTABLE_CAT (ストリング): 主キー・テーブル・カタログ (NULL の場合もあります)
2. PKTABLE_SCHEM (ストリング): 主キー・テーブル・スキーマ (NULL の場合もあります)
3. PKTABLE_NAME (ストリング): 主キー・テーブル名
4. PKCOLUMN_NAME (ストリング): 主キー列名
5. FKTABLE_CAT (String): エクスポートされる外部キー表カタログ (NULL の場合もあります)
6. FKTABLE_SCHEM (String): エクスポートされる外部キー表スキーマ (NULL の場合もあります)
7. FKTABLE_NAME (String): エクスポートされる外部キー表名
8. FKCOLUMN_NAME (String): エクスポートされる外部キー列名
9. KEY_SEQ (short): 外部キー内の順序番号
10. UPDATE_RULE (short): 主キーが更新された場合の外部キーに対する処理。
    - IMPORTEDKEY-CASCADE (0) - 主キーの更新と一致するように、インポートされたキーを変更します。
    - IMPORTEDKEY-RESTRICT (1) - IMPORTEDKEY-NOACTION と同じ (ODBC 2.x 互換性のため)
    - IMPORTEDKEY-SETNULL (2) - 主キーが更新された場合、インポートされたキーを NULL に変更します
    - IMPORTEDKEY-NOACTION (3) - 主キーがインポートされている場合、その主キーの更新を許可しません
    - IMPORTEDKEY-SETDEFAULT (4) - 主キーが更新された場合、インポートされたキーをデフォルト値に変更します
11. DELETE_RULE (short): 主キーが削除された場合の外部キーに対する処理。
    - IMPORTEDKEY-CASCADE (0) - 削除されたキーをインポートする行を削除します
    - IMPORTEDKEY-RESTRICT (1) - IMPORTEDKEY-NOACTION と同じ (ODBC 2.x 互換性のため)
    - IMPORTEDKEY-SETNULL (2) - 主キーが削除された場合、インポートされたキーを NULL に変更します
    - IMPORTEDKEY-NOACTION (3) - 主キーがインポートされている場合、その主キーの削除を許可しません
    - IMPORTEDKEY-SETDEFAULT (4) - 主キーが削除された場合、インポートされたキーをデフォルト値に変更します
12. FK_NAME (String): 外部キー名 (NULL の場合もあります)
13. PK_NAME (String): 主キー名 (NULL の場合もあります)
14. DEFERRABILITY (short): コミットまで、外部キーの制約の評価を据え置くことができるかどうか
    - IMPORTEDKEY-INITIALLYDEFERRED (5) - 定義については SQL92 を参照してください

- IMPORTEDKEY-INITIALLYIMMEDIATE (6) - 定義については SQL92 を参照してください
- IMPORTEDKEY-NOTDEFERRABLE (7) - 定義については SQL92 を参照してください

DBINDEXINFO

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

DBINDEXINFO 関数は、特定の表の索引の記述および統計を取得します。これらは、NON_UNIQUE、TYPE、INDEX_NAME、ORDINAL_POSITION の順に並べ替えられます。これはステートメント・ハンドルを割り振り、 STMT-HANDLE パラメーターを設定します。このステートメントは、BINDCOLUMN および FETCH などの関数を使用して戻された情報を検索するのに使用される必要がある、オープン・カーソルを含みます。カーソルは CLOSECURSOR 関数を使用してクローズされなければならず、ステートメントは CLOSESTATEMENT 関数を使用してクローズされなければなりません。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE         PIC S9(8) BINARY.
   01 CON-HANDLE         PIC S9(8) BINARY.
   01 STMT-HANDLE        PIC S9(8) BINARY.
   01 CATALOG            PIC X(nnn).
   01 CATALOG-LEN        PIC S9(8) BINARY.
   01 SCHEMA             PIC X(nnn).
   01 SCHEMA-LEN         PIC S9(8) BINARY.
   01 TABLENAME          PIC X(nnn).
   01 TABLENAME-LEN      PIC S9(8) BINARY.
   01 UNIQUE             PIC S9(8) BINARY.
   01 APPROXIMATE        PIC S9(8) BINARY.
   01 RETCODE            PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-DBINDEXINFO ENV-HANDLE CON-HANDLE
        STMT-HANDLE CATALOG CATALOG-LEN SCHEMA SCHEMA-LEN
        TABLENAME TABLENAME-LEN UNIQUE APPROXIMATE RETCODE.
```

FUNCTION (入力)
DBINDEXINFO を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

CON-HANDLE (入力)
接続ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (出力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。この関数呼び出しにより設定された STMT-HANDLE 値は、ステートメントを必要とする他の DBCLI 関数への後続の呼び出しで指定される必要があります。

CATALOG (入力)
カタログ名を含む文字フィールド。このフィールドは CATALOG-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。CATALOG がブランクで構成されている場合 (しかし CATALOG-LEN が

ゼロより大きい場合)、関数はカタログのないものを検索します。
CATALOG-LEN がゼロの場合、カタログ名は検索の絞り込みに使用されません。

CATALOG-LEN (入力)
CATALOG パラメーターで指定されたカタログ名の長さを含む、フルワード・バイナリー変数。

SCHEMA (入力)
スキーマ名を含む文字フィールド。このフィールドは SCHEMA-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。
SCHEMA がブランクで構成されている場合 (しかし SCHEMA-LEN がゼロより大きい場合)、関数はスキーマのないものを検索します。 SCHEMA-LEN がゼロの場合、スキーマ名は検索の絞り込みに使用されません。

SCHEMA-LEN (入力)
SCHEMA パラメーターで指定されたスキーマ名の長さを含む、フルワード・バイナリー変数。

TABLENAME (入力)
テーブル名を含む文字フィールド。このフィールドは TABLENAME-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。

TABLENAME-LEN (入力)
TABLENAME パラメーターで指定されたテーブル名の長さを含む、フルワード・バイナリー変数。

UNIQUE (入力)
BOOLEAN-FALSE (0) または BOOLEAN-TRUE (1) のいずれかを含むフルワード・バイナリー変数。 true の場合、固有値の索引のみを返します。false の場合、固有かどうかに関係なく、索引を返します。

APPROXIMATE (入力)
BOOLEAN-FALSE (0) または BOOLEAN-TRUE (1) のいずれかを含むフルワード・バイナリー変数。 true の場合、おおよその値またはデータ以外の値が結果に反映されることが許可されます。false の場合、正確な結果が要求されます。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。
戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

戻されたカーソルは、以下の列を含みます。

1. TABLE_CAT (ストリング): テーブル・カタログ (NULL の場合もあります)
2. TABLE_SCHEM (ストリング): テーブル・スキーマ (NULL の場合もあります)
3. TABLE_NAME (ストリング): テーブル名
4. NON_UNIQUE (Boolean): 索引値を非固有にできるかどうか。TYPE が TABLEINDEX-STATISTIC (0) の場合は false です
5. INDEX_QUALIFIER (String): 索引カタログ (NULL の場合もあります)。TYPE が TABLEINDEX-STATISTIC (0) の場合は NULL です

6. INDEX_NAME (String): 索引名。TYPE が TABLEINDEX-STATISTIC (0) の場合は NULL です
7. TYPE (short): 索引タイプ。
   - TABLEINDEX-STATISTIC (0) - 表の索引記述とともに返される表統計をこれによって識別します
   - TABLEINDEX-CLUSTERED (1) - これはクラスター索引です
   - TABLEINDEX-HASHED (2) - これはハッシュ索引です
   - TABLEINDEX-OTHER (3) - これは他のスタイルの索引です
8. ORDINAL_POSITION (short): 索引内の列の順序番号。TYPE が TABLEINDEX-STATISTIC (0) の場合は 0 です。
9. COLUMN_NAME (String): 列名。TYPE が TABLEINDEX-STATISTIC (0) の場合は NULL です。
10. ASC_OR_DESC (String): 列のソート・シーケンス。「A」= 昇順、「D」= 降順。ソート・シーケンスがサポートされていない場合は NULL になることがあります。TYPE が TABLEINDEX-STATISTIC (0) の場合は NULL です
11. CARDINALITY (int): TYPE が TABLEINDEX-STATISTIC (0) の場合、これは表内の行数です。それ以外の場合、これは索引内の固有値の数です。
12. PAGES (int): TYPE が TABLEINDEX-STATISTIC (0) の場合、これは表で使用されているページの数です。それ以外の場合、これは現在の索引で使用されているページの数です。
13. FILTER_CONDITION (String): フィルター条件 (ある場合)。(NULL の場合もあります)

DBPRIMARYKEYS

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

DBPRIMARYKEYS 関数は、特定の表の主キー列の記述を取得します。これらは、COLUMN_NAME の順に並べ替えられます。これはステートメント・ハンドルを割り振り、 STMT-HANDLE パラメーターを設定します。このステートメントは、BINDCOLUMN および FETCH などの関数を使用して戻された情報を検索するのに使用される必要がある、オープン・カーソルを含みます。カーソルは CLOSECURSOR 関数を使用してクローズされなければならず、ステートメントは CLOSESTATEMENT 関数を使用してクローズされなければなりません。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE         PIC S9(8) BINARY.
   01 CON-HANDLE         PIC S9(8) BINARY.
   01 STMT-HANDLE        PIC S9(8) BINARY.
   01 CATALOG            PIC X(nnn).
   01 CATALOG-LEN        PIC S9(8) BINARY.
   01 SCHEMA             PIC X(nnn).
   01 SCHEMA-LEN         PIC S9(8) BINARY.
   01 TABLENAME          PIC X(nnn).
   01 TABLENAME-LEN      PIC S9(8) BINARY.
   01 RETCODE            PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-DBPRIMARYKEYS ENV-HANDLE CON-HANDLE
        STMT-HANDLE CATALOG CATALOG-LEN SCHEMA SCHEMA-LEN
        TABLENAME TABLENAME-LEN RETCODE.
```

FUNCTION (入力)
DBPRIMARYKEYS を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

CON-HANDLE (入力)
接続ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (出力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。この関数呼び出しにより設定された STMT-HANDLE 値は、ステートメントを必要とする他の DBCLI 関数への後続の呼び出しで指定される必要があります。

CATALOG (入力)
カタログ名を含む文字フィールド。このフィールドは CATALOG-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。CATALOG がブランクで構成されている場合 (しかし CATALOG-LEN がゼロより大きい場合)、関数はカタログのないものを検索します。
CATALOG-LEN がゼロの場合、カタログ名は検索の絞り込みに使用されません。

CATALOG-LEN (入力)
CATALOG パラメーターで指定されたカタログ名の長さを含む、フルワード・バイナリー変数。

SCHEMA (入力)
スキーマ名を含む文字フィールド。このフィールドは SCHEMA-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。SCHEMA がブランクで構成されている場合 (しかし SCHEMA-LEN がゼロより大きい場合)、関数はスキーマのないものを検索します。 SCHEMA-LEN がゼロの場合、スキーマ名は検索の絞り込みに使用されません。

SCHEMA-LEN (入力)
SCHEMA パラメーターで指定されたスキーマ名の長さを含む、フルワード・バイナリー変数。

TABLENAME (入力)
テーブル名を含む文字フィールド。このフィールドは TABLENAME-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。

TABLENAME-LEN (入力)
TABLENAME パラメーターで指定されたテーブル名の長さを含む、フルワード・バイナリー変数。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

戻されたカーソルは、以下の列を含みます。

1. TABLE_CAT (ストリング): テーブル・カタログ (NULL の場合もあります)
2. TABLE_SCHEM (ストリング): テーブル・スキーマ (NULL の場合もあります)

3. TABLE_NAME (ストリング): テーブル名
4. COLUMN_NAME (ストリング): 列名
5. KEY_SEQ (short): 主キー内の順序番号
6. PK_NAME (String): 主キー名 (NULL の場合もあります)

DBPROCEDURECOLS

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

DBPROCEDURECOLS 関数は、特定のカタログのストアード・プロシージャー・パラメーターおよび結果列の記述を取得します。これはステートメント・ハンドルを割り振り、 STMT-HANDLE パラメーターを設定します。このステートメントは、BINDCOLUMN および FETCH などの関数を使用して戻された情報を検索するのに使用される必要がある、オープン・カーソルを含みます。カーソルは CLOSECURSOR 関数を使用してクローズされなければならず、ステートメントは CLOSESTATEMENT 関数を使用してクローズされなければなりません。

スキーマ、プロシージャー、およびパラメーター名の基準に一致する記述のみが返されます。これらは、PROCEDURE_SCHEM、PROCEDURE_NAME の順に並べ替えられます。このうち、戻り値 (ある場合) が 1 番目です。以下で、パラメーターを呼び出し順に説明します。列の説明は列番号の順序に従っています。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE         PIC S9(8) BINARY.
   01 CON-HANDLE         PIC S9(8) BINARY.
   01 STMT-HANDLE        PIC S9(8) BINARY.
   01 CATALOG            PIC X(nnn).
   01 CATALOG-LEN        PIC S9(8) BINARY.
   01 SCHEMAPATT         PIC X(nnn).
   01 SCHEMAPATT-LEN     PIC S9(8) BINARY.
   01 PROCNAMEPATT       PIC X(nnn).
   01 PROCNAMEPATT-LEN   PIC S9(8) BINARY.
   01 COLNAMEPATT        PIC X(nnn).
   01 COLNAMEPATT-LEN    PIC S9(8) BINARY.
   01 RETCODE            PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-DBPROCEDURECOLS ENV-HANDLE CON-HANDLE
        STMT-HANDLE CATALOG CATALOG-LEN SCHEMAPATT SCHEMAPATT-LEN
        PROCNAMEPATT PROCNAMEPATT-LEN COLNAMEPATT COLNAMEPATT-LEN RETCODE.
```

FUNCTION (入力)
DBPROCEDURECOLS を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

CON-HANDLE (入力)
接続ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (出力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。この関数呼び出しにより設定された STMT-HANDLE 値は、ステートメントを必要とする他の DBCLI 関数への後続の呼び出しで指定される必要があります。

CATALOG (入力)
カタログ名を含む文字フィールド。このフィールドは CATALOG-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。CATALOG がブランクで構成されている場合 (しかし CATALOG-LEN がゼロより大きい場合)、関数はカタログのないものを検索します。
CATALOG-LEN がゼロの場合、カタログ名は検索の絞り込みに使用されません。

CATALOG-LEN (入力)
CATALOG パラメーターで指定されたカタログ名の長さを含む、フルワード・バイナリー変数。

SCHEMAPATT (入力)
スキーマ名パターンを含む文字フィールド。このフィールドは SCHEMAPATT-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。SCHEMAPATT がブランクで構成されている場合 (しかし SCHEMAPATT-LEN がゼロより大きい場合)、関数はスキーマのないものを検索します。 SCHEMAPATT-LEN がゼロの場合、スキーマ名パターンは検索の絞り込みに使用されません。このパターン内では以下のとおりです。

- '%' は 0 文字以上の任意のサブストリングと一致することを意味します。
- '_' は任意の 1 文字と一致することを意味します。

SCHEMAPATT-LEN (入力)
SCHEMAPATT パラメーターで指定されたスキーマ名の長さを含む、フルワード・バイナリー変数。

PROCNAMEPATT (入力)
プロシージャー名パターンを含む文字フィールド。このフィールドは PROCNAMEPATT-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。PROCNAMEPATT-LEN がゼロの場合、プロシージャー名パターンは検索の絞り込みに使用されません。このパターン内では以下のとおりです。

- '%' は 0 文字以上の任意のサブストリングと一致することを意味します。
- '_' は任意の 1 文字と一致することを意味します。

PROCNAMEPATT-LEN (入力)
PROCNAMEPATT パラメーターで指定されたプロシージャー名パターンの長さを含む、フルワード・バイナリー変数。

COLNAMEPATT (入力)
列名パターンを含む文字フィールド。このフィールドは COLNAMEPATT-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。COLNAMEPATT-LEN がゼロの場合、列名パターンは検索の絞り込みに使用されません。このパターン内では以下のとおりです。

- '%' は 0 文字以上の任意のサブストリングと一致することを意味します。
- '_' は任意の 1 文字と一致することを意味します。

COLNAMEPATT-LEN (入力)
COLNAMEPATT パラメーターで指定された列名パターンの長さを含む、フルワード・バイナリー変数。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

戻されたカーソルは、以下の列を含みます。

1. PROCEDURE_CAT (String): プロシージャー・カタログ (NULL の場合もあります)
2. PROCEDURE_SCHEM (String): プロシージャー・スキーマ (NULL の場合もあります)
3. PROCEDURE_NAME (String): プロシージャー名
4. COLUMN_NAME (String): 列/パラメーター名
5. COLUMN_TYPE (short): 列/パラメーターの種類
   - PROCCOLUMN-UNKNOWN (0) - 不明
   - PROCCOLUMN-IN (1) - IN パラメーター
   - PROCCOLUMN-INOUT (2) - INOUT パラメーター
   - PROCCOLUMN-RESUILT (3) - カーソル内の結果列
   - PROCCOLUMN-OUT (4) - OUT パラメーター
   - PROCCOLUMN-RETURN (5) - プロシージャーの戻り値
6. DATA_TYPE (int): データ・タイプ SQL タイプ
7. TYPE_NAME (String): データ・ソースに依存するタイプ名。UDT の場合、タイプ名は完全修飾です。
8. PRECISION (int): 精度
9. LENGTH (int): バイト単位のデータ長
10. SCALE (short): スケール
11. RADIX (short): 基数
12. NULLABLE (short)): NULL を含めることができるかどうか。
    - NULLABLE-NONULLS (0) - NULL 値を許可しません
    - NULLABLE-NULLABLE (1) - NULL 値を許可します
    - NULLABLE-UNKNOWN (2) - NULL 可能性が不明です
13. REMARKS (String): パラメーター/列に関するコメント

DBPROCEDURES

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

DBPROCEDURES 関数は、特定のカタログで使用可能なストアード・プロシージャーの記述を取得します。これはステートメント・ハンドルを割り振り、STMT-HANDLE パラメーターを設定します。このステートメントは、BINDCOLUMN および FETCH などの関数を使用して戻された情報を検索するのに使用される必要がある、オープン・カーソルを含みます。カーソルは CLOSECURSOR 関数を使用してクローズされなければならず、ステートメントは CLOSESTATEMENT 関数を使用してクローズされなければなりません。スキーマおよびプロシージャー名の基準に一致するプロシージャー記述のみが返されます。これらは、PROCEDURE_SCHEM、PROCEDURE_NAME の順に並べ替えられます。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE            PIC S9(8) BINARY.
   01 CON-HANDLE            PIC S9(8) BINARY.
   01 STMT-HANDLE           PIC S9(8) BINARY.
   01 CATALOG               PIC X(nnn).
   01 CATALOG-LEN           PIC S9(8) BINARY.
   01 SCHEMAPATT            PIC X(nnn).
   01 SCHEMAPATT-LEN        PIC S9(8) BINARY.
   01 PROCNAMEPATT          PIC X(nnn).
   01 PROCNAMEPATT-LEN      PIC S9(8) BINARY.
   01 RETCODE               PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-DBPROCEDURES ENV-HANDLE CON-HANDLE
        STMT-HANDLE CATALOG CATALOG-LEN SCHEMAPATT SCHEMAPATT-LEN
        PROCNAMEPATT PROCNAMEPATT-LEN RETCODE.
```

FUNCTION (入力)
DBPROCEDURES を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

CON-HANDLE (入力)
接続ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (出力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。この関数呼び出しにより設定された STMT-HANDLE 値は、ステートメントを必要とする他の DBCLI 関数への後続の呼び出しで指定される必要があります。

CATALOG (入力)
カタログ名を含む文字フィールド。このフィールドは CATALOG-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。CATALOG がブランクで構成されている場合 (しかし CATALOG-LEN がゼロより大きい場合)、関数はカタログのないものを検索します。
CATALOG-LEN がゼロの場合、カタログ名は検索の絞り込みに使用されません。

CATALOG-LEN (入力)
CATALOG パラメーターで指定されたカタログ名の長さを含む、フルワード・バイナリー変数。

SCHEMAPATT (入力)
スキーマ名パターンを含む文字フィールド。このフィールドは SCHEMAPATT-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。SCHEMAPATT がブランクで構成されている場合 (しかし SCHEMAPATT-LEN がゼロより大きい場合)、関数はスキーマのないものを検索します。 SCHEMAPATT-LEN がゼロの場合、スキーマ名パターンは検索の絞り込みに使用されません。このパターン内では以下のとおりです。

- '%' は 0 文字以上の任意のサブストリングと一致することを意味します。
- '_' は任意の 1 文字と一致することを意味します。

SCHEMAPATT-LEN (入力)
SCHEMAPATT パラメーターで指定されたスキーマ名パターンの長さを含む、フルワード・バイナリー変数。

PROCNAMEPATT (入力)
プロシージャー名パターンを含む文字フィールド。このフィールドは PROCNAMEPATT-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。PROCNAMEPATT-LEN がゼロの場合、プロシージャー名パターンは検索の絞り込みに使用されません。このパターン内では以下のとおりです。

- '%' は 0 文字以上の任意のサブストリングと一致することを意味します。
- '_' は任意の 1 文字と一致することを意味します。

PROCNAMEPATT-LEN (入力)
PROCNAMEPATT パラメーターで指定されたプロシージャー名パターンの長さを含む、フルワード・バイナリー変数。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

戻されたカーソルは、以下の列を含みます。

1. PROCEDURE_CAT (String): プロシージャー・カタログ (NULL の場合もあります)
2. PROCEDURE_SCHEM (String): プロシージャー・スキーマ (NULL の場合もあります)
3. PROCEDURE_NAME (String): プロシージャー名
4. 将来の使用のために予約されています
5. 将来の使用のために予約されています
6. 将来の使用のために予約されています
7. REMARKS (String): プロシージャーに関する説明
8. PROCEDURE_TYPE (short): プロシージャーの種類
   - PROCTYPE-UNKNOWN (0) - 結果を返す場合があります
   - PROCTYPE-NORESULT (1) - 結果を返しません
   - PROCTYPE-RETURNSRESULT (2) - 結果を返します

DBSCHEMAS

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

DBSCHEMAS 関数は、このデータベースで使用可能なスキーマ名のリストを取得します。これはステートメント・ハンドルを割り振り、 STMT-HANDLE パラメーターを設定します。このステートメントは、BINDCOLUMN および FETCH などの関数を使用して戻された情報を検索するのに使用される必要がある、オープン・カーソルを含みます。カーソルは CLOSECURSOR 関数を使用してクローズされなければならず、ステートメントは CLOSESTATEMENT 関数を使用してクローズされなければなりません。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE           PIC S9(8) BINARY.
   01 CON-HANDLE           PIC S9(8) BINARY.
   01 STMT-HANDLE          PIC S9(8) BINARY.
```

```
   01 RETCODE              PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-DBSCHEMAS ENV-HANDLE CON-HANDLE
        STMT-HANDLE RETCODE.
```

FUNCTION (入力)
DBSCHEMAS を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

CON-HANDLE (入力)
接続ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (出力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。この関数呼び出しにより設定された STMT-HANDLE 値は、ステートメントを必要とする他の DBCLI 関数への後続の呼び出しで指定される必要があります。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

戻されたカーソルは、以下の列を含みます。

1. TABLE_SCHEM (ストリング): スキーマ名
2. TABLE_CAT (ストリング): カタログ名 (NULL の場合もあります)

DBSUPERTABLES

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

DBSUPERTABLES 関数は、このデータベース内の特定のスキーマに定義された表階層の記述を取得します。これはステートメント・ハンドルを割り振り、STMT-HANDLE パラメーターを設定します。このステートメントは、BINDCOLUMN および FETCH などの関数を使用して戻された情報を検索するのに使用される必要がある、オープン・カーソルを含みます。カーソルは CLOSECURSOR 関数を使用してクローズされなければならず、ステートメントは CLOSESTATEMENT 関数を使用してクローズされなければなりません。カタログ、スキーマ、および表名と一致する表のスーパー表情報のみが返されます。

表名パラメーターを完全修飾名とすることができます。この場合、カタログとスキーマ・パターンのパラメーターは無視されます。表にスーパー表が含まれない場合、その表はここに表示されません。スーパー表は、副表と同じカタログおよびスキーマに定義される必要があります。したがって、タイプ記述にスーパー表のこの情報を含める必要はありません。ドライバーがタイプ階層をサポートしない場合、空の結果セットが返されます。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE         PIC S9(8) BINARY.
   01 CON-HANDLE         PIC S9(8) BINARY.
   01 STMT-HANDLE        PIC S9(8) BINARY.
   01 CATALOG            PIC X(nnn).
   01 CATALOG-LEN        PIC S9(8) BINARY.
```

```
   01 SCHEMAPATT         PIC X(nnn).
   01 SCHEMAPATT-LEN     PIC S9(8) BINARY.
   01 TABLENAMEPATT      PIC X(nnn).
   01 TABLENAMEPATT-LEN  PIC S9(8) BINARY.
   01 RETCODE            PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-DBSUPERTABLES ENV-HANDLE CON-HANDLE
        STMT-HANDLE CATALOG CATALOG-LEN SCHEMAPATT SCHEMAPATT-LEN
        TABLENAMEPATT TABLENAMEPATT-LEN RETCODE.
```

FUNCTION (入力)
DBSUPERTABLES を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

CON-HANDLE (入力)
接続ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (出力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。この関数呼び出しにより設定された STMT-HANDLE 値は、ステートメントを必要とする他の DBCLI 関数への後続の呼び出しで指定される必要があります。

CATALOG (入力)
カタログ名を含む文字フィールド。このフィールドは CATALOG-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。CATALOG がブランクで構成されている場合 (しかし CATALOG-LEN がゼロより大きい場合)、関数はカタログのないものを検索します。
CATALOG-LEN がゼロの場合、カタログ名は検索の絞り込みに使用されません。

CATALOG-LEN (入力)
CATALOG パラメーターで指定されたカタログ名の長さを含む、フルワード・バイナリー変数。

SCHEMAPATT (入力)
スキーマ名パターンを含む文字フィールド。このフィールドは SCHEMAPATT-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。SCHEMAPATT がブランクで構成されている場合 (しかし SCHEMAPATT-LEN がゼロより大きい場合)、関数はスキーマのないものを検索します。 SCHEMAPATT-LEN がゼロの場合、スキーマ名パターンは検索の絞り込みに使用されません。このパターン内では以下のとおりです。

- '%' は 0 文字以上の任意のサブストリングと一致することを意味します。
- '_' は任意の 1 文字と一致することを意味します。

SCHEMAPATT-LEN (入力)
SCHEMAPATT パラメーターで指定されたスキーマ名パターンの長さを含む、フルワード・バイナリー変数。

TABLENAMEPATT (入力)
テーブル名パターンを含む文字フィールド。このフィールドは TABLENAMEPATT-LEN パラメーターにより指定された長さ以内で、右側にブ

ランクを埋め込む必要があります。TABLENAMEPATT-LEN がゼロの場合、テーブル名パターンは検索の絞り込みに使用されません。このパターン内では以下のとおりです。

- '%' は 0 文字以上の任意のサブストリングと一致することを意味します。
- '_' は任意の 1 文字と一致することを意味します。

TABLENAMEPATT-LEN (入力)
TABLENAMEPATT パラメーターで指定されたテーブル名パターンの長さを含む、フルワード・バイナリー変数。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

戻されたカーソルは、以下の列を含みます。

- TABLE_CAT (String): 表のカタログ (NULL の場合もあります)
- TABLE_SCHEM (String): 表のスキーマ (NULL の場合もあります)
- TABLE_NAME (String): 表名
- SUPERTABLE_NAME (String): 直属のスーパー表の名前

DBSUPERTYPES

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

DBSUPERTYPES 関数は、このデータベース内の特定のスキーマに定義されたユーザー定義タイプ (UDT) 階層の記述を取得します。これはステートメント・ハンドルを割り振り、 STMT-HANDLE パラメーターを設定します。このステートメントは、BINDCOLUMN および FETCH などの関数を使用して戻された情報を検索するのに使用される必要がある、オープン・カーソルを含みます。カーソルは CLOSECURSOR 関数を使用してクローズされなければならず、ステートメントは CLOSESTATEMENT 関数を使用してクローズされなければなりません。直接のスーパータイプ/サブタイプの関係のみがモデル化されます。カタログ、スキーマ、およびタイプ名と一致する UDT のスーパータイプ情報のみが返されます。

タイプ名パラメーターを完全修飾名とすることができます。指定された UDT 名が完全修飾名の場合、カタログとスキーマ・パターンのパラメーターは無視されます。UDT に直接のスーパー・タイプが含まれない場合、その UDT はここに表示されません。この方法により返されたカーソルの行には、指定された UDT および直接のスーパータイプが記述されます。ドライバーがタイプ階層をサポートしない場合、空の結果セットが返されます。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE         PIC S9(8) BINARY.
   01 CON-HANDLE         PIC S9(8) BINARY.
   01 STMT-HANDLE        PIC S9(8) BINARY.
   01 CATALOG            PIC X(nnn).
   01 CATALOG-LEN        PIC S9(8) BINARY.
   01 SCHEMAPATT         PIC X(nnn).
   01 SCHEMAPATT-LEN     PIC S9(8) BINARY.
   01 TYPENAMEPATT       PIC X(nnn).
   01 TYPENAMEPATT-LEN   PIC S9(8) BINARY.
   01 RETCODE            PIC S9(8) BINARY.
```

```
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-DBSUPERTYPES ENV-HANDLE CON-HANDLE
        STMT-HANDLE CATALOG CATALOG-LEN SCHEMAPATT SCHEMAPATT-LEN
        TYPENAMEPATT TYPENAMEPATT-LEN RETCODE.
```

FUNCTION (入力)
DBSUPERTYPES を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

CON-HANDLE (入力)
接続ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (出力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。この関数呼び出しにより設定された STMT-HANDLE 値は、ステートメントを必要とする他の DBCLI 関数への後続の呼び出しで指定される必要があります。

CATALOG (入力)
カタログ名を含む文字フィールド。このフィールドは CATALOG-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。CATALOG がブランクで構成されている場合 (しかし CATALOG-LEN がゼロより大きい場合)、関数はカタログのないものを検索します。
CATALOG-LEN がゼロの場合、カタログ名は検索の絞り込みに使用されません。

CATALOG-LEN (入力)
CATALOG パラメーターで指定されたカタログ名の長さを含む、フルワード・バイナリー変数。

SCHEMAPATT (入力)
スキーマ名パターンを含む文字フィールド。このフィールドは SCHEMAPATT-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。SCHEMAPATT がブランクで構成されている場合 (しかし SCHEMAPATT-LEN がゼロより大きい場合)、関数はスキーマのないものを検索します。 SCHEMAPATT-LEN がゼロの場合、スキーマ名パターンは検索の絞り込みに使用されません。このパターン内では以下のとおりです。

- '%' は 0 文字以上の任意のサブストリングと一致することを意味します。
- '_' は任意の 1 文字と一致することを意味します。

SCHEMAPATT-LEN (入力)
SCHEMAPATT パラメーターで指定されたスキーマ名パターンの長さを含む、フルワード・バイナリー変数。

TYPENAMEPATT (入力)
UDT 名パターンを含む文字フィールド。このフィールドは TYPENAMEPATT-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。TYPENAMEPATT-LEN がゼロの場合、UDT 名パターンは検索の絞り込みに使用されません。このパターン内では以下のとおりです。

- '%' は 0 文字以上の任意のサブストリングと一致することを意味します。
- '_' は任意の 1 文字と一致することを意味します。

TYPENAMEPATT-LEN (入力)
TYPENAMEPATT パラメーターで指定された UDT 名パターンの長さを含む、フルワード・バイナリー変数。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

戻されたカーソルは、以下の列を含みます。

1. TYPE_CAT (String): UDT のカタログ (NULL の場合もあります)
2. TYPE_SCHEM (String): UDT のスキーマ (NULL の場合もあります)
3. TYPE_NAME (String): UDT のタイプ名
4. SUPERTYPE_CAT (String): 直接のスーパータイプのカタログ (NULL の場合もあります)
5. SUPERTYPE_SCHEM (String): 直接のスーパータイプのスキーマ (NULL の場合もあります)
6. SUPERTYPE_NAME (String): 直接のスーパータイプの名前

DBTABLEPRIV

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

DBTABLEPRIV 関数は、カタログ内で使用可能な各表のアクセス権限の記述を取得します。これはステートメント・ハンドルを割り振り、 STMT-HANDLE パラメーターを設定します。このステートメントは、BINDCOLUMN および FETCH などの関数を使用して戻された情報を検索するのに使用される必要がある、オープン・カーソルを含みます。カーソルは CLOSECURSOR 関数を使用してクローズされなければならず、ステートメントは CLOSESTATEMENT 関数を使用してクローズされなければなりません。

表特権は表内の 1 つ以上の列に適用されます。

- この特権がすべての列に適用されると考えるのは間違いです (一部のシステムではこの考えが当てはまる場合がありますが、すべてのシステムに当てはまるとは限りません)。
- スキーマおよび表名の基準に一致する特権のみが返されます。これらは、TABLE_SCHEM、TABLE_NAME、PRIVILEGE の順に並べ替えられます。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE            PIC S9(8) BINARY.
   01 CON-HANDLE            PIC S9(8) BINARY.
   01 STMT-HANDLE           PIC S9(8) BINARY.
   01 CATALOG               PIC X(nnn).
   01 CATALOG-LEN           PIC S9(8) BINARY.
   01 SCHEMAPATT            PIC X(nnn).
   01 SCHEMAPATT-LEN        PIC S9(8) BINARY.
   01 TABLEPATT             PIC X(nnn).
   01 TABLEPATT-LEN         PIC S9(8) BINARY.
   01 RETCODE               PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-DBTABLEPRIV ENV-HANDLE CON-HANDLE
        STMT-HANDLE CATALOG CATALOG-LEN SCHEMAPATT SCHEMAPATT-LEN
        TABLEPATT TABLEPATT-LEN RETCODE.
```

FUNCTION (入力)
DBTABLEPRIV を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

CON-HANDLE (入力)
接続ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (出力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。この関数呼び出しにより設定された STMT-HANDLE 値は、ステートメントを必要とする他の DBCLI 関数への後続の呼び出しで指定される必要があります。

CATALOG (入力)
カタログ名を含む文字フィールド。このフィールドは CATALOG-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。CATALOG がブランクで構成されている場合 (しかし CATALOG-LEN がゼロより大きい場合)、関数はカタログのないものを検索します。
CATALOG-LEN がゼロの場合、カタログ名は検索の絞り込みに使用されません。

CATALOG-LEN (入力)
CATALOG パラメーターで指定されたカタログ名の長さを含む、フルワード・バイナリー変数。

SCHEMAPATT (入力)
スキーマ名パターンを含む文字フィールド。このフィールドは SCHEMAPATT-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。SCHEMAPATT がブランクで構成されている場合 (しかし SCHEMAPATT-LEN がゼロより大きい場合)、関数はスキーマのないものを検索します。 SCHEMAPATT-LEN がゼロの場合、スキーマ名パターンは検索の絞り込みに使用されません。このパターン内では以下のとおりです。

- '%' は 0 文字以上の任意のサブストリングと一致することを意味します。
- '_' は任意の 1 文字と一致することを意味します。

SCHEMAPATT-LEN (入力)
SCHEMA パラメーターで指定されたスキーマ名パターンの長さを含む、フルワード・バイナリー変数。

TABLEPATT (入力)
テーブル名パターンを含む文字フィールド。このフィールドは TABLEPATT-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。TABLEPATT-LEN がゼロの場合、テーブル名パターンは検索の絞り込みに使用されません。このパターン内では以下のとおりです。

- '%' は 0 文字以上の任意のサブストリングと一致することを意味します。
- '_' は任意の 1 文字と一致することを意味します。

TABLEPATT-LEN (入力)
TABLEPATT パラメーターで指定されたテーブル名パターンの長さを含む、フルワード・バイナリー変数。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

戻されたカーソルは、以下の列を含みます。

1. TABLE_CAT (ストリング): テーブル・カタログ (NULL の場合もあります)
2. TABLE_SCHEM (ストリング): テーブル・スキーマ (NULL の場合もあります)
3. TABLE_NAME (ストリング): テーブル名
4. GRANTOR (String): アクセス権限の許可者 (NULL の場合もあります)
5. GRANTEE (String): アクセス権限の被許可者
6. PRIVILEGE (String): アクセス権限の名前
   (SELECT、INSERT、UPDATE、REFRENCES など)
7. IS_GRANTABLE (String): 被許可者が他のユーザーに権限を付与することが許可される場合は「YES」、それ以外の場合は「NO」、不明の場合は NULL

DBTABLES

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

DBTABLES 関数は、特定のカタログ内で使用可能な表の記述を取得します。これはステートメント・ハンドルを割り振り、 STMT-HANDLE パラメーターを設定します。このステートメントは、BINDCOLUMN および FETCH などの関数を使用して戻された情報を検索するのに使用される必要がある、オープン・カーソルを含みます。カーソルは CLOSECURSOR 関数を使用してクローズされなければならず、ステートメントは CLOSESTATEMENT 関数を使用してクローズされなければなりません。

カタログ、スキーマ、表名、およびタイプの基準に一致する表の記述のみが返されます。これらは、TABLE_TYPE、TABLE_SCHEM、TABLE_NAME の順に並べ替えられます。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE         PIC S9(8) BINARY.
   01 CON-HANDLE         PIC S9(8) BINARY.
   01 STMT-HANDLE        PIC S9(8) BINARY.
   01 CATALOG            PIC X(nnn).
   01 CATALOG-LEN        PIC S9(8) BINARY.
   01 SCHEMAPATT         PIC X(nnn).
   01 SCHEMAPATT-LEN     PIC S9(8) BINARY.
   01 TABLEPATT          PIC X(nnn).
   01 TABLEPATT-LEN      PIC S9(8) BINARY.
   01 TYPES              PIC X(nnn).
   01 TYPES-LEN          PIC S9(8) BINARY.
   01 RETCODE            PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-DBTABLES ENV-HANDLE CON-HANDLE
        STMT-HANDLE CATALOG CATALOG-LEN SCHEMAPATT SCHEMAPATT-LEN
        TABLEPATT TABLEPATT-LEN TYPES TYPES-LEN RETCODE.
```

FUNCTION (入力)
DBTABLES を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

CON-HANDLE (入力)
接続ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (出力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。この関数呼び出しにより設定された STMT-HANDLE 値は、ステートメントを必要とする他の DBCLI 関数への後続の呼び出しで指定される必要があります。

CATALOG (入力)
カタログ名を含む文字フィールド。このフィールドは CATALOG-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。CATALOG がブランクで構成されている場合 (しかし CATALOG-LEN がゼロより大きい場合)、関数はカタログのないものを検索します。
CATALOG-LEN がゼロの場合、カタログ名は検索の絞り込みに使用されません。

CATALOG-LEN (入力)
CATALOG パラメーターで指定されたカタログ名の長さを含む、フルワード・バイナリー変数。

SCHEMAPATT (入力)
スキーマ名パターンを含む文字フィールド。このフィールドは SCHEMAPATT-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。SCHEMAPATT がブランクで構成されている場合 (しかし SCHEMAPATT-LEN がゼロより大きい場合)、関数はスキーマのないものを検索します。 SCHEMAPATT-LEN がゼロの場合、スキーマ名パターンは検索の絞り込みに使用されません。このパターン内では以下のとおりです。

- '%' は 0 文字以上の任意のサブストリングと一致することを意味します。
- '_' は任意の 1 文字と一致することを意味します。

SCHEMAPATT-LEN (入力)
SCHEMAPATT パラメーターで指定されたスキーマ名パターンの長さを含む、フルワード・バイナリー変数。

TABLEPATT (入力)
テーブル名パターンを含む文字フィールド。このフィールドは TABLEPATT-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。TABLEPATT-LEN がゼロの場合、テーブル名パターンは検索の絞り込みに使用されません。このパターン内では以下のとおりです。

- '%' は 0 文字以上の任意のサブストリングと一致することを意味します。
- '_' は任意の 1 文字と一致することを意味します。

TABLEPATT-LEN (入力)
TABLEPATT パラメーターで指定されたテーブル名パターンの長さを含む、フルワード・バイナリー変数。

TYPES (入力)
セミコロンで区切られたタイプ名のリストを含む文字フィールド。このフィール

ドは TYPES-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。TYPES-LEN がゼロの場合、すべてのタイプが含まれます。

TYPES-LEN (入力)
TYPES パラメーターで指定されたタイプ・リストの長さを含む、フルワード・バイナリー変数。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

戻されたカーソルは、以下の列を含みます。

1. TABLE_CAT (ストリング): テーブル・カタログ (NULL の場合もあります)
2. TABLE_SCHEM (ストリング): テーブル・スキーマ (NULL の場合もあります)
3. TABLE_NAME (ストリング): テーブル名
4. TABLE_TYPE (String): 表タイプ。標準的なタイプは、「TABLE」、「VIEW」、「SYSTEM TABLE」、「GLOBAL TEMPORARY」、「LOCAL TEMPORARY」、「ALIAS」、「SYNONYM」です。
5. REMARKS (String): 表に関する説明
6. TYPE_CAT (String): タイプ・カタログ (NULL の場合もあります)
7. TYPE_SCHEM (String): タイプ・スキーマ (NULL の場合もあります)
8. TYPE_NAME (String): タイプ名 (NULL の場合もあります)
9. SELF_REFERENCING_COL_NAME (String): 型付き表の指定された「identifier」列の名前 (NULL の場合もあります)
10. REF_GENERATION (String): SELF_REFERENCING_COL_NAME の値の作成方法を示します。値は、「SYSTEM」、「USER」、「DERIVED」です。(NULL の場合もあります。)

DBTABLETYPES

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

DBTABLETYPES 関数は、使用可能な表タイプのリストを取得します。これはステートメント・ハンドルを割り振り、 STMT-HANDLE パラメーターを設定します。このステートメントは、BINDCOLUMN および FETCH などの関数を使用して戻された情報を検索するのに使用される必要がある、オープン・カーソルを含みます。カーソルは CLOSECURSOR 関数を使用してクローズされなければならず、ステートメントは CLOSESTATEMENT 関数を使用してクローズされなければなりません。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE            PIC S9(8) BINARY.
   01 CON-HANDLE            PIC S9(8) BINARY.
   01 STMT-HANDLE           PIC S9(8) BINARY.
   01 RETCODE               PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-DBTABLETYPES ENV-HANDLE CON-HANDLE
        STMT-HANDLE RETCODE.
```

FUNCTION (入力)
DBTABLETYPES を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

CON-HANDLE (入力)
接続ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (出力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。この関数呼び出しにより設定された STMT-HANDLE 値は、ステートメントを必要とする他の DBCLI 関数への後続の呼び出しで指定される必要があります。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

戻されたカーソルは、以下の列を含みます。

- TABLE_TYPE (String): 表タイプ。標準的なタイプは、「TABLE」、「VIEW」、「SYSTEM TABLE」、「GLOBAL TEMPORARY」、「LOCAL TEMPORARY」、「ALIAS」、「SYNONYM」です。

DBTYPEINFO

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

DBTYPEINFO 関数は、このデータベースでサポートされるすべての標準 SQL タイプの記述を取得します。これはステートメント・ハンドルを割り振り、STMT-HANDLE パラメーターを設定します。このステートメントは、BINDCOLUMN および FETCH などの関数を使用して戻された情報を検索するのに使用される必要がある、オープン・カーソルを含みます。カーソルは CLOSECURSOR 関数を使用してクローズされなければならず、ステートメントは CLOSESTATEMENT 関数を使用してクローズされなければなりません。返されるタイプは、DATA_TYPE、「対応する JDBC SQL タイプにデータ・タイプがどれだけ密接にマップしているか」の順で並べ替えられます。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE         PIC S9(8) BINARY.
   01 CON-HANDLE         PIC S9(8) BINARY.
   01 STMT-HANDLE        PIC S9(8) BINARY.
   01 RETCODE            PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-DBTYPEINFO ENV-HANDLE CON-HANDLE
        STMT-HANDLE RETCODE.
```

FUNCTION (入力)
DBTYPEINFO を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

CON-HANDLE (入力)
接続ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (出力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。この関数呼び出しにより設定された STMT-HANDLE 値は、ステートメントを必要とする他の DBCLI 関数への後続の呼び出しで指定される必要があります。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

戻されたカーソルは、以下の列を含みます。

1. TYPE_NAME (String): タイプ名
2. DATA_TYPE (int): SQL データ・タイプ
3. PRECISION (int): 最大精度
4. LITERAL_PREFIX (String): リテラルの引用に使用される接頭部 (NULL の場合もあります)
5. LITERAL_SUFFIX (String): リテラルの引用に使用される接尾部 (NULL の場合もあります)
6. CREATE_PARAMS (String): タイプの作成に使用されるパラメーター (NULL の場合もあります)
7. NULLABLE (short): このタイプに NULL を使用できるかどうか。
   - NULLABLE-NONULLS (0) - NULL 値を許可しません
   - NULLABLE-NULLABLE (1) - NULL 値を許可します
   - NULLABLE-UNKNOWN (2) - NULL 可能性が不明です
8. CASE_SENSITIVE (Boolean): 大/小文字を区別するかどうか。
9. SEARCHABLE (short): このタイプに基づいて「WHERE」を使用できるかどうか。
   - TYPEPRED-NONE (0) - サポートされません
   - TYPEPRED-CHAR (1) - WHERE .. LIKE でのみサポートされます
   - TYPEPRED-BASIC (2) - WHERE .. LIKE 以外でサポートされます
   - TYPEPRED-SEARCHABLE (3) - すべての WHERE .. でサポートされます
10. UNSIGNED_ATTRIBUTE (Boolean): 符号なしかどうか。
11. FIXED_PREC_SCALE (Boolean): 金額値にできるかどうか。
12. AUTO_INCREMENT (Boolean): 自動増分値として使用できるかどうか。
13. LOCAL_TYPE_NAME (String): タイプ名のローカライズ・バージョン (NULL の場合もあります)
14. MINIMUM_SCALE (short): サポートされる最小スケール
15. MAXIMUM_SCALE (short): サポートされる最大スケール
16. SQL_DATA_TYPE (int): 未使用
17. SQL_DATETIME_SUB (int): 未使用
18. NUM_PREC_RADIX (int): 通常、2 または 10

DBUDTS

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

DBUDTS 関数は、特定のスキーマに定義されたユーザー定義タイプ (UDT) の記述を取得します。これはステートメント・ハンドルを割り振り、 STMT-HANDLE パラメーターを設定します。このステートメントは、BINDCOLUMN および FETCH などの関数を使用して戻された情報を検索するのに使用される必要がある、オープン・カーソルを含みます。カーソルは CLOSECURSOR 関数を使用してクローズされなければならず、ステートメントは CLOSESTATEMENT 関数を使用してクローズされなければなりません。

スキーマ固有の UDT にはタイプ JAVA_OBJECT、STRUCT、または DISTINCT が含まれている可能性があります。カタログ、スキーマ、タイプ名、およびタイプの基準に一致するタイプのみが返されます。これらは、DATA_TYPE、TYPE_SCHEM、TYPE_NAME の順に並べ替えられます。タイプ名パラメーターを完全修飾名とすることができます。この場合、カタログとスキーマ・パターンのパラメーターは無視されます。ドライバーが UDT をサポートしていない場合、空の結果セットが返されます。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE         PIC S9(8) BINARY.
   01 CON-HANDLE         PIC S9(8) BINARY.
   01 STMT-HANDLE        PIC S9(8) BINARY.
   01 CATALOG            PIC X(nnn).
   01 CATALOG-LEN        PIC S9(8) BINARY.
   01 SCHEMAPATT         PIC X(nnn).
   01 SCHEMAPATT-LEN     PIC S9(8) BINARY.
   01 TYPEPATT           PIC X(nnn).
   01 TYPEPATT-LEN       PIC S9(8) BINARY.
   01 TYPES              PIC X(nnn).
   01 TYPES-LEN          PIC S9(8) BINARY.
   01 RETCODE            PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-DBUDTS ENV-HANDLE CON-HANDLE
        STMT-HANDLE CATALOG CATALOG-LEN SCHEMAPATT SCHEMAPATT-LEN
        TYPEPATT TYPEPATT-LEN TYPES TYPES-LEN RETCODE.
```

FUNCTION (入力)
DBUDTS を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

CON-HANDLE (入力)
接続ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (出力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。この関数呼び出しにより設定された STMT-HANDLE 値は、ステートメントを必要とする他の DBCLI 関数への後続の呼び出しで指定される必要があります。

CATALOG (入力)
カタログ名を含む文字フィールド。このフィールドは CATALOG-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要がありま

す。CATALOG がブランクで構成されている場合 (しかし CATALOG-LEN がゼロより大きい場合)、関数はカタログのないものを検索します。
CATALOG-LEN がゼロの場合、カタログ名は検索の絞り込みに使用されません。

CATALOG-LEN (入力)
CATALOG パラメーターで指定されたカタログ名の長さを含む、フルワード・バイナリー変数。

SCHEMAPATT (入力)
スキーマ名パターンを含む文字フィールド。このフィールドは SCHEMAPATT-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。SCHEMAPATT がブランクで構成されている場合 (しかし SCHEMAPATT-LEN がゼロより大きい場合)、関数はスキーマのないものを検索します。 SCHEMAPATT-LEN がゼロの場合、スキーマ名パターンは検索の絞り込みに使用されません。このパターン内では以下のとおりです。

- '%' は 0 文字以上の任意のサブストリングと一致することを意味します。
- '_' は任意の 1 文字と一致することを意味します。

SCHEMAPATT-LEN (入力)
SCHEMAPATT パラメーターで指定されたスキーマ名パターンの長さを含む、フルワード・バイナリー変数。

TYPEPATT (入力)
タイプ名パターンを含む文字フィールド。このフィールドは TYPEPATT-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。TYPEPATT-LEN がゼロの場合、タイプ名パターンは検索の絞り込みに使用されません。このパターン内では以下のとおりです。

- '%' は 0 文字以上の任意のサブストリングと一致することを意味します。
- '_' は任意の 1 文字と一致することを意味します。

TYPEPATT-LEN (入力)
TYPEPATT パラメーターで指定されたタイプ名パターンの長さを含む、フルワード・バイナリー変数。

TYPES (入力)
セミコロンで区切られたタイプ名のリストを含む文字フィールド。このフィールドは TYPES-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。TYPES-LEN がゼロの場合、すべてのタイプが含まれます。

TYPES-LEN (入力)
TYPES パラメーターで指定されたタイプ・リストの長さを含む、フルワード・バイナリー変数。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。
戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

戻されたカーソルは、以下の列を含みます。

1. TYPE_CAT (String): タイプのカタログ (NULL の場合もあります)
2. TYPE_SCHEM (String): タイプのスキーマ (NULL の場合もあります)

3. TYPE_NAME (String): タイプ名
4. CLASS_NAME (String): Java クラス名
5. DATA_TYPE (int): SQL タイプ値。JAVA_OBJECT、STRUCT、または DISTINCT のいずれか
6. REMARKS (String): タイプに関する説明
7. BASE_TYPE (short): DISTINCT タイプのソース・タイプのタイプ・コード、または SQL タイプに定義されている構造化タイプのユーザー生成参照タイプ SELF_REFERENCING_COLUMN を実装するタイプのソース・タイプのタイプ・コード (DATA_TYPE が DISTINCT でない場合や、REFERENCE_GENERATION = USER_DEFINED と指定された STRUCT でない場合は NULL)

DBVERSIONCOLS

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

DBVERSIONCOLS 関数は、列内の任意の値が更新された場合に自動で更新される表の列の記述を取得します。これはステートメント・ハンドルを割り振り、STMT-HANDLE パラメーターを設定します。このステートメントは、BINDCOLUMN および FETCH などの関数を使用して戻された情報を検索するのに使用される必要がある、オープン・カーソルを含みます。カーソルは CLOSECURSOR 関数を使用してクローズされなければならず、ステートメントは CLOSESTATEMENT 関数を使用してクローズされなければなりません。スキーマ、プロシージャー、およびパラメーター名の基準に一致する記述のみが返されます。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE         PIC S9(8) BINARY.
   01 CON-HANDLE         PIC S9(8) BINARY.
   01 STMT-HANDLE        PIC S9(8) BINARY.
   01 CATALOG            PIC X(nnn).
   01 CATALOG-LEN        PIC S9(8) BINARY.
   01 SCHEMA             PIC X(nnn).
   01 SCHEMA-LEN         PIC S9(8) BINARY.
   01 TABLENAME          PIC X(nnn).
   01 TABLENAME-LEN      PIC S9(8) BINARY.
   01 RETCODE            PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-DBVERSIONCOLS ENV-HANDLE CON-HANDLE
        STMT-HANDLE CATALOG CATALOG-LEN SCHEMA SCHEMA-LEN
        TABLENAME TABLENAME-LEN RETCODE.
```

FUNCTION (入力)
DBVERSIONCOLS を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

CON-HANDLE (入力)
接続ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (出力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。この関数呼び出

しにより設定された STMT-HANDLE 値は、ステートメントを必要とする他の DBCLI 関数への後続の呼び出しで指定される必要があります。

CATALOG (入力)
カタログ名を含む文字フィールド。このフィールドは CATALOG-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。CATALOG がブランクで構成されている場合 (しかし CATALOG-LEN がゼロより大きい場合)、関数はカタログのないものを検索します。
CATALOG-LEN がゼロの場合、カタログ名は検索の絞り込みに使用されません。

CATALOG-LEN (入力)
CATALOG パラメーターで指定されたカタログ名の長さを含む、フルワード・バイナリー変数。

SCHEMA (入力)
スキーマ名を含む文字フィールド。このフィールドは SCHEMA-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。SCHEMA がブランクで構成されている場合 (しかし SCHEMA-LEN がゼロより大きい場合)、関数はスキーマのないものを検索します。 SCHEMA-LEN がゼロの場合、スキーマ名は検索の絞り込みに使用されません。

SCHEMA-LEN (入力)
SCHEMA パラメーターで指定されたスキーマ名の長さを含む、フルワード・バイナリー変数。

TABLENAME (入力)
テーブル名を含む文字フィールド。このフィールドは TABLENAME-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。

TABLENAME-LEN (入力)
TABLENAME パラメーターで指定されたテーブル名の長さを含む、フルワード・バイナリー変数。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

戻されたカーソルは、以下の列を含みます。

1. SCOPE (short): 未使用
2. COLUMN_NAME (ストリング): 列名
3. DATA_TYPE (int): java.sql.Types からの SQL データ・タイプ
4. TYPE_NAME (String): データ・ソースに依存するタイプ名
5. COLUMN_SIZE (int): 精度
6. BUFFER_LENGTH (int): バイト単位の列値の長さ
7. DECIMAL_DIGITS (short): スケール
8. PSEUDO_COLUMN (short): これが Oracle ROWID のような疑似列かどうか
   - VERSIONCOL-UNKNOWN (0) - 疑似列かどうか不明
   - VERSIONCOL-NOTPSEUDO (1) - 疑似列ではない

- VERSIONCOL-PSEUDO (2) - 疑似列である

DISCONNECT

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

DISCONNECT 関数は、DBCliServer への接続を閉じるとともに、データベースからの切断も行います。また、接続を表す接続ハンドルを解放します。DISCONNECT 関数への呼び出しが成功すると、接続ハンドルは有効でなくなります。

関連する関数:

- 343 ページの『CONNECT』
- 344 ページの『CONNECTSSL』

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE          PIC S9(8) BINARY.
   01 CON-HANDLE          PIC S9(8) BINARY.
   01 RETCODE             PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-DISCONNECT ENV-HANDLE CON-HANDLE
        RETCODE.
```

FUNCTION (入力)
DISCONNECT を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

CON-HANDLE (入力)
接続ハンドルを含むフルワード・バイナリー変数。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

EXECUTE

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

EXECUTE 関数はステートメントを実行します。入力パラメーターにバインドされたホスト変数の内容は、データベースに送信され、ステートメント内のパラメーターを置き換えます。

- 更新ステートメントであった場合は、この関数によって更新カウントが提供されます。また、後で GETUPDATECOUNT 関数を使用して更新カウントを取得することもできます。
- カーソルを返す照会ステートメントであった場合は、FETCH 関数を使用して結果行を取り出すことができます。

1 つのステートメントで複数の結果を提供することができます。追加の結果を取得するには、GETMORERESULTS 関数を使用する必要があります。ストアード・プロシージャーから出力変数を取得するには、結果を取得できなくなるまで GETMORERESULTS

関数を呼び出す必要があります。これにより、出力パラメーターにバインドされたホスト変数がストアード・プロシージャーによって値セットに設定されます。ステートメントは、(異なる入力パラメーター値を使用して) いつでも再実行できます。再実行では、最後の実行から開かれたままのカーソルがあれば、自動的に閉じられます。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE           PIC S9(8) BINARY.
   01 STMT-HANDLE          PIC S9(8) BINARY.
   01 UPDATE-COUNT         PIC S9(8) BINARY.
   01 RETCODE              PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-EXECUTE ENV-HANDLE
        STMT-HANDLE [UPDATE-COUNT] RETCODE.
```

FUNCTION (入力)
EXECUTE を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (入力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。

UPDATE-COUNT (出力、オプション)
ステートメントの実行によって更新された行数に設定されるフルワード・バイナリー変数。更新カウントがない場合は -1 に設定されます。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

FETCH

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

FETCH 関数を使用すると、アプリケーションでオープン・カーソルから 1 行のデータを取得できます。取り出し操作によっては、FETCH 関数でカーソルを必要な行に配置できます。FETCH 関数では、BINDCOLUMN 関数を使用してカーソルにバインドされたすべてのホスト変数が設定されます。また、この関数では、列が NULL かどうかや、列が更新、削除、または挿入されたかどうかを示す、対応する標識変数も設定されます。ステートメントが実行されると、カーソルは 1 行目の前に配置されます。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE            PIC S9(8) BINARY.
   01 STMT-HANDLE           PIC S9(8) BINARY.
   01 OPERATION             PIC S9(8) BINARY.
   01 OFFSET                PIC S9(8) BINARY.
   01 RETCODE               PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-FETCH ENV-HANDLE
        STMT-HANDLE [OPERATION [OFFSET]] RETCODE.
```

FUNCTION (入力)
FETCH を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (入力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。

OPERATION (入力、オプション)
命令コードを含むフルワード・バイナリー変数。OPERATION パラメーターが省略された場合、デフォルトの FETCH-NEXT 操作が行われます。以下の操作がサポートされます。

FETCH-NEXT (0)
次の行にカーソルを配置し、データを取り出します。これがデフォルトの操作です。

FETCH-PREVIOUS (1)
前の行にカーソルを配置し、データを取り出します。この操作はカーソルがスクロール可能である場合にのみ使用可能であり、それはステートメントが CURSOR-TYPE-SCROLL-INSENSITIVE または CURSOR-TYPE-SCROLL-SENSITIVE で作成された場合です。

FETCH-FIRST (2)
結果の 1 行目にカーソルを配置し、データを取り出します。この操作はカーソルがスクロール可能である場合にのみ使用可能であり、それはステートメントが CURSOR-TYPE-SCROLL-INSENSITIVE または CURSOR-TYPE-SCROLL-SENSITIVE で作成された場合です。

FETCH-LAST (3)
結果の最終行にカーソルを配置し、データを取り出します。この操作はカーソルがスクロール可能である場合にのみ使用可能であり、それはステートメントが CURSOR-TYPE-SCROLL-INSENSITIVE または CURSOR-TYPE-SCROLL-SENSITIVE で作成された場合です。

FETCH-ABSOLUTE (4)
OFFSET パラメーターで指定された行にカーソルを配置します。行番号がゼロの場合、カーソルは 1 行目の前に配置されます。負の行番号では、カーソルは結果の最終行に配置されます。この操作はカーソルがスクロール可能である場合にのみ使用可能であり、それはステートメントが CURSOR-TYPE-SCROLL-INSENSITIVE または CURSOR-TYPE-SCROLL-SENSITIVE で作成された場合です。

FETCH-RELATIVE (5)
OFFSET パラメーターに指定された行数だけカーソルを移動します。負の OFFSET では、カーソルは結果の先頭に向けて移動し、正の OFFSET では、カーソルは結果の最後尾に向けて移動します。OFFSET がゼロの場合、カーソルの位置はそのままですが、行がデータベースから再度取り出されます。この操作はカーソルがスクロール可能である場合にのみ使用可能であり、それはステートメントが CURSOR-TYPE-SCROLL-INSENSITIVE または CURSOR-TYPE-SCROLL-SENSITIVE で作成された場合です。

OFFSET (入力、オプション)
OPERATION パラメーターに応じて絶対行番号または相対行番号のどちらかを含むフルワード・バイナリー変数。OFFSET パラメーターが省略された場合、オフセットはデフォルトのゼロになります。FETCH-ABSOLUTE および FETCH-RELATIVE では OFFSET パラメーターを指定する必要があります。他のすべての操作では、OFFSET パラメーターは無視されます。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

GETCOLUMNINFO

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

GETCOLUMNINFO 関数を使用すると、ステートメントが実行された後で、カーソルに含まれる列の属性をアプリケーションで照会できます。その後、アプリケーションで、BINDCOLUMN 関数を使用してホスト変数を列にバインドできます。

関連する関数:
- 334 ページの『BINDCOLUMN』
- 417 ページの『GETNUMCOLUMNS』

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE            PIC S9(8) BINARY.
   01 STMT-HANDLE           PIC S9(8) BINARY.
   01 COL-INDEX             PIC S9(8) BINARY.
   01 NAME                  PIC X(nnn).
   01 NAME-LEN              PIC S9(8) BINARY.
   01 SQL-TYPE              PIC S9(8) BINARY.
   01 PRECISION             PIC S9(8) BINARY.
   01 SCALE                 PIC S9(8) BINARY.
   01 SIGNED                PIC S9(8) BINARY.
   01 NULLABLE              PIC S9(8) BINARY.
   01 RETCODE               PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-GETCOLUMNINFO ENV-HANDLE
        STMT-HANDLE COL-INDEX NAME NAME-LEN SQL-TYPE
        [PRECISION [SCALE [SIGNED [NULLABLE]]]]
        RETCODE.
```

FUNCTION (入力)
GETCOLUMNINFO を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (入力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。

COL-INDEX (入力)
情報を取得する列の番号を含むフルワード・バイナリー変数。列番号は、1 から始まります。

NAME (出力)
列の名前に設定される文字フィールド。このフィールドの長さは NAME-LEN パラメーターを使用して設定されます。指定された長さよりも列名が長い場合、列名は警告なしで切り捨てられます。

NAME-LEN (入力)
NAME フィールドの長さを含むフルワード・バイナリー変数。

SQL-TYPE (出力)
列の SQL-TYPE に設定されるフルワード・バイナリー変数。可能な値については、コピーブックの SQL-TYPE 宣言を参照してください。

PRECISION (出力、オプション)
列の精度に設定されるフルワード・バイナリー変数。このパラメーターはオプションです。

SCALE (出力、オプション)
列のスケールに設定されるフルワード・バイナリー変数。このパラメーターはオプションです。これを指定する場合、PRECISION パラメーターも指定する必要があります。

SIGNED (出力、オプション)
列が符号なしの場合に 0 に設定されるフルワード・バイナリー変数。ステートメント・パラメーターが符号ありの場合は 1 に設定されます。このパラメーターはオプションです。これを指定する場合、PRECISION および SCALE パラメーターも指定する必要があります。

NULLABLE (出力、オプション)
列に NULL を含めることができない場合に 0 に設定されるフルワード・バイナリー変数。ステートメント・パラメーターに NULL を含めることができる場合は 1 です。このパラメーターはオプションです。これを指定する場合、PRECISION、SCALE、および SIGNED パラメーターも指定する必要があります。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

GETCONNATTR

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

GETCONNATTR 関数を使用すると、アプリケーション接続レベルの属性を取得できます。

関連する関数: 431 ページの『SETCONNATTR』。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE           PIC S9(8) BINARY.
   01 CON-HANDLE           PIC S9(8) BINARY.
   01 ATTR                 PIC S9(8) BINARY.
   01 VALUE                PIC S9(8) BINARY.
   01 VALUE-LEN            PIC S9(8) BINARY.
```

```
   01 RETCODE              PIC S9(8) BINARY.
PROCEDURE
  CALL 'IESDBCLI' USING FUNC-GETCONNATTR ENV-HANDLE CON-HANDLE
       ATTR VALUE VALUE-LEN RETCODE.
```

FUNCTION (入力)
GETCONNATTR を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

CON-HANDLE (入力)
接続ハンドルを含むフルワード・バイナリー変数。

ATTR (入力)
設定または取得する属性の属性 ID を含むフルワード・バイナリー変数。

VALUE (入出力)
属性値を受け取るフィールド。一部の属性はテキスト値を使用するので、代わりに文字フィールドを使用する必要があります。

VALUE-LEN (入力)
VALUE フィールドの長さを含むフルワード・バイナリー変数。フルワード・バイナリー値を使用する属性では、長さは 4 に設定される必要があります。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

以下の接続属性が定義されます。

CONNATTR-TRANSACTION-ISOLATION (1)
この属性は、この接続で使用される、トランザクションの分離レベルを示します。値は以下のいずれかを含むフルワード・バイナリー変数です。

TRANSACTION-NONE (0):
トランザクションがサポートされていません。

TRANSACTION-READ-UNCOMMITTED (1):
ダーティー読み取り、反復不能読み取り、および幻像読み取りが実行される可能性があります。

TRANSACTION-READ-COMMITTED (2):
ダーティー読み取りが妨げられます。反復不能読み取りおよび幻像読み取りが実行される可能性があります。

TRANSACTION-REPEATABLE-READ (4):
ダーティー読み取りおよび反復不能読み取りが妨げられます。幻像読み取りが実行される可能性があります。

TRANSACTION-SERIALIZABLE (8):
ダーティー読み取り、反復不能読み取り、および幻像読み取りが妨げられます。

CONNATTR-READONLY (2)
この属性は、接続が読み取り専用モードで動作するかどうかを示します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

CONNATTR-HOLDABILITY (3)
この属性は、この接続を使用してステートメントを作成する際にデフォルトの保持可能性モードを使用するかどうかを指定します。値は以下のいずれかを含むフルワード・バイナリー変数です。

HOLD-CURSORS-OVER-COMMIT (1):
COMMIT が実行されても、オープン・カーソルは開いたままです。

CLOSE-CURSORS-AT-COMMIT (2):
COMMIT が実行されると、すべてのオープン・カーソルが閉じられます。

CONNATTR-CATALOG (4)
この属性はデータベース・カタログを示します。値は VALUE-LEN パラメーターにより長さが決められる文字フィールドです。

CONNATTR-AUTO-COMMIT (5)
この属性は、この接続の自動コミット・モードを制御します。接続が自動コミット・モードである場合、そのすべての SQL ステートメントは個々のトランザクションとして実行され、コミットされます。それ以外の場合、その SQL ステートメントはいくつかのトランザクションにグループ分けされます。これらのトランザクションは、COMMIT 関数または ROLLBACK 関数の呼び出しによって終了します。デフォルトでは、新しい接続は自動コミット・モードではありません。値は、FALSE (自動コミットなし) には 0、TRUE (自動コミットが有効) には 1 を含む、フルワード・バイナリー変数です。

CONNATTR-STATNUMSTMTS (6)
この読み取り専用属性を使用して、この接続で作成されたステートメントの総数を取得できます。この属性を指定した SETCONNATTR 呼び出しによって、カウンターがゼロにリセットされます。

CONNATTR-STATNUMEXECS (7)
この読み取り専用属性を使用して、この接続 (のすべてのステートメント) で実行されたステートメントの総実行数を取得できます。この属性を指定した SETCONNATTR 呼び出しによって、カウンターがゼロにリセットされます。

CONNATTR-STATNUMROWS (8)
この読み取り専用属性を使用して、この接続 (のすべてのステートメント) で取り出された行の総数を取得できます。この属性を指定した SETCONNATTR 呼び出しによって、カウンターがゼロにリセットされます。

CONNATTR-SERVER-TRACE (9)
この属性を使用して、DBCliServer 側でのトレースのサポートを有効または無効にできます。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

CONNATTR-SERVER-TRACE-LEVEL (10)
この属性を使用して、DBCliServer 側でのトレース・サポートのトレース・レベルを取得または設定できます。値は、以下のトレース・レベルを含むフルワード・バイナリー変数です。

TRACELEVEL-FLOW (4)
アプリケーションのメイン・フローをトレースします

TRACELEVEL-NORMAL (5)
ほとんどのメッセージをトレースします

TRACELEVEL-FINE (6)
最も広範囲に及ぶトレース

データベースのメタ情報を取得するには、読み取り専用の以下の接続属性を定義します。

DBATTR-CATALOGSEPARATOR (16)
このデータベースでカタログと表名の間の分離記号として使用されるストリングを取得します。値は VALUE-LEN パラメーターにより長さが決められる文字フィールドです。

DBATTR-CATALOGTERM (17)
データベース・ベンダーで使用されている、「カタログ」を意味する推奨用語を取得します。値は VALUE-LEN パラメーターにより長さが決められる文字フィールドです。

DBATTR-DATABASEPRODUCTNAME (18)
この接続の接続先であるデータベース製品の名前を取得します。値は VALUE-LEN パラメーターにより長さが決められる文字フィールドです。

DBATTR-DATABASEPRODUCTVERSION (19)
この接続の接続先であるデータベース製品のバージョン番号を取得します。値は VALUE-LEN パラメーターにより長さが決められる文字フィールドです。

DBATTR-DRIVERNAME (20)
この接続をデータベースに接続するために使用されている JDBC ドライバーの名前を取得します。値は VALUE-LEN パラメーターにより長さが決められる文字フィールドです。

DBATTR-DRIVERVERSION (21)
この接続をデータベースに接続するために使用されている JDBC ドライバーのバージョン番号を取得します。値は VALUE-LEN パラメーターにより長さが決められる文字フィールドです。

DBATTR-EXTRANAMECHARACTERS (22)
引用符で囲まれていない識別子名内で使用されている可能性がある「余分な」文字をすべて取得します (a から z、A から Z、0 から 9、および _ 以外の文字)。値は VALUE-LEN パラメーターにより長さが決められる文字フィールドです。

DBATTR-IDENTIFIERQUOTESTRING (23)
SQL 識別子の引用符に使用されているストリングを取得します。識別子を引用符で囲むことがサポートされていない場合、この属性はスペース「 」を返します。値は VALUE-LEN パラメーターにより長さが決められる文字フィールドです。値は VALUE-LEN パラメーターにより長さが決められる文字フィールドです。

DBATTR-NUMERICFUNCTIONS (24)
この接続の接続先であるデータベースで使用可能な数学関数のコンマ区切りリストを取得します。これらは、JDBC 関数のエスケープ文節で使用される Open /Open CLI 数学関数名です。値は VALUE-LEN パラメーターにより長さが決められる文字フィールドです。

DBATTR-PROCEDURETERM (25)
データベース・ベンダーで使用されている、「プロシージャー」を意味する推奨用語を取得します。値は VALUE-LEN パラメーターにより長さが決められる文字フィールドです。

DBATTR-SQLKEYWORDS (26)
データベースのすべての SQL キーワードのうち、SQL 92 キーワードでは「ない」キーワードのコンマ区切りリストを取得します。値は VALUE-LEN パラメーターにより長さが決められる文字フィールドです。

DBATTR-SCHEMATERM (27)
データベース・ベンダーで使用されている、「スキーマ」を意味する推奨用語を取得します。値は VALUE-LEN パラメーターにより長さが決められる文字フィールドです。

DBATTR-SEARCHSTRINGESCAPE (28)
ワイルドカード文字をエスケープするために使用できるストリングを取得します。これは、パターンである (したがって、ワイルドカード文字の 1 つが使用されている) カタログ検索パラメーター内の「_」または「%」をエスケープするのに使用できるストリングです。文字「_」は任意の 1 文字を表します。文字「%」はゼロ文字以上の任意のシーケンスを表します。値は VALUE-LEN パラメーターにより長さが決められる文字フィールドです。

DBATTR-STRINGFUNCTIONS (29)
データベースで使用可能なストリング関数のコンマ区切りリストを取得します。これらは、JDBC 関数のエスケープ文節で使用される Open Group CLI ストリング関数名です。値は VALUE-LEN パラメーターにより長さが決められる文字フィールドです。

DBATTR-SYSTEMFUNCTIONS (30)
データベースで使用可能なシステム関数のコンマ区切りリストを取得します。これらは、JDBC 関数のエスケープ文節で使用される Open Group CLI システム関数名です。値は VALUE-LEN パラメーターにより長さが決められる文字フィールドです。

DBATTR-TIMEDATEFUNCTIONS (31)
このデータベースで使用可能な時間関数と日付関数のコンマ区切りリストを取得します。値は VALUE-LEN パラメーターにより長さが決められる文字フィールドです。

DBATTR-URL (32)
この DBMS の URL を取得します。値は VALUE-LEN パラメーターにより長さが決められる文字フィールドです。

DBATTR-USERNAME (33)
このデータベースで認識されているユーザー名を取得します。値は VALUE-LEN パラメーターにより長さが決められる文字フィールドです。

DBATTR-ALLPROCEDURESARECALLABLE (34)
関数 DBPROCEDURES によって返されたすべてのプロシージャーを現在のユーザーが呼び出すことができるかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-ALLTABLESARESELECTABLE (35)
SELECT ステートメント内の関数 DBTABLES によって返されたすべての表を現在のユーザーが使用できるかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-DATADEFCAUSESTRANSACTCOMMIT (36)
トランザクション内のデータ定義ステートメントがトランザクションのコミットを強制するかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-DATADEFIGNOREDINTRANSACT (37)
このデータベースがトランザクション内のデータ定義ステートメントを無視するかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-DELETESAREDETECTED (38)
関数 FETCH が可視行の削除を検出できるかどうかを取得します。この属性が FALSE を返す場合、削除された行がカーソルから除去されることを意味します。値は 3 つのフルワード・バイナリー・フィールドからなる 12 バイトのエリアです。最初のフルワード・フィールドはカーソル・タイプ CURSOR-TYPE-FORWARD-ONLY (1003) 用、2 番目はカーソル・タイプ CURSOR-TYPE-SCROLL-INSENSITIVE (1004) 用、そして 3 番目はカーソル・タイプ CURSOR-TYPE-SCROLL-SENSITIVE(1005) 用です。 3 つのフィールドは、FALSE には 0、TRUE には 1 を含みます。

DBATTR-DOESMAXROWSIZEINCLUDEBLOBS (39)
属性 DBATTR-MAXROWSIZE (145) の値に SQL データ・タイプ LONGVARCHAR および LONGVARBINARY が含まれるかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-INSERTSAREDETECTED (40)
関数 FETCH が可視行の挿入を検出できるかどうかを取得します。値は 3 つのフルワード・バイナリー・フィールドからなる 12 バイトのエリアです。最初のフルワード・フィールドはカーソル・タイプ CURSOR-TYPE-FORWARD-ONLY (1003) 用、2 番目はカーソル・タイプ CURSOR-TYPE-SCROLL-INSENSITIVE (1004) 用、そして 3 番目はカーソル・タイプ CURSOR-TYPE-SCROLL-SENSITIVE(1005) 用です。 3 つのフィールドは、FALSE には 0、TRUE には 1 を含みます。

DBATTR-ISCATALOGATSTART (41)
カタログが完全修飾表名の先頭に表示されるかどうかを取得します。先頭に表示されない場合、カタログは末尾に表示されます。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-ISREADONLY (42)
このデータベースが読み取り専用モードであるかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-LOCATORSUPDATECOPY (43)
LOB への更新がコピーに対して加えられたか、LOB に直接加えられたかを示します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-NULLPLUSNONNULLISNULL (44)
NULL 値と NULL 以外の値の連結を NULL とすることをこのデータベースがサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-NULLSARESORTEDATEND (45)
ソート順に関係なく、NULL 値が末尾にソートされるかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-NULLSARESORTEDATSTART (46)
ソート順に関係なく、NULL 値が先頭にソートされるかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-NULLSARESORTEDHIGH (47)
NULL 値が高位にソートされるかどうかを取得します。高位にソートされるとは、NULL 値がドメイン内の他の値よりも高位にソートされることを意味します。昇順で、この属性値が TRUE の場合、NULL 値は最後に現れます。対照的に、属性 DBATTR-NULLSARESORTEDATEND (45) は、ソート順に関係なく NULL 値が末尾にソートされるかどうかを示します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-NULLSARESORTEDLOW (48)
NULL 値が下位にソートされるかどうかを取得します。下位にソートされるとは、NULL 値がドメイン内の他の値よりも下位にソートされることを意味します。昇順で、この属性値が TRUE の場合、NULL 値は先頭に現れます。対照的に、属性 DBATTR-NULLSARESORTEDATSTART (46) は、ソート順に関係なく NULL 値が先頭にソートされるかどうかを示します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-OTHERSDELETESAREVISIBLE (49)
他からの削除が可視かどうかを取得します。値は 3 つのフルワード・バイナリー・フィールドからなる 12 バイトのエリアです。最初のフルワード・フィールドはカーソル・タイプ CURSOR-TYPE-FORWARD-ONLY (1003) 用、2 番目はカーソル・タイプ CURSOR-TYPE-SCROLL-INSENSITIVE (1004) 用、そして 3 番目はカーソル・タイプ CURSOR-TYPE-SCROLL-SENSITIVE(1005) 用です。3 つのフィールドは、FALSE には 0、TRUE には 1 を含みます。

DBATTR-OTHERSINSERTSAREVISIBLE (50)
他からの挿入が可視かどうかを取得します。値は 3 つのフルワード・バイナリー・フィールドからなる 12 バイトのエリアです。最初のフルワード・フィールドはカーソル・タイプ CURSOR-TYPE-FORWARD-ONLY (1003) 用、2 番目はカーソル・タイプ CURSOR-TYPE-SCROLL-INSENSITIVE (1004) 用、そして 3 番目はカーソル・タイプ CURSOR-TYPE-SCROLL-SENSITIVE(1005) 用です。3 つのフィールドは、FALSE には 0、TRUE には 1 を含みます。

DBATTR-OTHERSUPDATESAREVISIBLE (51)
他からの更新が可視かどうかを取得します。値は 3 つのフルワード・バイナリー・フィールドからなる 12 バイトのエリアです。最初のフルワード・フィールドはカーソル・タイプ CURSOR-TYPE-FORWARD-ONLY (1003) 用、2 番目はカーソル・タイプ CURSOR-TYPE-SCROLL-INSENSITIVE (1004) 用、そして 3

番目はカーソル・タイプ CURSOR-TYPE-SCROLL-SENSITIVE(1005) 用です。3 つのフィールドは、FALSE には 0、TRUE には 1 を含みます。

DBATTR-OWNDELETESAREVISIBLE (52)
カーソル自体の削除が可視かどうかを取得します。値は 3 つのフルワード・バイナリー・フィールドからなる 12 バイトのエリアです。最初のフルワード・フィールドはカーソル・タイプ CURSOR-TYPE-FORWARD-ONLY (1003) 用、2 番目はカーソル・タイプ CURSOR-TYPE-SCROLL-INSENSITIVE (1004) 用、そして 3 番目はカーソル・タイプ CURSOR-TYPE-SCROLL-SENSITIVE(1005) 用です。 3 つのフィールドは、FALSE には 0、TRUE には 1 を含みます。

DBATTR-OWNINSERTSAREVISIBLE (53)
カーソル自体の挿入が可視かどうかを取得します。値は 3 つのフルワード・バイナリー・フィールドからなる 12 バイトのエリアです。最初のフルワード・フィールドはカーソル・タイプ CURSOR-TYPE-FORWARD-ONLY (1003) 用、2 番目はカーソル・タイプ CURSOR-TYPE-SCROLL-INSENSITIVE (1004) 用、そして 3 番目はカーソル・タイプ CURSOR-TYPE-SCROLL-SENSITIVE(1005) 用です。 3 つのフィールドは、FALSE には 0、TRUE には 1 を含みます。

DBATTR-OWNUPDATESAREVISIBLE (54)
カーソル自体の更新が可視かどうかを取得します。値は 3 つのフルワード・バイナリー・フィールドからなる 12 バイトのエリアです。最初のフルワード・フィールドはカーソル・タイプ CURSOR-TYPE-FORWARD-ONLY (1003) 用、2 番目はカーソル・タイプ CURSOR-TYPE-SCROLL-INSENSITIVE (1004) 用、そして 3 番目はカーソル・タイプ CURSOR-TYPE-SCROLL-SENSITIVE(1005) 用です。 3 つのフィールドは、FALSE には 0、TRUE には 1 を含みます。

DBATTR-STORESLOWERCASEIDENT (55)
引用符で囲まれていない大/小文字混合の SQL 識別子をデータベースが大/小文字を区別しないものとして処理し、その SQL 識別子を小文字で格納するかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-STORESLOWERCASEQUOTEDIDENT (56)
引用符で囲まれた大/小文字混合の SQL 識別子をこのデータベースが大/小文字を区別しないものとして処理し、その SQL 識別子を小文字で格納するかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-STORESMIXEDCASEIDENT (57)
引用符で囲まれていない大/小文字混合の SQL 識別子をデータベースが大/小文字を区別しないものとして処理し、その SQL 識別子を大/小文字混合で格納するかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-STORESMIXEDCASEQUOTEDIDENT (58)
引用符で囲まれた大/小文字混合の SQL 識別子をこのデータベースが大/小文字を区別しないものとして処理し、その SQL 識別子を大/小文字混合で格納するかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-STORESUPPERCASEIDENT (59)
引用符で囲まれていない大/小文字混合の SQL 識別子をデータベースが大/小

文字を区別しないものとして処理し、その SQL 識別子を大文字で格納するかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-STORESUPPERCASEQUOTEDIDENT (60)
引用符で囲まれた大/小文字混合の SQL 識別子をこのデータベースが大/小文字を区別しないものとして処理し、その SQL 識別子を大文字で格納するかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPANSI92ENTRYLEVELSQL (61)
このデータベースが ANSI92 のエントリー・レベルの SQL 文法をサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPANSI92FULLSQL (62)
このデータベースが ANSI92 の完全な SQL 文法をサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPANSI92INTERMSQL (63)
このデータベースが ANSI92 の中間レベルの SQL 文法をサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPALTTABLEWITHADDCOL (64)
このデータベースが ADD COLUMN 形式の ALTER TABLE をサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPALTTABLEWITHDROPCOL (65)
このデータベースが DROP COLUMN 形式の ALTER TABLE をサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPBATCHUPDATES (66)
このデータベースがバッチ更新をサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPCATSINDATAMANIPULATION (67)
データ操作ステートメント内でカタログ名を使用できるかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPCATSININDEXDEFINITIONS (68)
索引定義ステートメント内でカタログ名を使用できるかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPCATSINPRIVILEGEDEFS (69)
特権定義ステートメント内でカタログ名を使用できるかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPCATSINPROCEDURECALLS (70)
プロシージャー呼び出しステートメント内でカタログ名を使用できるかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPCATSINTABLEDEFINITIONS (71)
表定義ステートメント内でカタログ名を使用できるかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPCOLUMNALIASING (72)
このデータベースが列の別名割り当てをサポートするかどうかを取得します。サポートする場合、SQL AS 文節を使用して、必要に応じて算出欄に名前を指定したり、列に別名を指定したりできます。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPCORESQLGRAMMAR (73)
このデータベースが ODBC Core SQL 文法をサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPCORRELATEDSUBQUERIES (74)
このデータベースが相関サブ照会をサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPDATADEFANDDATAMANTRANS (75)
このデータベースが 1 つのトランザクション内でデータ定義ステートメントとデータ操作ステートメントの両方をサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPDATAMANTRANSACTIONSONLY (76)
このデータベースが 1 つのトランザクション内でデータ操作ステートメントのみをサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPDIFFTABLECORRNAMES (77)
表の相関名がサポートされる場合に、その相関名が表の名前と異なるように制限されるかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPEXPRESSIONSINORDERBY (78)
このデータベースが ORDER BY リスト内の式をサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPEXTENDEDSQLGRAMMAR (79)
このデータベースが ODBC Extended SQL 文法をサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPFULLOUTERJOINS (80)
完全にネストされた外部結合をこのデータベースがサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPGETGENERATEDKEYS (81)
ステートメントが実行された後に、自動生成されたキーを取得できるかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPGROUPBY (82)
このデータベースが GROUP BY 文節の何らかのフォームをサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPGROUPBYBEYONDSELECT (83)
SELECT ステートメントのすべての列が GROUP BY 文節に含まれる場合に、GROUP BY 文節の SELECT ステートメントに含まれない列の使用をこのデータベースがサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPGROUPBYUNRELATED (84)
GROUP BY 文節の SELECT ステートメントに含まれない列の使用をこのデータベースがサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPINTEGRITYENHANCEMENTFAC (85)
このデータベースが SQL Integrity Enhancement Facility をサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPLIKEESCAPECLAUSE (86)
このデータベースが LIKE エスケープ文節の指定をサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPLIMITEDOUTERJOINS (87)
このデータベースが提供する外部結合のサポートが制限されているかどうかを取得します (属性 DBATTR-SUPFULLOUTERJOINS (80) が TRUE の場合、この属性は TRUE になります)。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPMINIMUMSQLGRAMMAR (88)
このデータベースが ODBC Minimum SQL 文法をサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPMIXEDCASEIDENTIFIERS (89)
引用符で囲まれていない大/小文字混合の SQL 識別子をこのデータベースが大/小文字を区別するものとして処理し、結果としてその SQL 識別子を大/小文字混合で格納するかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPMIXEDCASEQUOTEDIDENT (90)
引用符で囲まれた大/小文字混合の SQL 識別子をこのデータベースが大/小文字を区別するものとして処理し、結果としてその SQL 識別子を大/小文字混合で格納するかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPMULTIPLEOPENRESULTS (91)
1 つのストアード・プロシージャー呼び出しから複数のカーソルを同時に返すことができるかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPMULTIPLERESULTSETS (92)
EXECUTE 関数への 1 回の呼び出しから複数のカーソルを取得することを、このデータベースがサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPMULTIPLETRANSACTIONS (93)
複数のトランザクションを (異なる接続で) 一度に開くことをこのデータベースが許可するかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPNAMEDPARAMETERS (94)
呼び出し可能なステートメントへの名前付きパラメーターをこのデータベースがサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPNONNULLABLECOLUMNS (95)
このデータベース内の列を NULL 不許可として定義できるかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPOPENCURACROSSCOMMIT (96)
コミット間でカーソルを開いたまま維持することをこのデータベースがサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPOPENCURACROSSROLLBACK (97)
ロールバック間でカーソルを開いたまま維持することをこのデータベースがサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPOPENSTMTSACROSSCOMMIT (98)
コミット間でステートメントを開いたまま維持することをこのデータベースがサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPOPENSTMTSACROSSROLLBACK (99)
ロールバック間でステートメントを開いたまま維持することをこのデータベースがサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPORDERBYUNRELATED (100)
ORDER BY 文節の SELECT ステートメントに含まれない列の使用をこのデータベースがサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPOUTERJOINS (101)
このデータベースが何らかの形式の外部結合をサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPPOSITIONEDDELETE (102)
位置指定された DELETE ステートメントをこのデータベースがサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPPOSITIONEDUPDATE (103)
位置指定された UPDATE ステートメントをこのデータベースがサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPRESULTSETTYPE (104)
このデータベースが結果セットの特定のタイプをサポートするかどうかを取得します。値は 3 つのフルワード・バイナリー・フィールドからなる 12 バイトのエリアです。最初のフルワード・フィールドはカーソル・タイプ CURSOR-TYPE-FORWARD-ONLY (1003) 用、2 番目はカーソル・タイプ CURSOR-TYPE-SCROLL-INSENSITIVE (1004) 用、そして 3 番目はカーソル・タイプ CURSOR-TYPE-SCROLL-SENSITIVE(1005) 用です。 3 つのフィールドは、FALSE には 0、TRUE には 1 を含みます。

DBATTR-SUPSAVEPOINTS (105)
このデータベースが保存ポイントをサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPSCHEMASINDATAMANIPUL (106)
データ操作ステートメント内でスキーマ名を使用できるかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPSCHEMASININDEXDEFS (107)
索引定義ステートメント内でスキーマ名を使用できるかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPSCHEMASINPRIVILEGEDEFS (108)
特権定義ステートメント内でスキーマ名を使用できるかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPSCHEMASINPROCEDURECALLS (109)
プロシージャー呼び出しステートメント内でスキーマ名を使用できるかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPSCHEMASINTABLEDEFS (110)
表定義ステートメント内でスキーマ名を使用できるかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPSELECTFORUPDATE (111)
このデータベースが SELECT FOR UPDATE ステートメントをサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPSTMTPOOLING (112)
このデータベースがステートメントのプーリングをサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPSTOREDPROCEDURES (113)
ストアード・プロシージャーのエスケープ構文を使用するストアード・プロシージャー呼び出しをこのデータベースがサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPSUBQUERIESINCOMPARISONS (114)
このデータベースが比較式内のサブ照会をサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPSUBQUERIESINEXISTS (115)
このデータベースが EXISTS 式内のサブ照会をサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPSUBQUERIESININS (116)
このデータベースが IN 式内のサブ照会をサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPSUBQUERIESINQUANTIFIEDS (117)
定量化式内のサブ照会をこのデータベースがサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPTABLECORRELATIONNAMES (118)
このデータベースが表の相関名をサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPTRANSACTIONS (119)
このデータベースがトランザクションをサポートするかどうかを取得します。サポートしない場合、COMMIT 関数の呼び出しはノーオペレーションであり、分離レベルは TRANSACTION-NONE (0) です。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPUNION (120)
このデータベースが SQL UNION をサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-SUPUNIONALL (121)
このデータベースが SQL UNION ALL をサポートするかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-UPDATESAREDETECTED (122)
FETCH 関数が可視行の更新を検出できるかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-USESLOCALFILEPERTABLE (123)
このデータベースが表ごとに 1 つのファイルを使用するかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-USESLOCALFILES (124)
このデータベースがローカル・ファイルに表を格納するかどうかを取得します。値は、FALSE には 0、TRUE には 1 を含む、フルワード・バイナリー変数です。

DBATTR-DATABASEMAJORVERSION (125)
基礎となっているデータベースのメジャー・バージョン番号を取得します。値はフルワード・バイナリー変数です。

DBATTR-DATABASEMINORVERSION (126)
基礎となっているデータベースのマイナー・バージョン番号を取得します。値はフルワード・バイナリー変数です。

DBATTR-DEFAULTTRANSACTIONISOLATION (127)
このデータベースのデフォルトのトランザクション分離レベルを取得します。値は、TRANSACTION-NONE(0)、TRANSACTION-READ-UNCOMMITTED (1)、TRANSACTION-READ-COMMITTED (2)、TRANSACTION-REPEATABLE-READ (4)、または TRANSACTION-SERIALIZABLE (8) を含むフルワード・バイナリー変数です。

DBATTR-DRIVERMAJORVERSION (128)
この JDBC ドライバーのメジャー・バージョン番号を取得します。値はフルワード・バイナリー変数です。

DBATTR-DRIVERMINORVERSION (129)
この JDBC ドライバーのマイナー・バージョン番号を取得します。値はフルワード・バイナリー変数です。

DBATTR-JDBCMAJORVERSION (130)
このドライバーのメジャー JDBC バージョン番号を取得します。値はフルワード・バイナリー変数です。

DBATTR-JDBCMINORVERSION (131)
このドライバーのマイナー JDBC バージョン番号を取得します。値はフルワード・バイナリー変数です。

DBATTR-MAXBINARYLITERALLENGTH (132)
このデータベースで許可される、インライン・バイナリー・リテラル内の 16 進文字の最大数を取得します。値がゼロの場合は、制限がないか、制限が不明であることを意味します。値はフルワード・バイナリー変数です。

DBATTR-MAXCATALOGNAMELENGTH (133)
このデータベースで許可される、カタログ名の最大文字数を取得します。値がゼロの場合は、制限がないか、制限が不明であることを意味します。値はフルワード・バイナリー変数です。

DBATTR-MAXCHARLITERALLENGTH (134)
このデータベースで許可される、文字リテラルの最大文字数を取得します。値がゼロの場合は、制限がないか、制限が不明であることを意味します。値はフルワード・バイナリー変数です。

DBATTR-MAXCOLUMNNAMELENGTH (135)
このデータベースで許可される、列名の最大文字数を取得します。値がゼロの場合は、制限がないか、制限が不明であることを意味します。値はフルワード・バイナリー変数です。

DBATTR-MAXCOLUMNSINGROUPBY (136)
このデータベースで許可される、GROUP BY 文節内の最大列数を取得します。値がゼロの場合は、制限がないか、制限が不明であることを意味します。値はフルワード・バイナリー変数です。

DBATTR-MAXCOLUMNSININDEX (137)
このデータベースで許可される、索引内の最大列数を取得します。値がゼロの場合は、制限がないか、制限が不明であることを意味します。値はフルワード・バイナリー変数です。

DBATTR-MAXCOLUMNSINORDERBY (138)
このデータベースで許可される、ORDER BY 文節内の最大列数を取得します。値がゼロの場合は、制限がないか、制限が不明であることを意味します。値はフルワード・バイナリー変数です。

DBATTR-MAXCOLUMNSINSELECT (139)
このデータベースで許可される、SELECT リスト内の最大列数を取得します。値がゼロの場合は、制限がないか、制限が不明であることを意味します。値はフルワード・バイナリー変数です。

DBATTR-MAXCOLUMNSINTABLE (140)
このデータベースで許可される、表内の最大列数を取得します。値がゼロの場合は、制限がないか、制限が不明であることを意味します。値はフルワード・バイナリー変数です。

DBATTR-MAXCONNECTIONS (141)
このデータベースへの可能な最大同時接続数を取得します。値がゼロの場合は、制限がないか、制限が不明であることを意味します。値はフルワード・バイナリー変数です。

DBATTR-MAXCURSORNAMELENGTH (142)
このデータベースで許可される、カーソル名の最大文字数を取得します。値がゼロの場合は、制限がないか、制限が不明であることを意味します。値はフルワード・バイナリー変数です。

DBATTR-MAXINDEXLENGTH (143)
このデータベースで許可される、索引の最大バイト数 (索引のすべての部分を含む) を取得します。値がゼロの場合は、制限がないか、制限が不明であることを意味します。値はフルワード・バイナリー変数です。

DBATTR-MAXPROCEDURENAMELENGTH (144)
このデータベースで許可される、プロシージャー名の最大文字数を取得します。値がゼロの場合は、制限がないか、制限が不明であることを意味します。値はフルワード・バイナリー変数です。

DBATTR-MAXROWSIZE (145)
このデータベースで許可される、1 行内の最大バイト数を取得します。値がゼロの場合は、制限がないか、制限が不明であることを意味します。値はフルワード・バイナリー変数です。

DBATTR-MAXSCHEMANAMELENGTH (146)
このデータベースで許可される、スキーマ名の最大文字数を取得します。値がゼロの場合は、制限がないか、制限が不明であることを意味します。値はフルワード・バイナリー変数です。

DBATTR-MAXSTMTLENGTH (147)
このデータベースで許可される、SQL ステートメント内の最大文字数を取得します。値がゼロの場合は、制限がないか、制限が不明であることを意味します。値はフルワード・バイナリー変数です。

DBATTR-MAXSTMTS (148)
同時に開くことができる、このデータベースへのアクティブ・ステートメントの最大数を取得します。値がゼロの場合は、制限がないか、制限が不明であることを意味します。値はフルワード・バイナリー変数です。

DBATTR-MAXTABLENAMELENGTH (149)
このデータベースで許可される、表名の最大文字数を取得します。値がゼロの場合は、制限がないか、制限が不明であることを意味します。値はフルワード・バイナリー変数です。

DBATTR-MAXTABLESINSELECT (150)
このデータベースで許可される、SELECT ステートメント内の最大表数を取得します。値がゼロの場合は、制限がないか、制限が不明であることを意味します。値はフルワード・バイナリー変数です。

DBATTR-MAXUSERNAMELENGTH (151)
このデータベースで許可される、ユーザー名の最大文字数を取得します。値がゼロの場合は、制限がないか、制限が不明であることを意味します。値はフルワード・バイナリー変数です。

DBATTR-RESULTSETHOLDABILITY (152)
カーソルのデフォルトの保持能力を取得します。値は、HOLD-CURSORS-OVER-COMMIT (1) または CLOSE-CURSORS-AT-COMMIT (2) を含むフルワード・バイナリー変数です。

DBATTR-SQLSTATETYPE (153)
GETLASTERROR によって返された SQLSTATE が X/Open (現在は Open Group として知られる) SQL CLI または SQL99 のどちらであるかを取得します。値は、X/Open では 1 を含み、SQL99 では 2 を含む、フルワード・バイナリー変数です。

DBATTR-SUPRESULTSETCONCURRENCY (154)
並行性タイプとカーソル・タイプの組み合わせをこのデータベースがサポートするかどうかを取得します。値は 6 つのフルワード・バイナリー・フィールドからなる 24 バイトのエリアです。最初のフルワード・フィールドはカーソル・タイプ CURSOR-TYPE-FORWARD-ONLY (1003) および CURSOR-CONCUR-READ-ONLY (1007) 用、2 番目はカーソル・タイプ CURSOR-TYPE-SCROLL-INSENSITIVE (1004) および CURSOR-CONCUR-READ-ONLY (1007) 用、3 番目はカーソル・タイプ CURSOR-TYPE-SCROLL-SENSITIVE(1005) および CURSOR-CONCUR-READ-ONLY (1007) 用です。4 番目のフルワード・フィールドはカーソル・タイプ CURSOR-TYPE-FORWARD-ONLY (1003) および CURSOR-CONCUR- UPDATABLE (1008) 用、5 番目はカーソル・タイプ CURSOR-TYPE-SCROLL-INSENSITIVE (1004) および CURSOR-CONCUR-

UPDATABLE (1008) 用、そして 6 番目はカーソル・タイプ CURSOR-TYPE-SCROLL-SENSITIVE(1005) および CURSOR-CONCUR-UPDATABLE (1008) 用です。6 つのフィールドは、FALSE には 0、TRUE には 1 を含みます。

DBATTR-SUPRESULTSETHOLDABILITY (155)
このデータベースがカーソルの保持能力をサポートするかどうかを取得します。値は 2 つのフルワード・バイナリー・フィールドからなる 8 バイトのエリアです。最初のフルワード・フィールドはカーソル保持能力 HOLD-CURSORS-OVER-COMMIT (1) 用、2 番目はカーソル・タイプ CLOSE-CURSORS-AT-COMMIT (2) 用です。2 つのフィールドは、FALSE には 0、TRUE には 1 を含みます。

DBATTR-SUPTRANSACTISOLATIONLEVEL (156)
このデータベースがトランザクション分離レベルをサポートするかどうかを取得します。値は 5 つのフルワード・バイナリー・フィールドからなる 20 バイトのエリアです。最初のフルワード・フィールドは分離レベル TRANSACTION-NONE (0) 用、2 番目は TRANSACTION-READ-UNCOMMITTED (1) 用、3 番目は TRANSACTION-READ-COMMITTED (2) 用、4 番目は TRANSACTION-REPEATABLE-READ (4) 用、そして 5 番目は TRANSACTION-SERIALIZABLE (8) 用です。5 つのフィールドは、FALSE には 0、TRUE には 1 を含みます。

GETENVATTR

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

GETENVATTR 関数を使用すると、アプリケーションで環境レベルの属性を取得できます。一般に、CONNECT 関数を使用して接続を確立する前に、環境属性を設定する必要があります。

関連する関数: 432 ページの『SETENVATTR』。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE         PIC S9(8) BINARY.
   01 ATTR               PIC S9(8) BINARY.
   01 VALUE              PIC S9(8) BINARY.
   01 VALUE-LEN          PIC S9(8) BINARY.
   01 RETCODE            PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-GETENVATTR ENV-HANDLE
        ATTR VALUE VALUE-LEN RETCODE.
```

FUNCTION (入力)
GETENVATTR を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

ATTR (入力)
設定または取得する属性の属性 ID を含むフルワード・バイナリー変数。

VALUE (入出力)
属性値を受け取るフィールド。一部の属性はテキスト値を使用するので、代わりに文字フィールドを使用する必要があります。

VALUE-LEN (入力)
VALUE フィールドの長さを含むフルワード・バイナリー変数。フルワード・バイナリー値を使用する属性では、長さは 4 に設定される必要があります。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

以下の環境属性が定義されます。

ENVATTR-TRACE (1)
DBCLI 内部トレースを有効または無効にすることができます。トレースは、動的に有効または無効にすることができます。この属性は即時に有効になります。値は以下のいずれかを含むフルワード・バイナリー変数です。

TRACE-OFF (0):
トレースが無効になります。

TRACE-SYSLST (1):
SYSLST へのトレースが有効になります

TRACE-SYSLOG (2):
SYSLOG へのトレースが有効になります。

TRACE-BOTH (3):
SYSLST および SYSLOG へのトレースが有効になります。

ENVATTR-RECV-BUFFER-SIZE (2)
この属性は、新しい接続を作成するときに使用される内部の受信バッファーのサイズを制御します。デフォルトの受信バッファー・サイズは 32KB です。値は、バイト単位のバッファー・サイズを含むフルワード・バイナリー変数です。この属性がゼロの値に設定された場合、受信バッファー・サイズはデフォルト・サイズ (32K) に設定されます。最小受信バッファー・サイズは 64 です。64 未満に設定しても、64 に設定されます。この属性を変更しても、既存の接続に影響はありません。SSL/TLS タイプの接続では、最小受信バッファー・サイズは 32K になります。

ENVATTR-SEND-BUFFER-SIZE (3)
この属性は、新しい接続を作成するときに使用される内部の送信バッファーのサイズを制御します。デフォルトの送信バッファー・サイズは 32KB です。値は、バイト単位のバッファー・サイズを含むフルワード・バイナリー変数です。この属性がゼロの値に設定された場合、送信バッファー・サイズはデフォルト・サイズ (32K) に設定されます。最小送信バッファー・サイズは 64 です。64 未満に設定しても、64 に設定されます。この属性を変更しても、既存の接続に影響はありません。SSL/TLS タイプの接続では、最小送信バッファー・サイズは 32K になります。

ENVATTR-DEFAULT-PORT (4)
この属性は、CONNECT 関数でポート番号 0 が指定された場合に使用される

デフォルトのポート番号を示します。値はフルワード・バイナリー変数です。初期のデフォルト・ポート番号は 16178 です。この属性を変更しても、既存の接続に影響はありません。

ENVATTR-CODEPAGE (5)
この属性は、テキスト・データを EBCDIC から ASCII またはその逆に変換するのに使用されるデフォルトの EBCDIC コード・ページを示します。値は最大 16 文字の文字フィールドです。デフォルトのコード・ページは Cp1047 です。この属性を変更しても、既存の接続に影響はありません。

ENVATTR-STATNUMCONS (6)
この読み取り専用属性を使用して、環境が作成された以降に開かれた接続の総数を取得できます。この属性を指定した SETENVATTR 呼び出しによって、カウンターがゼロにリセットされます。

ENVATTR-STATNUMSTMTS (7)
この読み取り専用属性を使用して、環境が作成された以降に開かれたステートメントの総数を取得できます。この属性を指定した SETENVATTR 呼び出しによって、カウンターがゼロにリセットされます。

ENVATTR-STATNUMEXECS (8)
この読み取り専用属性を使用して、環境が作成された以降に (すべてのステートメントと接続で) 実行されたステートメントの総実行数を取得できます。この属性を指定した SETENVATTR 呼び出しによって、カウンターがゼロにリセットされます。

ENVATTR-STATNUMROWS (9)
この読み取り専用属性を使用して、環境が作成された以降に (すべてのステートメントと接続で) 取り出された行の総数を取得できます。この属性を指定した SETENVATTR 呼び出しによって、カウンターがゼロにリセットされます。

ENVATTR-USE-CONNPOOL (10)
この属性を使用してアプリケーション用に接続プーリングの使用を有効にできます。デフォルトでは、接続プーリングは使用されません。接続プーリングの使用を有効にするには、アプリケーションで SETENVATTR 関数を使用してこの属性を TRUE (1) に設定する必要があります。

ENVATTR-CONNPOOL-TIMEOUT (11)
この属性を使用して、接続プーリングに入れられた接続の保持時間 (秒単位) を指定できます。この期間に接続が再利用されなかった場合、接続は閉じられます。デフォルトのタイムアウトは 600 秒 (10 分) です。SETENVATTR 関数を使用して ENVATTR-CONNPOOL-TIMEOUT 属性を設定することで、アプリケーションでタイムアウトを変更できます。

ENVATTR-CONNPOOL-MAXCONS (12)
この属性を使用して、接続プーリングに格納できる同一接続 (つまり、同じホスト名/IP アドレス、ポート、データベース名、ユーザー ID、およびパスワードを持つ接続) の最大数を指定できます。接続が接続プーリングに返されたときに、指定された最大同時接続数をすでに超えていた場合、接続は閉じられます。デフォルト数は 16 です。SETENVATTR 関数を使用して ENVATTR-CONNPOOL-MAXCONS 属性を設定することで、アプリケーションでこの最大数を変更できます。

GETLASTERROR

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

GETLASTERROR 関数を使用すると、最後の呼び出しに関するエラーの詳細をアプリケーションで取得できます。ほとんどの関数およびエラー状況について、エラーの詳細を入手できます。詳細には、テキストのエラー・メッセージだけでなく SQLCODE と SQLSTATE (ある場合) も含まれます。GETERRORMSG 関数では、戻りコードが返されますが、SQLCODE、SQLSTATE、またはエラー・メッセージは設定されません。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE       PIC S9(8) BINARY.
   01 SQLCODE          PIC S9(8) BINARY.
   01 SQLSTATE         PIC X(5).
   01 ERRMSG           PIC X(nnn).
   01 ERRMSG-LEN       PIC S9(8) BINARY.
   01 RETCODE          PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-GETLASTERROR ENV-HANDLE
        SQLCODE SQLSTATE [ERRMSG ERRMSG-LEN]
        RETCODE.
```

FUNCTION (入力)
GETLASTERROR を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

SQLCODE (出力)
最後の呼び出しの SQLCODE に設定されるフルワード・バイナリー変数。SQLCODE がない場合は 0 に設定されます。

SQLSTATE (出力)
最後の呼び出しの SQLSTATE に設定される 5 バイトの文字フィールド。SQLSTATE がない場合はブランクに設定されます。

ERRMSG (出力、オプション)
最後の呼び出しのテキスト・エラー・メッセージに設定される文字フィールド。エラー・メッセージがない場合はブランクに設定されます。ERRMSG パラメーターを指定する場合、ERRMSG-LEN パラメーターも指定する必要があります。エラー・メッセージが ERRMSG フィールド (ERRMSG-LEN に指定された長さ) よりも大きい場合、そのメッセージは切り捨てられます。

ERRMSG-LEN (入力、オプション)
ERRMSG フィールドの長さを含むフルワード・バイナリー変数。ERRMSG-LEN パラメーターを指定する場合、ERRMSG パラメーターも指定する必要があります。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

GETMORERESULTS

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

GETMORERESULTS 関数は、他の結果があるかどうかを検査し、他の結果がある場合はカーソルを次の結果に移動します。他の結果がない場合、GETMORERESULTS 関数は ENOMOREDATA を返します。

- 結果には、別のカーソルまたは更新カウントが考えられます。
- GETMORERESULTS 関数の呼び出しによって、すべてのオープン・カーソルが暗黙的に閉じられます。
- ステートメントがストアード・プロシージャー呼び出しであった場合、出力パラメーターにバインドされたホスト変数は、ストアード・プロシージャーによって設定された値に設定されます。
- 出力パラメーターは、他の結果がなくなった場合にのみ設定されます。したがって、ENOMOREDATA が返されるまで、GETMORERESULTS 関数を呼び出す必要があります。
- 更新ステートメントであった場合は、この関数によって更新カウントが提供されます。また、後で GETUPDATECOUNT 関数を使用して更新カウントを取得することもできます。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE            PIC S9(8) BINARY.
   01 STMT-HANDLE           PIC S9(8) BINARY.
   01 UPDATE-COUNT          PIC S9(8) BINARY.
   01 RETCODE               PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-GETMORERESULTS ENV-HANDLE
        STMT-HANDLE [UPDATE-COUNT] RETCODE.
```

FUNCTION (入力)
GETMORERESULTS を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (入力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。

UPDATE-COUNT (出力、オプション)
ステートメントの実行によって更新された行数に設定されるフルワード・バイナリー変数。更新カウントがない場合は -1 に設定されます。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

GETNUMCOLUMNS

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

GETNUMCOLUMNS 関数を使用すると、ステートメントが実行された後のカーソルに含まれている列の数をアプリケーションで照会できます。その後、アプリケーションで、BINDCOLUMN 関数を使用してホスト変数を列にバインドできます。

関連する関数:

- 334 ページの『BINDCOLUMN』
- 395 ページの『GETCOLUMNINFO』

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE            PIC S9(8) BINARY.
   01 STMT-HANDLE           PIC S9(8) BINARY.
   01 NUM-COLUMNS           PIC S9(8) BINARY.
   01 RETCODE               PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-GETNUMCOLUMNS ENV-HANDLE
        STMT-HANDLE NUM-COLUMNS RETCODE.
```

FUNCTION (入力)
GETNUMCOLUMNS を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (入力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。

NUM-COLUMNS (出力)
ステートメントが実行された後のカーソルに含まれている列の数に設定されるフルワード・バイナリー変数。カーソルがない場合は、エラーが返されます。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

GETNUMPARAMETERS

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

GETNUMPARAMETERS 関数を使用すると、ステートメントに含まれるパラメーターの数をアプリケーションで照会できます。その後、アプリケーションで、BINDPARAMETER 関数を使用してホスト変数をパラメーターにバインドできます。

関連する関数:

- 338 ページの『BINDPARAMETER』
- 419 ページの『GETPARAMETERINFO』

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE            PIC S9(8) BINARY.
   01 STMT-HANDLE           PIC S9(8) BINARY.
   01 NUM-PARAMS            PIC S9(8) BINARY.
   01 RETCODE               PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-GETNUMPARAMETERS ENV-HANDLE
        STMT-HANDLE NUM-PARAMS RETCODE.
```

FUNCTION (入力)
GETNUMPARAMETERS を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (入力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。

NUM-PARAMS (出力)
このステートメントのパラメーターの数に設定されるフルワード・バイナリー変数。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

GETPARAMETERINFO

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

GETPARAMETERINFO 関数を使用すると、ステートメントで使用されているパラメーターの属性をアプリケーションで照会できます。その後、アプリケーションで、BINDPARAMETER 関数を使用してホスト変数をパラメーターにバインドできます。

関連する関数:

- 338 ページの『BINDPARAMETER』
- 418 ページの『GETNUMPARAMETERS』

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE         PIC S9(8) BINARY.
   01 STMT-HANDLE        PIC S9(8) BINARY.
   01 PARAM-INDEX        PIC S9(8) BINARY.
   01 SQL-TYPE           PIC S9(8) BINARY.
   01 PRECISION          PIC S9(8) BINARY.
   01 SCALE              PIC S9(8) BINARY.
   01 SIGNED             PIC S9(8) BINARY.
   01 NULLABLE           PIC S9(8) BINARY.
   01 INOUTMODE          PIC S9(8) BINARY.
   01 RETCODE            PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-GETPARAMETERINFO ENV-HANDLE
        STMT-HANDLE PARM-INDEX SQL-TYPE [PRECISION [SCALE [SIGNED
        [NULLABLE [INOUTMODE]]]]] RETCODE.
```

FUNCTION (入力)
GETPARAMETERINFO を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (入力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。

PARAM-INDEX (入力)
情報を取得するステートメント・パラメーターの番号を含むフルワード・バイナリー変数。パラメーター番号は、1 から始まります。

SQL-TYPE (出力)
ステートメント・パラメーターの SQL-TYPE に設定されるフルワード・バイナリー変数。可能な値については、コピーブックの SQL-TYPE 宣言を参照してください。

PRECISION (出力、オプション)
ステートメント・パラメーターの精度に設定されるフルワード・バイナリー変数。このパラメーターはオプションです。

SCALE (出力、オプション)
ステートメント・パラメーターのスケールに設定されるフルワード・バイナリー変数。このパラメーターはオプションです。これを指定する場合、PRECISION パラメーターも指定する必要があります。

SIGNED (出力、オプション)
列が符号なしの場合に 0 に設定されるフルワード・バイナリー変数。ステートメント・パラメーターが符号ありの場合は 1 に設定されます。このパラメーターはオプションです。これを指定する場合、PRECISION および SCALE パラメーターも指定する必要があります。

NULLABLE (出力、オプション)
ステートメント・パラメーターに NULL を含めることができない場合に 0 に設定されるフルワード・バイナリー変数。ステートメント・パラメーターに NULL を含めることができる場合は 1 です。このパラメーターはオプションです。これを指定する場合、PRECISION、SCALE、および SIGNED パラメーターも指定する必要があります。

INOUTMODE (出力、オプション)
ステートメント・パラメーターの入出力モードに設定されるフルワード・バイナリー変数。可能な値は、INOUT-MODE-UNKNOWN (0)、INOUT-MODE-IN (1)、INOUT-MODE-OUT (2)、INOUT-MODE-INOUT (4) です。このパラメーターはオプションです。これを指定する場合、
PRECISION、SCALE、SIGNED、および NULLABLE パラメーターも指定する必要があります。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

GETROWNUMBER

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

GETROWNUMBER 関数を使用すると、カーソルが現在配置されている現在の行番号をアプリケーションで照会できます。オープン・カーソルがない場合、関数はエラーを返します。現在の行番号を取得する代わりの方法として、BINDCOLUMN 関数を使用し

て、フルワード・バイナリー・ホスト変数を列番号 0 にバインドする方法があります。これにより、FETCH 関数が呼び出されるたびにホスト変数が現在の行番号に設定されます。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE         PIC S9(8) BINARY.
   01 STMT-HANDLE        PIC S9(8) BINARY.
   01 ROW-NUMBER         PIC S9(8) BINARY.
   01 RETCODE            PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-GETROWNUMBER ENV-HANDLE
        STMT-HANDLE ROW-NUMBER RETCODE.
```

FUNCTION (入力)
GETROWNUMBER を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (入力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。

ROW-NUMBER (出力)
カーソルが現在配置されている現在の行番号に設定されるフルワード・バイナリー変数。カーソルがない場合は、エラーが返されます。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

GETSTMTATTR

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

GETSTMTATTR 関数を使用すると、アプリケーションがステートメント・レベルで属性を取得できます。

関連する関数: 435 ページの『SETSTMTATTR』。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE           PIC S9(8) BINARY.
   01 STMT-HANDLE          PIC S9(8) BINARY.
   01 ATTR                 PIC S9(8) BINARY.
   01 VALUE                PIC S9(8) BINARY.
   01 VALUE-LEN            PIC S9(8) BINARY.
   01 RETCODE              PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-GETSTMTATTR ENV-HANDLE STMT-HANDLE
        ATTR VALUE VALUE-LEN RETCODE.
```

FUNCTION (入力)
GETSTMTATTR を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (入力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。

ATTR (入力)
設定または取得する属性の属性 ID を含むフルワード・バイナリー変数。

VALUE (入出力)
属性値を受け取るフィールド。一部の属性はテキスト値を使用するので、代わりに文字フィールドを使用する必要があります。

VALUE-LEN (入力)
VALUE フィールドの長さを含むフルワード・バイナリー変数。フルワード・バイナリー値を使用する属性では、長さは 4 に設定される必要があります。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

以下のステートメント属性が定義されます。

STMTATTR-PREFETCHROWS (1)
FETCH 関数の実行時にプリフェッチされる行数を指定します。値は、プリフェッチする行数を含むフルワード・バイナリー変数です。デフォルトは 128 行です。値が大きいほどパフォーマンスは向上しますが、より多くのメモリーが必要になります。この属性をゼロに設定すると、デフォルトの 128 行に設定されます。

STMTATTR-FETCHSIZE (2)
より多くの行が必要な場合にデータベースからフェッチすべき行数に関して、JDBC ドライバーにヒントを与えます。値ゼロが指定されている場合、ヒントは無視されます。デフォルト値はゼロです。

STMTATTR-QUERYTIMEOUT (3)
ステートメントが実行されるまでに JDBC ドライバーが待機する秒数を指定します。この制限を超えると、SQL エラーが生成されますゼロは無制限を意味します。デフォルト値はゼロです。

STMTATTR-MAXFIELDSIZE (4)
文字またはバイナリー値を格納するカーソル列の最大バイト数の制限を、任意のバイト数に指定します。この制限は、BINARY、 VARBINARY、 LONGVARBINARY、 CHAR、 VARCHAR、 LONGVARCHAR の各フィールドにのみ適用されます。この制限を超えると、超過データは確認されることなく破棄されます。移植性を最大にするためには、256 より大きい値を指定してください。ゼロは無制限を意味します。デフォルト値はゼロです。

STMTATTR-MAXROWS (5)
任意のカーソルに入れることのできる最大行数の制限を指定します。この制限を超えると、超過した行は確認されることなく除去されます。ゼロは無制限を意味します。デフォルト値はゼロです。

STMTATTR-TYPE (6)
この属性は読み取り専用です。ここにはカーソル・タイプが含まれています。カーソル・タイプは、カーソルが前方スクロールのみか、通常のスクロールに使用できるかを制御します。以下の値が使用できます。

CURSOR-TYPE-FORWARD-ONLY (1003):
カーソルは、前方にのみ移動できます。

CURSOR-TYPE-SCROLL-INSENSITIVE (1004):
カーソルはスクロール可能ですが、一般に、他者による変更は認識しません。

CURSOR-TYPE-SCROLL-SENSITIVE (1005):
カーソルはスクロール可能で、一般に、他者による変更も認識します。

STMTATTR-CONCURRENCY (7)
この属性は読み取り専用です。ここには並行性モードが含まれています。並行性モードは、カーソルが読み取り専用か、更新可能かを制御します。以下の値が使用できます。

CURSOR-CONCUR-READ-ONLY (1007):
カーソルは読み取り専用です。

CURSOR-CONCUR-UPDATABLE (1008):
カーソルは更新可能です。

STMTATTR-HOLDABILITY (8)
この属性は読み取り専用です。ここには保持能力モードが含まれています。値は以下のいずれかを含むフルワード・バイナリー変数です。

HOLD-CURSORS-OVER-COMMIT (1):
COMMIT が実行されても、オープン・カーソルは開いたままです。

CLOSE-CURSORS-AT-COMMIT (2):
COMMIT が実行されると、すべてのオープン・カーソルが閉じられます。

STMTATTR-STATNUMEXECS (9)
この読み取り専用属性で、このステートメントが実行された合計回数を取得できます。

STMTATTR-STATNUMROWS (10)
この読み取り専用属性で、このステートメントを使用して取り出された行の合計数を取得できます。

STMTATTR-NOBINDCHECK (11)
この属性で、EXECUTE 関数を介してステートメントを実行するときにすべての入力パラメーターをバインドするかどうかの確認を「オフ」にすることができます。デフォルトではこの属性は FALSE (0) で、これはバインドの確認が実行される ことを意味します。バインドの確認を「オフ」にするには、この属性を TRUE (1) に設定します。

GETUPDATECOUNT

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

GETUPDATECOUNT 関数は、最後のステートメント実行から更新カウント値を取得します。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE           PIC S9(8) BINARY.
   01 STMT-HANDLE          PIC S9(8) BINARY.
```

```
   01 UPDATE-COUNT         PIC S9(8) BINARY.
   01 RETCODE              PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-GETUPDATECOUNT ENV-HANDLE
        STMT-HANDLE UPDATE-COUNT RETCODE.
```

FUNCTION (入力)
GETUPDATECOUNT を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (入力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。

UPDATE-COUNT (出力)
ステートメントの実行によって更新された行数に設定されるフルワード・バイナリー変数。更新カウントがない場合は -1 に設定されます。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

INITENV

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

INITENV 関数は、DBCLI API に対する最初の呼び出しでなければなりません。これによって、EZA TCP/IP インターフェースなどの環境が初期化されます。 INITENV 呼び出しは、環境ハンドルを割り振り、ENV-HANDLE パラメーターを設定します。アプリケーションは、後続のすべての関数にこの環境ハンドルを渡す必要があります。

注: この関数を使用する前に、必ず 326 ページの『プログラミングの制限と要件』(特に、CICS トランザクションで DBCLI API を使用する際の注意事項) を一読してください。

関連する関数: 436 ページの『TERMENV』。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE             PIC S9(8) BINARY.
   01 IDENT.
       02 TCPNAME            PIC X(8).
       02 ADSNAME            PIC X(8).
   01 RETCODE                PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-INITENV ENV-HANDLE
        [TCPNAME [ADSNAME]] RETCODE.
```

FUNCTION (入力)
INITENV を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (出力)
環境ハンドルを含むフルワード・バイナリー変数。INITENV 呼び出しにより設定された ENV-HANDLE 値は、それ以後、他の DBCLI 関数を呼び出すたびに指定する必要があります。

TCPNAME (入力、オプション)
このオプション・パラメーターを使用すると、このアプリケーションで使用するローカルの TCP/IP スタックを選択できます。この 8 バイトのパラメーターは、「SOCKETnn」に設定することも、単に「nn」(左揃えまたは右揃えされ、6 個のブランクが埋め込まれる) とだけ設定することもできます。値「nn」によって、選択した TCP/IP スタックの ID が決まります。この値が、TCP/IP 始動 JCL では ID パラメーターで指定されるためです。 TCPNAME が指定されていない場合は、// OPTION SYSPARM='nn' ステートメントの ID が使用されます。上記のいずれも指定されていない場合、ID はデフォルトで「00」になります。

ADSNAME (入力、オプション)
このオプション・パラメーターを使用すると、EZA 処理環境で使用される TCP/IP インターフェース・ルーチンの名前を指定できます。ここで何も指定しない場合は、IBM 提供の TCP/IP インターフェース・ルーチン EZASOH99 が使用されます。この指定は、JCL ステートメント // SETPARM EZA$PHA='routine-name' で上書きできることにご注意ください。
ADSNAME パラメーターは、TCPNAME パラメーターも指定されている場合に限って指定できます。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

INITSSL

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

SSL/TLS 接続を使用する計画がある場合は、CONNECTSSL 関数を呼び出す前に SSL/TLS 環境を初期化する必要があります。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE       PIC S9(8) BINARY.
   01 SECTYPE          PIC X(8) VALUE IS 'SSL30'.
   01 KEYRING          PIC X(nnn) VALUE IS 'CRYPTO.KEYRING'.
   01 KEYRING-LEN      PIC S9(8) BINARY VALUE IS 14.
   01 SESS-TIMOUT      PIC S9(8) BINARY VALUE IS 86400.
   01 RETCODE          PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-INITSSL ENV-HANDLE
        SECTYPE KEYRING KEYRING-LEN [SESS-TIMEOUT] RETCODE.
```

FUNCTION (入力)
INITSSL を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

SECTYPE (入力)
このパラメーターは、受け入れ可能な最小限のセキュリティー・プロトコルを指定します。値は大文字で入力し、ブランクを埋め込む必要があります。使用できる値は、「SSL30」(SSL バージョン 3.0 の場合) または「TLS31」(TLS バージョン 1.0 の場合) です。

KEYRING (入力)
このパラメーターは、SSL 鍵および証明書の読み取り元となる鍵リングを指定します。通常は、z/VSE ライブラリーと、鍵素材が格納されているサブライブラリー (*lib.sublib*) を指定します。

KEYRING-LEN (入力)
KEYRING フィールドの長さを含むフルワード・バイナリー変数。

SESS-TIMEOUT (入力、オプション)
SSL/TLS セッション・タイムアウト (秒) を含むフルワード・バイナリー変数。このパラメーターが指定されていない場合は、デフォルトのタイムアウト (86400 秒) が使用されます。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

PREPARECALL

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

PREPARECALL 関数は、ストアード・プロシージャー呼び出しステートメントの実行準備を行います。準備中に、データベースは通常 SQL ステートメントをプリコンパイルし、ステートメントのアクセス・プランを作成します。

SQL ステートメントには、実行時に評価されるパラメーターが含まれることがあります。 SQL ステートメント内では、パラメーターは疑問符 (?) でマーク付けされます。パラメーターには、出現順に「1」から始まる数字が付けられます。準備が終わったら、アプリケーションはホスト変数をパラメーターにバインドできます。後でステートメントを実行する際には、そのホスト変数の内容が使用され、データベースに送信されます。

関連する関数: 428 ページの『PREPARESTATEMENT』。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE           PIC S9(8) BINARY.
   01 CON-HANDLE           PIC S9(8) BINARY.
   01 STMT-HANDLE          PIC S9(8) BINARY.
   01 SQL                  PIC X(nnn) VALUE IS 'CALL INOUT_PARAM(?)'.
   01 SQL-LEN              PIC S9(8) BINARY VALUE IS 30.
   01 CURSOR-TYPE          PIC S9(8) BINARY VALUE IS 1003.
   01 CONCURRENCY          PIC S9(8) BINARY VALUE IS 1007.
   01 HOLDABILITY          PIC S9(8) BINARY VALUE IS 1.
   01 RETCODE              PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-PREPARECALL ENV-HANDLE CON-HANDLE
        STMT-HANDLE SQL SQL-LEN [CURSOR-TYPE CONCURRENCY [HOLDABILITY]]
        RETCODE.
```

FUNCTION (入力)
PREPARECALL を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

CON-HANDLE (入力)
接続ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (出力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。PREPARECALL 呼び出しにより設定された STMT-HANDLE 値は、ステートメントを必要とする他の DBCLI 関数への後続の呼び出しで指定される必要があります。

SQL (入力)
準備対象の SQL ステートメントを含む文字フィールド。このフィールドは SQL-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。

SQL-LEN (入力)
SQL パラメーターで指定された SQL ステートメントの長さを含む、フルワード・バイナリー変数。

CURSOR-TYPE (入力、オプション)
カーソル・タイプを含むフルワード・バイナリー変数。このパラメーターはオプションです。これを指定する場合は、CONCURRENCY パラメーターも指定する必要があります。カーソル・タイプは、カーソルが前方スクロールのみか、通常のスクロールに使用できるかを制御します。以下の値が使用できます。

CURSOR-TYPE-FORWARD-ONLY (1003):
カーソルは、前方にのみ移動できます。

CURSOR-TYPE-SCROLL-INSENSITIVE (1004):
カーソルはスクロール可能ですが、一般に、他者による変更は認識しません。

CURSOR-TYPE-SCROLL-SENSITIVE (1005):
カーソルはスクロール可能で、一般に、他者による変更も認識します。

CONCURRENCY (入力、オプション)
並行性モードを含むフルワード・バイナリー変数。このパラメーターはオプションです。これを指定する場合は、CURSOR-TYPE パラメーターも指定する必要があります。並行性モードは、カーソルが読み取り専用か、更新可能かを制御します。以下の値が使用できます。

CURSOR-CONCUR-READ-ONLY (1007):
カーソルは読み取り専用です。

CURSOR-CONCUR-UPDATABLE (1008):
カーソルは更新可能です。

HOLDABILITY (入力、オプション)
保持能力モードを含むフルワード・バイナリー変数。このパラメーターはオプションです。これを指定する場合は、CURSOR-TYPE パラメーターと

CONCURRENCY パラメーターも指定する必要があります。値は以下のいずれかを含むフルワード・バイナリー変数です。

HOLD-CURSORS-OVER-COMMIT (1):
COMMIT が実行されても、オープン・カーソルは開いたままです。

CLOSE-CURSORS-AT-COMMIT (2):
COMMIT が実行されると、すべてのオープン・カーソルが閉じられます。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

PREPARESTATEMENT

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

PREPARESTATEMENT 関数は、SQL ステートメントの実行準備を行います。このステートメントを表すステートメント・ハンドルを割り振り、STMT-HANDLE パラメーターを設定します。アプリケーションは、ステートメントを必要とする後続のすべての関数にステートメント・ハンドルを受け渡す必要があります。アプリケーションは、一度に複数のステートメントを準備できます。アプリケーションは、必要なステートメント・ハンドルを後続の関数で確実に使用する必要があります。

SQL ステートメントには、実行時に評価されるパラメーターが含まれることがあります。 SQL ステートメント内では、パラメーターは疑問符 (?) でマーク付けされます。パラメーターには、出現順に 1 から始まる数字が付けられます。準備が終わったら、アプリケーションはホスト変数をパラメーターにバインドできます。後でステートメントを実行する際には、そのホスト変数の内容が使用され、データベースに送信されます。

関連する関数:
- 341 ページの『CLOSESTATEMENT』
- 426 ページの『PREPARECALL』

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE           PIC S9(8) BINARY.
   01 CON-HANDLE           PIC S9(8) BINARY.
   01 STMT-HANDLE          PIC S9(8) BINARY.
   01 SQL                  PIC X(nnn) VALUE IS 'SELECT * FROM TABLE WHERE A=?'.
   01 SQL-LEN              PIC S9(8) BINARY VALUE IS 30.
   01 CURSOR-TYPE          PIC S9(8) BINARY VALUE IS 1003.
   01 CONCURRENCY          PIC S9(8) BINARY VALUE IS 1007.
   01 HOLDABILITY          PIC S9(8) BINARY VALUE IS 1.
   01 RETCODE              PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-PREPARESTATEMENT ENV-HANDLE CON-HANDLE
        STMT-HANDLE SQL SQL-LEN [CURSOR-TYPE CONCURRENCY [HOLDABILITY]]
        RETCODE.
```

FUNCTION (入力)
PREPARESTATEMENT を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

CON-HANDLE (入力)
接続ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (出力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。PREPARESTATEMNT 呼び出しにより設定された STMT-HANDLE 値は、ステートメントを必要とする他の DBCLI 関数への後続の呼び出しで指定される必要があります。

SQL (入力)
準備対象の SQL ステートメントを含む文字フィールド。このフィールドは SQL-LEN パラメーターにより指定された長さ以内で、右側にブランクを埋め込む必要があります。

SQL-LEN (入力)
SQL パラメーターで指定された SQL ステートメントの長さを含む、フルワード・バイナリー変数。

CURSOR-TYPE (入力、オプション)
カーソル・タイプを含むフルワード・バイナリー変数。このパラメーターはオプションです。これを指定する場合は、CONCURRENCY パラメーターも指定する必要があります。カーソル・タイプは、カーソルが前方スクロールのみか、通常のスクロールに使用できるかを制御します。以下の値が使用できます。

CURSOR-TYPE-FORWARD-ONLY (1003):
カーソルは、前方にのみ移動できます。

CURSOR-TYPE-SCROLL-INSENSITIVE (1004):
カーソルはスクロール可能ですが、一般に、他者による変更は認識しません。

CURSOR-TYPE-SCROLL-SENSITIVE (1005):
カーソルはスクロール可能で、一般に、他者による変更も認識します。

CONCURRENCY (入力、オプション)
並行性モードを含むフルワード・バイナリー変数。このパラメーターはオプションです。これを指定する場合は、CURSOR-TYPE パラメーターも指定する必要があります。並行性モードは、カーソルが読み取り専用か、更新可能かを制御します。以下の値が使用できます。

CURSOR-CONCUR-READ-ONLY (1007):
カーソルは読み取り専用です。

CURSOR-CONCUR-UPDATABLE (1008):
カーソルは更新可能です。

HOLDABILITY (入力、オプション)
保持能力モードを含むフルワード・バイナリー変数。このパラメーターはオプションです。これを指定する場合は、CURSOR-TYPE パラメーターと CONCURRENCY パラメーターも指定する必要があります。値は以下のいずれかを含むフルワード・バイナリー変数です。

HOLD-CURSORS-OVER-COMMIT (1):
COMMIT が実行されても、オープン・カーソルは開いたままです。

CLOSE-CURSORS-AT-COMMIT (2):
COMMIT が実行されると、すべてのオープン・カーソルが閉じられます。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

RELEASESAVEPOINT

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

RELEASESAVEPOINT 関数は、指定した保存ポイントを現在の作業論理単位から削除します。

関連する関数: 434 ページの『SETSAVEPOINT』。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE           PIC S9(8) BINARY.
   01 CON-HANDLE           PIC S9(8) BINARY.
   01 SAVEPOINT            PIC S9(8) BINARY.
   01 RETCODE              PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-RELEASESAVEPOINT ENV-HANDLE CON-HANDLE
        SAVEPOINT RETCODE.
```

FUNCTION (入力)
RELEASESAVEPOINT を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

CON-HANDLE (入力)
接続ハンドルを含むフルワード・バイナリー変数。

SAVEPOINT (入力)
解放する保存ポイントを含むフルワード・バイナリー変数。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

ROLLBACK

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

ROLLBACK 関数は、最後の作業論理単位で実行されたデータベース内のすべての変更をロールバックします。保存ポイントが指定されている場合、その保存ポイント以降に行われたすべての変更がロールバックされます。保存ポイントは、SETSAVEPOINT 関数で設定できます。

関連する関数: 342 ページの『COMMIT』。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE         PIC S9(8) BINARY.
```

```
   01 CON-HANDLE         PIC S9(8) BINARY.
   01 SAVEPOINT          PIC S9(8) BINARY.
   01 RETCODE            PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-ROLLBACK ENV-HANDLE CON-HANDLE
        [SAVEPOINT] RETCODE.
```

FUNCTION (入力)
ROLLBACK を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

CON-HANDLE (入力)
接続ハンドルを含むフルワード・バイナリー変数。

SAVEPOINT (入力、オプション)
ロールバックする保存ポイントを含むフルワード・バイナリー変数。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

SETCONNATTR

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

SETCONNATTR 関数を使用すると、アプリケーションが接続レベルで属性を設定できます。

関連する関数: 396 ページの『GETCONNATTR』。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE           PIC S9(8) BINARY.
   01 CON-HANDLE           PIC S9(8) BINARY.
   01 ATTR                 PIC S9(8) BINARY.
   01 VALUE                PIC S9(8) BINARY.
   01 VALUE-LEN            PIC S9(8) BINARY.
   01 RETCODE              PIC S9(8) BINARY.
PROCEDURE
  CALL 'IESDBCLI' USING FUNC-SETCONNATTR ENV-HANDLE CON-HANDLE
       ATTR VALUE VALUE-LEN RETCODE.
```

FUNCTION (入力)
SETCONNATTR を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

CON-HANDLE (入力)
接続ハンドルを含むフルワード・バイナリー変数。

ATTR (入力)
設定または取得する属性の属性 ID を含むフルワード・バイナリー変数。

VALUE (入出力)
新しい属性値を含むフィールド。一部の属性はテキスト値を使用するので、代わりに文字フィールドを使用する必要があります。

VALUE-LEN (入力)
VALUE フィールドの長さを含むフルワード・バイナリー変数。フルワード・バイナリー値を使用する属性では、長さは 4 に設定される必要があります。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

定義される接続属性 (データベースのメタ情報 を取得するための読み取り専用接続属性を含む) の詳細については、 396 ページの『GETCONNATTR』を参照してください。

SETENVATTR

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

SETENVATTR 関数を使用すると、アプリケーションが環境レベルで属性を設定できます。一般に、CONNECT 関数を使用して接続を確立する前に、環境属性を設定する必要があります。

関連する関数: 413 ページの『GETENVATTR』。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE         PIC S9(8) BINARY.
   01 ATTR               PIC S9(8) BINARY.
   01 VALUE              PIC S9(8) BINARY.
   01 VALUE-LEN          PIC S9(8) BINARY.
   01 RETCODE            PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-SETENVATTR ENV-HANDLE
        ATTR VALUE VALUE-LEN RETCODE.
```

FUNCTION (入力)
SETENVATTR を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

ATTR (入力)
設定または取得する属性の属性 ID を含むフルワード・バイナリー変数。

VALUE (入出力)
新しい属性値を含むフィールド。一部の属性はテキスト値を使用するので、代わりに文字フィールドを使用する必要があります。

VALUE-LEN (入力)
VALUE フィールドの長さを含むフルワード・バイナリー変数。フルワード・バイナリー値を使用する属性では、長さは 4 に設定される必要があります。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

定義される環境属性 の詳細については、 413 ページの『GETENVATTR』を参照してください。

SETPOS

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

SETPOS 関数を使用すると、アプリケーションが、データを取り出すことなくカーソルを配置できるようになります。また、オープン・カーソルのデータ行の最新表示、更新、削除、または挿入も可能です。操作もよりますが、SETPOS 関数は、BINDCOLUMN 関数を使用してカーソルにバインドされたすべてのホスト変数を使用します。さらに、列が NULL であるか、または更新の際に無視してもよいかを示す、対応する標識変数も使用します。

カーソルの行が更新できるのは、カーソルが更新可能またはスクロール可能な場合、すなわちステートメントが CURSOR-CONCUR-UPDATABLE と、CURSOR-TYPE-SCROLL-INSENSITIVE または CURSOR-TYPE-SCROLL-SENSITIVE のいずれかを使用して作成されている場合に限られます。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE            PIC S9(8) BINARY.
   01 STMT-HANDLE           PIC S9(8) BINARY.
   01 OPERATION             PIC S9(8) BINARY.
   01 ROW-NUMBER            PIC S9(8) BINARY.
   01 RETCODE               PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-SETPOS ENV-HANDLE
        STMT-HANDLE OPERATION ROW-NUMBER RETCODE.
```

FUNCTION (入力)
SETPOS を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (入力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。

OPERATION (入力)
命令コードを含むフルワード・バイナリー変数。以下の操作がサポートされます。

SETPOS-POSITION (0)
ROW-NUMBER パラメーターで指定した行にカーソルを配置します。

SETPOS-REFRESH (1)
ROW-NUMBER パラメーターで指定した行にカーソルを配置して、JDBC ドライバーでその行の最新表示を行います。

SETPOS-UPDATE (2)
ROW-NUMBER パラメーターで指定した行にカーソルを配置して、その行の列を更新します。 SETPOS-UPDATE 操作は、BINDCOLUMN 関数を使用してカーソルにバインドされたすべてのホスト変数を使用します。さらに、列が NULL であるか、または更新の際に無視してもよいか (INDICATE-IGNOREUPDATE) を示す、対応する標識変数も使用します。

SETPOS-DELETE (3)
ROW-NUMBER パラメーターで指定した行にカーソルを配置して、その行を削除します。

SETPOS-INSERT (4)
新しい行を挿入します。この操作では ROW-NUMBER パラメーターは無視されます。 SETPOS-INSERT 操作は、BINDCOLUMN 関数を使用してカーソルにバインドされたすべてのホスト変数を使用します。さらに、列が NULL であるか、または挿入の際に無視してもよいか (INDICATE-IGNOREUPDATE) を示す、対応する標識変数も使用します。

ROW-NUMBER (入力)
絶対行番号を含むフルワード・バイナリー変数。SETPOS-INSERT 操作では、ROW-NUMBER は無視されます。他のすべての操作では必須です。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

SETSAVEPOINT

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

SETSAVEPOINT 関数は、現在の作業論理単位に保存ポイントを設定します。保存ポイントは、あとで ROLLBACK 関数によって変更をロールバックする際に使用できます。

関連する関数: 430 ページの『RELEASESAVEPOINT』。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE         PIC S9(8) BINARY.
   01 CON-HANDLE         PIC S9(8) BINARY.
   01 SAVEPOINT          PIC S9(8) BINARY.
   01 RETCODE            PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-SETSAVEPOINT ENV-HANDLE CON-HANDLE
        SAVEPOINT RETCODE.
```

FUNCTION (入力)
SETSAVEPOINT を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

CON-HANDLE (入力)
接続ハンドルを含むフルワード・バイナリー変数。

SAVEPOINT (出力)
あとで RELEASESAVEPOINT 関数または ROLLBACK 関数で使用できる保存ポイントのハンドルを含むフルワード・バイナリー変数。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

SETSTMTATTR

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

SETSTMTATTR 関数を使用すると、アプリケーションがステートメント・レベルで属性を設定できます。

関連する関数: 421 ページの『GETSTMTATTR』。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE            PIC S9(8) BINARY.
   01 STMT-HANDLE           PIC S9(8) BINARY.
   01 ATTR                  PIC S9(8) BINARY.
   01 VALUE                 PIC S9(8) BINARY.
   01 VALUE-LEN             PIC S9(8) BINARY.
   01 RETCODE               PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-SETSTMTATTR ENV-HANDLE STMT-HANDLE
        ATTR VALUE VALUE-LEN RETCODE.
```

FUNCTION (入力)
SETSTMTATTR を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

STMT-HANDLE (入力)
ステートメント・ハンドルを含むフルワード・バイナリー変数。

ATTR (入力)
設定または取得する属性の属性 ID を含むフルワード・バイナリー変数。

VALUE (入出力)
新しい属性値を含むフィールド。一部の属性はテキスト値を使用するので、代わりに文字フィールドを使用する必要があります。

VALUE-LEN (入力)
VALUE フィールドの長さを含むフルワード・バイナリー変数。フルワード・バイナリー値を使用する属性では、長さは 4 に設定される必要があります。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

定義されるステートメント属性 の詳細については、 421 ページの『GETSTMTATTR』を参照してください。

TERMENV

(カテゴリーごとにソートされたすべての DBCLI 関数の「ロードマップ」については、 333 ページの表 8 を参照してください。)

TERMENV 関数は、DBCLI API に対する最後の呼び出しでなければなりません。これによって、EZA TCP/IP インターフェースなどの環境が終了します。 TERMENV 呼び出しは、環境ハンドルを解放します。 INITENV 呼び出しを除いて、DBCLI 関数の呼び出しはこれ以上は許可されません。

関連する関数: 424 ページの『INITENV』。

```
WORKING STORAGE
   COPY IESDBCOB.
   01 ENV-HANDLE    PIC S9(8) BINARY.
   01 RETCODE       PIC S9(8) BINARY.
PROCEDURE
   CALL 'IESDBCLI' USING FUNC-TERMENV ENV-HANDLE RETCODE.
```

FUNCTION (入力)
TERMENV を含む 16 バイトの文字フィールド。フィールドは左揃えされ、右側にブランクが埋め込まれます。

ENV-HANDLE (入力)
環境ハンドルを含むフルワード・バイナリー変数。

RETCODE (出力)
DBCLI 呼び出しにより戻されたコードを含む、フルワード・バイナリー変数。戻りコード・パラメーターは、常に最後のパラメーターでなければなりません。

DBCLI 使用時のパフォーマンスに関する考慮事項

データベース呼び出しレベル・インターフェース (DBCLI) 使用時には、以下のようなパフォーマンスに関する考慮事項が適用されます。

SSL/TLS の使用

SSL (Secure Socket Layer)/TLS (Transport Layer Security) を使用すると、DBCLI サーバー (DBCliServer) との通信を必要とするすべての関数呼び出しに、一定量のオーバーヘッドが加わります。このオーバーヘッドは、ネットワーク接続で送受信されるすべてのデータを暗号化処理するために生じるものです。ハードウェア暗号サポートの使用を強くお勧めします (CPACF および Crypto Express® カード)。

プリフェッチ

通常、照会またはストアード・プロシージャー呼び出しの結果行は、複数行の「チャンク」で取り出されます。つまり、バインドされたすべての列のデータは 1 つのデータ・ブロックとしてネットワーク接続経由で送信されます。プリフェッチ と呼ばれるこのプロセスによって、処理とネットワークのオーバーヘッドが節減されます。

- プリフェッチされる行数は、ステートメント属性 STMTATTR-PREFETCHROWS (0) で制御できます。デフォルトでは、128 行がプリフェッチされます。
- プリフェッチの行数を多くすると、取り出しのパフォーマンスは向上する可能性がありますが、より多くのメモリーが必要になります。

- メモリーの使用量は、1 行当たりの列の数とそのサイズにも依存します。

注: カーソル・タイプ CURSOR-TYPE-SCROLL-SENSITIVE (1005) が使用されている場合、プリフェッチはこのステートメントでは無効 になります。

- スクロール可能で、他者による変更を認識するカーソルの場合、プリフェッチは使用できません。既にプリフェッチされた行の行データがデータベースで変更される可能性があるためです。
- したがって、カーソル・タイプ CURSOR-TYPE-SCROLL-SENSITIVE (1005) を使用すると、FETCH 関数が呼び出されたときには、すべての行が一度に別々に取り出されます。
- 取り出される行の数が非常に多い場合には、これがパフォーマンスに悪影響を与えることがあります。

列のバインディング

照会またはストアード・プロシージャーの結果データを取得するには、ホスト変数をカーソルの列にバインドする必要があります。行の (プリ) フェッチ時に、すべてのバインド済み列のデータはネットワーク経由で転送されます。行のすべての列のデータ変換は、DBCliServer サイドで行われます。列のデータのサイズは、列にバインドされるホスト変数に依存しています。

最良のパフォーマンスを得るには、関心対象ではない列はバインドしないでください。さらに、列にバインドするホスト変数が適正なサイズであることも確認してください (例えば、列が 5 文字分の長さしかない場合、200 文字の文字ストリングはバインドしないでください)。

DBCLI 使用時のエラー原因の調査

DBCLI インターフェースの使用時にエラーが発生した場合は、以下のアクションを実行することをお勧めします。

- DBCLI 呼び出しが提供する戻りコードの確認。戻りコードは、各関数呼び出しの RETCODE パラメーターに返されます。 RETCODE パラメーターは、それぞれの呼び出しで最後のパラメーターでなければなりません。
- GETLASTERROR 関数への呼び出しで提供されるエラー情報の確認。この関数は、SQLCODE、SQLSTATE、およびテキストのエラー・メッセージを提供します。エラー・メッセージは、データベースや JDBC ドライバーに依存する場合があります。
- DBCLI の内部トレースを有効にしておくと役に立つことがあります。このトレースには、SYSLST、SYSLOG、あるいはその両方の診断情報が表示されます。トレースを有効にする手順は次のとおりです。
  - アプリケーションを実行するジョブで // SETPARM [SYSTEM] CLI$TRC=nnnn ステートメントを使用します。 CLI$TRC システム・パラメーターに指定できる値は次のとおりです。
    - OFF (デフォルト。指定しない場合と同じ) - トレースはオフです
    - SYSLST - トレースは SYSLST (ジョブ・リスト) に書き込まれます
    - SYSLOG - トレースは SYSLOG (コンソール) に書き込まれます

- BOTH または YES - トレースは SYSLOG および SYSLST に書き込まれます

– SETENVATTR 関数への呼び出しを使用して、プログラムに環境属性 ENVATTR-TRACE を設定します。この環境属性では、以下のいずれかの値を指定できます。

- TRACE-OFF (0) - トレースはオフです
- TRACE-SYSLST (1) - トレースは SYSLST (ジョブ・リスト) に書き込まれます
- TRACE-SYSLOG (2) - トレースは SYSLOG (コンソール) に書き込まれます
- TRACE-BOTH (3) - トレースは SYSLOG および SYSLST に書き込まれます

- DBCLI CICS/REXX サポート・ルーチン IESDBCIR をトレースするには、次のように DBCLI コマンドを定義することによってトレースをオンにできます。

  'DEFCMD DBCLI CALL = **CALLTRC** IESDBCIR (CICSLOAD'

  内部コマンド名として「CALLTRC」を指定すると、IESDBCIR 内でトレースがアクティブになります。トレース出力は、z/VSE コンソールと CICS リストに出されます。
- EZAAPI コマンドを使用して EZA ソケットの API トレースを有効にしておくと、役に立つ場合があります (資料「*IBM z/VSE TCP/IP サポート*」を参照)。このトレースには、EZA ソケット・インターフェースに関するトレース情報が表示されます。

DBCLI で使用される戻りコード

DBCLI 呼び出しは、RETCODE フィールドに以下の戻りコードのいずれかを返します。

記号	値	説明
EOK	0	エラーは発生しませんでした。
ECDLOAD	1	フェーズ IESDBCLA.PHASE をロードする際にロード障害が発生しました。
EHANDLE	2	関数に渡されたハンドルが無効です。
ENOMEM	3	使用可能なメモリーが不足しています。
EPARAM	4	パラメーターが無効または欠落しています。
EFUNCTION	5	指定された関数が無効です。
EINITFAILED	6	EZA TCP/IP インターフェースの初期化に失敗しました。TCPNAME または OPTION SYSPARM、および ADSNAME または SYSPARM EZA$PHS が正しく指定されているかどうかと、TCP/IP スタックが稼働中であるかどうかを確認してください。 **注:** また、 326 ページの『プログラミングの制限と要件』(特に、CICS トランザクションで DBCLI API を使用する際の注意事項) も確認してください。

記号	値	説明
ECONNFAILED	7	DBCliServer への接続に失敗しました。 IP アドレスまたはホスト名、およびポート番号を確認してください。
ENOTCONN	8	DBCliServer への接続が存在しません。
ESENDFAILED	9	送信操作が失敗しました。おそらく、DBCliServer への接続が切断されています。
ERECVFAILED	10	受信操作が失敗しました。おそらく、DBCliServer への接続が切断されています。
EPROTO	11	プロトコル違反が検出されました。このエラーを IBM サポートに報告してください。
EFAIL	12	一般障害が発生しました。詳しくは、GETLASTERROR 関数を使用して、エラー・メッセージ、SQLCODE、および SQLSTATE を確認してください。
ESQL	13	SQL エラーが発生しました。詳しくは、GETLASTERROR 関数を使用して、エラー・メッセージ、SQLCODE、および SQLSTATE を確認してください。
EATTRID	14	指定された属性が無効です。
EATTRLEN	15	属性の長さが無効です。
EATTRVAL	16	属性値が無効です。
EINDEX	17	パラメーターまたは列の索引が範囲外です。
ETYPE	18	データ型が無効です。
ESTATE	19	この関数は、接続またはステートメントの状態では使用できません。
ENOMOREDATA	20	これ以上のデータは取り出せません。

ほとんどの戻りコードに対して、DBCLI コードはテキストによるエラー・メッセージも提供します。 GETLASTERROR 関数を使用すれば、SQLCODE や SQLSTATE に加えてエラー・メッセージも取得できます。

バッチ照会ツール

DBCLI バッチ照会ツールは、バッチ区画で実行されているジョブ内から、データベースに接続し、SQL ステートメントを実行し、結果を検索することを可能にするバッチ・ツールです。プログラミングは不要です。

DBCLI バッチ照会ツールは、区画内で実行されているバッチ・ジョブから、次のように呼び出します。

```
// EXEC IESDBCLB[,PARM='key=value key=value ...]
[commands]
/*
```

PARM ステートメントと SYSIPT のいずれか (または両方) から入力を受け取ります。 PARM ステートメントの SYSIPT=nnn パラメーターで代替入力を指定できます。

出力はすべて SYSLST か、SET SYSLST コマンドと SET OUTPUT コマンドのいずれか (または両方) で指定した場所へ出力されます。

DBCLI バッチ照会ツールは、PARM ステートメントによって各種パラメーターを受け入れます。パラメーターはそれぞれブランクで区切ります。各パラメーターは key=value のペアで構成されます。サポートされているパラメーターのリストについては、下記の SET コマンドを参照してください。 SET コマンドでサポートされているキーワードはいずれも、PARM で指定することも可能です。

PARM の SYSIPT=nnn と SET コマンドのいずれか (または両方) で代替入力を指定していない限り、コマンドは一般的に SYSIPT から読み取られます。コマンドは複数行にわたってかまいません。その場合はセミコロン (;) で終了します。セミコロンはその行の最後の文字でなければなりません。

コマンドの形式は次のとおりです。

```
COMMAND [KEY=VALUE KEY=VALUE ...]
```

アスタリスク (*) で始まる行はコメント行であり、無視されます。空の行も無視されます。

以下のコマンドがサポートされます。他はすべて SQL ステートメントとして扱われ、データベースに渡されます (接続されている場合)。

SET

SET コマンドは以下の設定に使用できます。

SYMBOLS=YES|TRUE|ON|NO|FALSE|OFF (デフォルト: OFF)

SYSBOLS=YES (あるいは TRUE または ON) を使用すると、SYSIPT 内のシンボリック・パラメーター (&ABC など) は、JCL でのシンボリック・パラメーターの解決と同じ方法で解決されます。

SYSIPT=nnnn

コマンドを読み取る代替入力を使用できるようにします。ライブラリー・メンバー (DD:PRD2.CONFIG(COMMANDS.TXT) など) または VSAM/SAM ファイル (DD:MYFILE など) を指定できます。

SYSLST=nnnn

メッセージ処理用の代替出力を使用できるようにします。ライブラリー・メンバー (DD:PRD2.CONFIG(OUTPUT.TXT) など) または VSAM/SAM ファイル (DD:MYFILE など) を指定できます。

OUTPUT=nnnn

SQL 照会結果用の代替出力を使用できるようにします。ライブラリー・メンバー (DD:PRD2.CONFIG(OUTPUT.TXT) など) または VSAM/SAM ファイル (DD:MYFILE など) を指定できます。

ECHO=YES|TRUE|ON|NO|FALSE|OFF (デフォルト: YES)

ECHO=YES (あるいは TRUE または ON) を使用すると、コマンドはすべて SYSLST でエコー出力されます。 SQL コマンドも、指定された代替出力でエコー出力されます。

LOCALE=nnnn

使用するロケール (En_US.IBM-1047 など) を指定します。ロケールは、数値の書式設定 (小数点、1000 単位の区切り、グループ化) の他、日時表示に作用します。サポートされるロケールのリストについては、マニュアル

「*LE/VSE C Run-Time Programming Guide*」(SC33-6688) の付録 D、『Locales Supplied with LE/VSE C Run-Time』を参照してください。

CODEPAGE=nnnn
使用する EBCDIC コード・ページ (Cp1047 など) を指定します。

AUTOCOMMIT=YES|TRUE|ON|NO|FALSE|OFF (デフォルト: OFF)
AUTOCOMMIT=YES (あるいは TRUE または ON) を指定すると、各 SQL ステートメントの実行直後に、DBCLI 対話式端末でテーブルに対して行われた更新がすべて自動的にコミットされます。 AUTOCOMMIT=NO (あるいは FALSE または OFF またはパラメーターを指定しない) の場合、更新は即時にコミットされず、ユーザーがデータベースを正常切断するか、PF8 (COMMIT) キーを押したときにのみコミットされます。 PF7 (ROLLBACK) は変更をロールバックします。

ERRORMODE=IGNORE|CONTINUE|STOP (デフォルト: STOP)
ERRORMODE=STOP の場合、コマンドの失敗により処理が停止し、それ以上コマンドは実行されません。 ERRORMODE=IGNORE または CONTINUE の場合、処理は次のコマンドへ続行されます。

FORMAT=LIST|TABLE|COLUMN (デフォルト: TABLE)
FORMAT=TABLE または LIST の場合、照会結果は表形式にフォーマットされます。表の幅が可能な出力レコード・サイズ (プリンターの場合は 132) を超える可能性がある場合は、COLUMNS フォーマットが代わりに使用されます。 FORMAT=COLUMN の場合、各列は別々の行に出力されます。

MAXCOLBUFFERSIZE=nnn (デフォルト: 65536)
照会結果の取り出し時に列バッファーとして使用される最大サイズ (バイト) を指定します。列がより大きいバッファーを必要とする場合、その列は取り出すことができず、その列には結果セット表示内で「TOO LONG」のマークが付けられます。

MAXCOLDISPLAYSIZE=nnn (デフォルト: 64)
列が表示される最大幅 (文字数) を指定します。列がより多くの文字数を必要とする場合、表示されるデータは切り捨てられます。指定可能な最大値は 512 です。これより大きい値は (メッセージなしで) 無視され、512 が使用されます。

CALL=YES|TRUE|ON|NO|FALSE|OFF (デフォルト: OFF)
CALL=YES (あるいは TRUE または ON) は、次の SQL コマンドがストアード・プロシージャー呼び出しであることを指示します。この設定は、各 SQL コマンドの後に OFF に戻されます。

TRACE=OFF|SYSLST|SYSLOG|BOTH|DBCLI
トレースを有効にします。トレースは、IBM サポート担当員が要求した場合にのみオンにしてください。
- TRACE=OFF: トレースはオフです (デフォルト)。
- TRACE=SYSLST: SYSLST へのトレースが有効です。
- TRACE=SYSLOG: SYSLOG へのトレースが有効です。
- TRACE=BOTH: SYSLST と SYSLOG へのトレースが有効です。
- TRACE=SCREEN: SYSLST と SYSLOG への画面トレースが有効です。

- TRACE=DBCLI: SYSLST と SYSLOG への DBCLI トレースが有効です。

CONNECT

CONNECT コマンドは、DBCLI サーバーおよびデータベースに接続します。

SERVER=nnnn (必須)
接続する DBCLI サーバーのホスト名または IP アドレスを指定します。

PORT=nnnn (オプション、デフォルト: 16178)
使用するポート番号を指定します。

DBNAME=nnnn (必須)
接続するデータベース名 (別名) を指定します。

USER=nnnn (オプション)
データベースでの認証に使用するユーザー ID を指定します。

PASSWORD=nnnn (オプション)
データベースでの認証に使用するパスワードを指定します。

SSLTYPE (オプション)
使用する SSL タイプ (SSL30 や TLS31 など) を指定します。

SSLKEYRING=nnnn (SSL 使用時に必須)
SSL 鍵と証明書を含む SSL 鍵リング・ライブラリーを指定します。

SSLKEYNAME=nnnn (SSL 使用時に必須)
使用する SSL 鍵名を指定します。

SSLCIPHERS=nnnn (オプション)
使用する SSL 暗号スイートを指定します。このパラメーターを指定しないと、SSL 実装でサポートされているすべての暗号スイートが使用されます。

以下の暗号スイートが TCP/IP for z/VSE および OpenSSL でサポートされています。

```
C027 for TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 OPENSSL ONLY
C014 for TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA  - OPENSSL ONLY
C013 for TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA  - OPENSSL ONLY
C012 for TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA - OPENSSL ONLY
  6B for TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 - OPENSSL ONLY
  67 for TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 - OPENSSL ONLY
  39 for TLS_DHE_RSA_WITH_AES_256_CBC_SHA    - OPENSSL ONLY
  33 for TLS_DHE_RSA_WITH_AES_128_CBC_SHA    - OPENSSL ONLY
  16 for SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA   - OPENSSL ONLY
  3D for TLS_RSA_WITH_AES_256_CBC_SHA256     - OPENSSL ONLY
  3C for TLS_RSA_WITH_AES_128_CBC_SHA256     - OPENSSL ONLY
  3B for TLS_RSA_WITH_NULL_SHA256 - DEPRECATED,OPENSSL ONLY
  35 for TLS_RSA_WITH_AES_256_CBC_SHA
  2F for TLS_RSA_WITH_AES_128_CBC_SHA
  0A for RSA1024_3DESCBC_SHA                 - DEPRECATED
  09 for RSA1024_DESCBC_SHA                  - DEPRECATED
  08 for RSA512_DES40CBC_SHA                 - DEPRECATED
  02 for RSA512_NULL_SHA                     - DEPRECATED
  01 for RSA512_NULL_MD5                     - DEPRECATED
```

SSLSESSIONTIMEOUT=nnnn (オプション。デフォルト: 86400)
SSL セッション・タイムアウト (秒) を指定します。

DISCONNECT

DISCONNECT コマンドは、DBCLI サーバーおよびデータベースから切断します。

COMMIT

COMMIT コマンドは、現行作業単位で行われたすべての変更をコミットします。

ROLLBACK

CONNECT コマンドは、現行作業単位で行われたすべての変更をロールバックします。

ジョブの例:

```
* $$ JOB JNM=RUNDBCLI,DISP=D,CLASS=4
* $$ LST DISP=D,CLASS=Q,PRI=3
// JOB RUNDBCLI
// LIBDEF *,SEARCH=(PRD2.CONFIG,PRD1.BASE,PRD2.TCPIPC)
// EXEC IESDBCLB,PARM='SYMBOLS=YES ECHO=ON'

CONNECT SERVER=my.database.server.com DBNAME=SAMPLE
        USER=db2user PASSWORD=password;

SELECT EMPNO,FIRSTNME,LASTNAME,SALARY,BONUS FROM EMPLOYEE;

DISCONNECT;
/*
/&
$$ EOJ
```

このジョブは、以下のような出力を生成します。

```
// JOB RUNDBCLI
// LIBDEF *,SEARCH=(PRD2.CONFIG,PRD1.BASE,PRD2.TCPIPC)
// EXEC IESDBCLB,PARM='SYMBOLS=YES ECHO=ON'
1S54I  PHASE IESDBCLB IS TO BE FETCHED FROM PRD1.BASE
DBCLI BATCH QUERY TOOL

CONNECT SERVER= my.database.server.com DBNAME=SAMPLE
        USER=db2user PASSWORD=(PASSWORD SUPPRESSED);
INFO: CONNECTED TO SERVER 'my.database.server.com' DBNAME 'SAMPLE'.
INFO: DATABASE PRODUCT 'DB2/NT' VERSION 'SQL09075'
INFO: LAST RC=0

SELECT EMPNO,FIRSTNME,LASTNAME,SALARY,BONUS FROM EMPLOYEE;
INFO: STATEMENT HAS BEEN EXECUTED, IT PRODUCED THE FOLLOWING RESULT SET:

EMPNO   FIRSTNME     LASTNAME        SALARY       BONUS
CHAR(6) VARCHAR(12)  VARCHAR(15)     DECIMAL(9,2) DECIMAL(9,2)
------- ------------ --------------- ------------ ------------
000010  CHRISTINE    HAAS               152750.00      1000.00
000020  MICHAEL      THOMPSON            94250.00       800.00
000030  SALLY        KWAN                98250.00       800.00
000050  JOHN         GEYER               80175.00       800.00
000060  IRVING       STERN               72250.00       500.00
000070  EVA          PULASKI             96170.00       700.00
000090  EILEEN       HENDERSON           89750.00       600.00
000100  THEODORE     SPENSER             86150.00       500.00
000110  VINCENZO     LUCCHESSI           66500.00       900.00
000120  SEAN         O'CONNELL           49250.00       600.00
000130  DELORES      QUINTANA            73800.00       500.00
000140  HEATHER      NICHOLLS            68420.00       600.00
000150  BRUCE        ADAMSON             55280.00       500.00
000160  ELIZABETH    PIANKA              62250.00       400.00
000170  MASATOSHI    YOSHIMURA           44680.00       500.00
```

```
000180  MARILYN      SCOUTTEN            51340.00       500.00
000190  JAMES        WALKER              50450.00       400.00
000200  DAVID        BROWN               57740.00       600.00
000210  WILLIAM      JONES               68270.00       400.00
000220  JENNIFER     LUTZ                49840.00       600.00
000230  JAMES        JEFFERSON           42180.00       400.00
000240  SALVATORE    MARINO              48760.00       600.00
000250  DANIEL       SMITH               49180.00       400.00
000260  SYBIL        JOHNSON             47250.00       300.00
000270  MARIA        PEREZ               37380.00       500.00
000280  ETHEL        SCHNEIDER           36250.00       500.00
000290  JOHN         PARKER              35340.00       300.00
000300  PHILIP       SMITH               37750.00       400.00
000310  MAUDE        SETRIGHT            35900.00       300.00
000320  RAMLAL       MEHTA               39950.00       400.00
000330  WING         LEE                 45370.00       500.00
000340  JASON        GOUNOT              43840.00       500.00
200010  DIAN         HEMMINGER           46500.00      1000.00
200120  GREG         ORLANDO             39250.00       600.00
200140  KIM          NATZ                68420.00       600.00
200170  KIYOSHI      YAMAMOTO            64680.00       500.00
200220  REBA         JOHN                69840.00       600.00
200240  ROBERT       MONTEVERDE          37760.00       600.00
200280  EILEEN       SCHWARTZ            46250.00       500.00
200310  MICHELLE     SPRINGER            35900.00       300.00
200330  HELENA       WONG                35370.00       500.00
200340  ROY          ALONZO              31840.00       500.00

INFO: ROWCOUNT: 42
INFO: LAST RC=0

DISCONNECT;
INFO: DISCONNECT SUCCESSFULL.
INFO: LAST RC=0

1S55I  LAST RETURN CODE WAS 0000
EOJ RUN       MAX.RETURN CODE=0000
```

対話式照会ツール

DBCLI 対話式照会ツールは、3270 端末から、データベースに接続し、SQL ステートメントを実行し、結果を検索することを可能にする、3270 端末ベースのユーザー・インターフェースです。これらを完全に対話式に行うことができます。プログラミングは不要です。

DBCLI 対話式照会ツールを開始するには、CICS 端末でトランザクション IDBT を開始します。

IDBT トランザクションはオプション・パラメーターを受け入れます。

```
IDBT [KEY=VALUE][,KEY=VALUE]....
```

パラメーター名は大/小文字が区別されません。複数のパラメーターはコンマまたはブランクで区切ります。

以下のパラメーターがサポートされています。

UPPERCASE=YES|TRUE|ON|NO|FALSE|OFF (デフォルト: OFF)
UPPERCASE=YES (あるいは TRUE または ON) を指定すると、端末入力と端末出力はすべて自動的に大文字に変換されます。これは、小文字が正しく表示さ

れない日本語ユーザーに役立ちます。一部のデータベースでは大/小文字が区別されますが、上段シフト・モードがオンの場合は大文字のみが入力可能であることに注意してください。

LOCALE=nnnnn
使用するロケール (En_US.IBM-1047 など) を指定します。ロケールは、数値の書式設定 (小数点、1000 単位の区切り、グループ化) の他、日時表示に作用します。サポートされるロケールのリストについては、マニュアル「*LE/VSE C Run-Time Programming Guide*」(SC33-6688) の付録 D、『Locales Supplied with LE/VSE C Run-Time』を参照してください。

AUTOCOMMIT=YES|TRUE|ON|NO|FALSE|OFF (デフォルト: OFF)
AUTOCOMMIT=YES (あるいは TRUE または ON) を指定すると、各 SQL ステートメントの実行直後に、DBCLI 対話式端末でテーブルに対して行われた更新がすべて自動的にコミットされます。 AUTOCOMMIT=NO (あるいは FALSE または OFF またはパラメーターを指定しない) の場合、更新は即時にコミットされず、ユーザーがデータベースを正常切断するか、PF8 (COMMIT) キーを押したときにのみコミットされます。 PF7 (ROLLBACK) は変更をロールバックします。

MAXCOLBUFFERSIZE=nnn (デフォルト: 65536)
照会結果の取り出し時に列バッファーとして使用される最大サイズ (バイト) を指定します。列がより大きいバッファーを必要とする場合、その列は取り出すことができず、その列には結果セット表示内で「TOO LONG」のマークが付けられます。

MAXCOLDISPLAYSIZE=nnn (デフォルト: 64)
列が表示される最大幅 (文字数) を指定します。列がより多くの文字数を必要とする場合、表示されるデータは切り捨てられます。指定可能な最大値は 512 です。これより大きい値は (メッセージなしで) 無視され、512 が使用されます。

TRACE=OFF|SYSLST|SYSLOG|BOTH|SCREEN|DBCLI
トレースを有効にします。トレースは、IBM サポート担当員が要求した場合にのみオンにしてください。
- TRACE=OFF: トレースはオフです (デフォルト)。
- TRACE=SYSLST: SYSLST へのトレースが有効です。
- TRACE=SYSLOG: SYSLOG へのトレースが有効です。
- TRACE=BOTH: SYSLST と SYSLOG へのトレースが有効です。
- TRACE=SCREEN: SYSLST と SYSLOG への画面トレースが有効です。
- TRACE=DBCLI: SYSLST と SYSLOG への DBCLI トレースが有効です。

対話式照会ツールでは、3270 ベースの画面が表示されます。 446 ページの図 151 の初期画面では、接続先の DBCliServer とデータベース別名の指定方法を示しています。必要な入力を行い、Enter を押します。

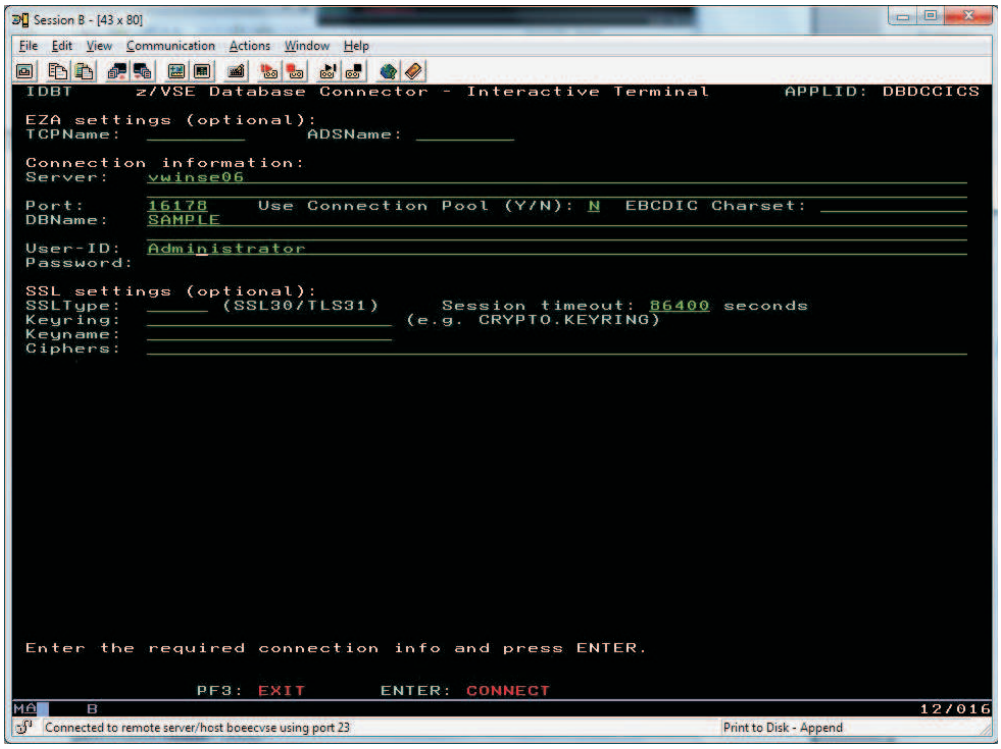


図 *151.* 対話式照会ツールの初期画面

447 ページの図 152 にある次の画面には、実行したい SQL ステートメントを入力できます。

PF3 前の画面に戻り、データベースから切断し、ここまでに行った変更をすべてコミットします。

PF4 接続しているデータベースがサポートする表をリストします。

PF5 接続しているデータベースがサポートする列をリストします。

PF7 このセッションで行った変更をすべてロールバックします。

PF8 変更をすべてコミットします。

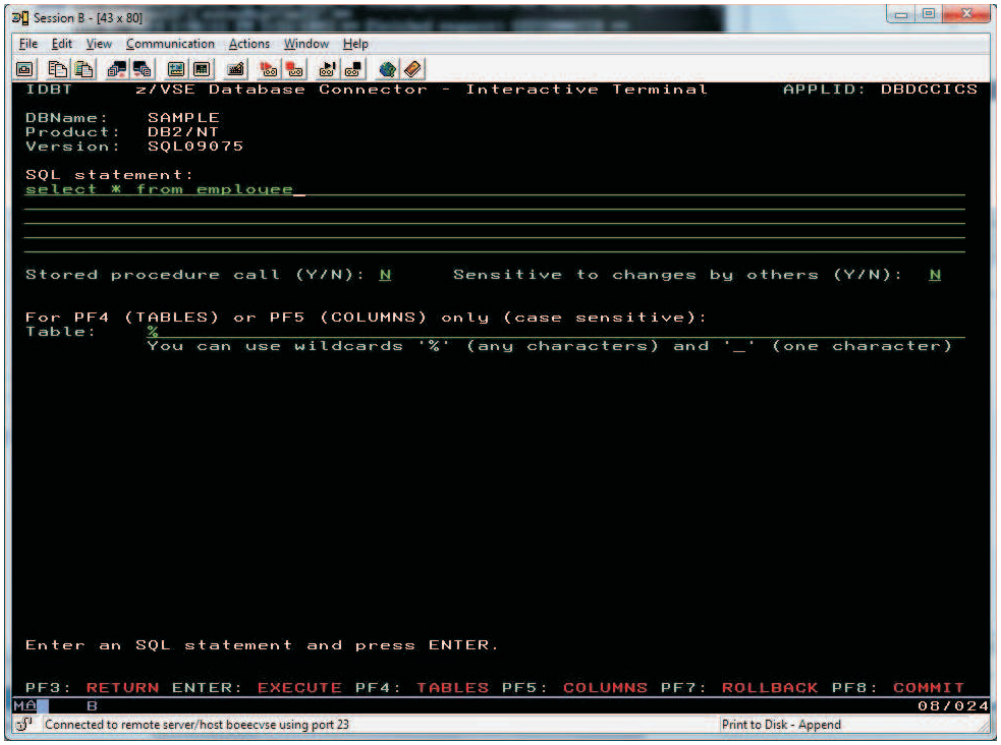


図 *152.* 対話式照会ツール - *SQL* ステートメント画面

SQL ステートメントを実行して結果セットが作成されると、結果セットは図 153 のように表示されます。

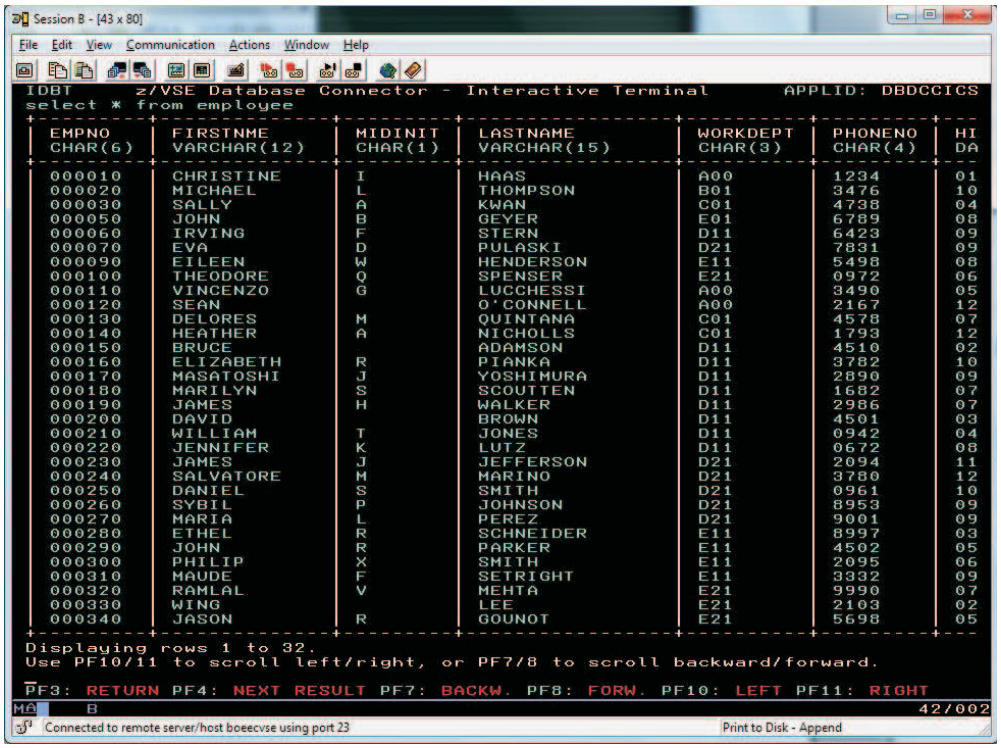


図 *153.* 対話式照会ツール - *SQL* ステートメント画面 - 結果

以下を押すことによってデータをスクロールできます。

PF3 前の画面に戻ります。

PF4 SQL ステートメントで複数の結果セットが生成された場合に、次の結果に移動します。

PF7 後方

注**:** 後方スクロールは、当該結果セットでスクロールがサポートされている場合にのみ可能です。

PF8 前方

PF10 左方

PF11 右方

第 23 章 データにアクセスするための Db2 ベース・コネクターの使用

このトピックでは、Db2 ベース・コネクターを使用して、VSAM データ・セットと DL/I データベースにアクセスする方法について説明します。

Db2 ベース・コネクターは、以下の製品を使用します。

- z/VSE ホスト上:
  - Db2 Server for VSE

    注: Db2 Server for VSE Client Edition は、Db2 ストアード・プロシージャーを z/VSE 上で実行できないため、Db2 Server for VSE Client Edition は、Db2 ベース・コネクターと使用するには十分ではありません。
  - Db2 ストアード・プロシージャー機能 (Db2 Server for VSE のバージョン 6 以上で使用可能)
  - VSAMSQL CLI
  - DL/I VSE
- 物理/論理中間層上:
  - Db2 Connect
  - ODBC、JDBC、または CLI

Db2 ベース・コネクターの概要については、 95 ページの『Db2 ベース・コネクター の概要』を参照してください。

このトピックに含まれるのは次のとおりです。

- 『Db2 ストアード・プロシージャーの使用法』 (ストアード・プロシージャー・サーバーの説明を含む)
- 451 ページの『Db2 ストアード・プロシージャーによる VSAM データへのアクセス』
- 459 ページの『Db2 ストアード・プロシージャーによる DL/I データへのアクセス』

Db2 ストアード・プロシージャーの使用法

Db2 ストアード・プロシージャーは、ユーザー自身が作成し、コンパイルしてライブラリーにカタログし、次に Db2 に定義できるプログラム です。ユーザーの Db2 ストアード・プロシージャーは、次のものから呼び出すことができます。

- Web クライアント
- 3 層の z/VSE 環境の物理/論理中間層
- ローカル・バッチ・プログラム

Db2 ストアード・プロシージャーは、任意の LE (言語処理環境) 対応言語で作成できます。Db2 ストアード・プロシージャー内の 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、z/VSE ホスト上で VSAM データまたは DL/I データのどちらがアクセスされるかによって異なります。

以下に、Db2 ストアード・プロシージャーを使用した場合の主な利点について説明します。

- データベース・マネージャーが複数ユーザー・モードで実行しているとき、ローカル・アプリケーションまたはリモート DRDA アプリケーションが、1 つの Db2 ストアード・プロシージャーを呼び出すことができます。 Db2 ストアード・プロシージャーによって出される SQL ステートメントは z/VSE ホストにはローカルであるので、分散ステートメントによる高ネットワーク・コストを生じません。 代わりに、Db2 ストアード・プロシージャーに入っているすべてのステートメントを処理するために、単一ネットワークの *send* および *receive* 操作を使用できます。
- Db2 ストアード・プロシージャーを使用して、データベース設計の詳細を、Web クライアントまたは物理/論理中間層上で実行されているアプリケーション・プログラムから隠すことができます。
- データベースを変更する場合は、Db2 ストアード・プロシージャーのみ (アプリケーション・プログラムではなく) を変更すれば済みます。
- Db2 ストアード・プロシージャーを使用して、機密データを、特定のアプリケーション・プログラムから隠すことができます。
- ビジネス・ロジックは z/VSE ホストでカプセル化でき、このビジネス・ロジックを多くのアプリケーション・プログラムに組み込む必要はありません。
- Db2 ストアード・プロシージャー・アプリケーションを z/VSE ホストで保守する環境を保守するほうが、多くの Web クライアントまたは物理/論理中間層に分散して保守するよりも簡単です。

ストアード・プロシージャー・サーバーのグループ化

ストアード・プロシージャー・サーバー は、分離された静的または動的区画に入っており、単一 Db2 Server for VSE の専用にする必要があります。 前のトピックで説明した Db2 ストアード・プロシージャーは、LE に準拠するストアード・プロシージャー・サーバーの制御のもとで実行されます。

ご使用のストアード・プロシージャー・サーバーをグループ化することにより、複数の区画に、データベース・ワークロードを分散できます。これは、ある Db2 ストアード・プロシージャーが Db2 Server for VSE を常時使用可能にしておかなければならない場合に便利です。この場合、1 つのストアード・プロシージャー・サーバー・グループを、Db2 ストアード・プロシージャーの専用にすることができます。次のような場合、ほかの Db2 ストアード・プロシージャーが、ほかのストアード・プロシージャー・サーバー・グループを共用できます。

- 一部の Db2 ストアード・プロシージャーに、特殊な要件 (例えば、大容量の仮想記憶が必要) がある場合。
- ほかの Db2 ストアード・プロシージャーが、その他の Db2 ストアード・プロシージャーで必要がないリソースにアクセスする必要がある場合。

ストアード・プロシージャー・サーバーをグループ化できる機能があることにより、データベース管理者は、環境を定義するときに柔軟性を持つことができ、さらに、この機能はシステムのチューニングにも役立ちます。

Db2 ストアード・プロシージャーを使用する際のプログラミング要件

Web クライアントまたは物理/論理中間層の場合: JDBC または ODBC/CLI (SQL プリコンパイラーは不要) が必要です。 ただし、Db2 ストアード・プロシージャーを呼び出すために、組み込み SQL が入っている別のプログラム言語を使用する場合は、SQL プリコンパイラーが必要です。

z/VSE ホストの場合: Db2 ストアード・プロシージャー に SQL 固有の呼び出しが入っていない場合は、SQL プリコンパイラーは必要ありません。

VSAMSQL CLI は小さなオブジェクト・モジュールから成っており、ユーザーは、これを、VSAMSQL CLI を使用するユーザーの Db2 ストアード・プロシージャーのそれぞれにリンクする必要があります。このオブジェクト・モジュールは、ユーザーの Db2 ストアード・プロシージャー が CLI (コール・レベル・インターフェース) を使用して、データがリレーショナルである場合と同様に VSE/VSAM データにアクセスできるようにする「スタブ」です。

Db2 ストアード・プロシージャーは、LE 準拠でなければなりません。 言語処理環境は、z/VSE で実行されるコンパイラーを使用して生成されたアプリケーションに対して、前提条件となるランタイム環境です。

以下は、ユーザーのアプリケーション・プログラムから使用できる最も重要なインターフェースです。

- 物理/論理中間層プラットフォーム上の ODBC (Open DataBase Connectivity)。
- z/VSE ホスト上で実行されている Db2 ストアード・プロシージャー内の CLI (コール・レベル・インターフェース)。

Db2 ストアード・プロシージャーによる VSAM データへのアクセス

このトピックでは、Db2 ストアード・プロシージャーを使用して VSE/VSAM データにアクセスする方法について説明します。

452 ページの図 154 では VSAM データへのアクセスのみが示されていますが、同じ Db2 ストアード・プロシージャーを使用して、Db2 データおよび DL/I データにアクセスできます。

マップされた VSAM データにアクセスするには、ユーザーのアプリケーション・プログラムで、IBM Db2 コール・レベル・インターフェースを基にした VSAMSQL コール・レベル・インターフェース (CLI) を使用します。VSAMSQL CLI を使用することにより、次のトピックで説明するように、アプリケーション・プログラムが、Db2 ストアード・プロシージャー内から、VSAM データに SQL に似た呼び出しを出すことができます。

- 452 ページの『概説: Db2 ストアード・プロシージャーを介して VSAM データにアクセスする』には、Db2 ストアード・プロシージャーにより、VSAMSQL CLI を使用して、マップされた VSAM データを表示する方法の概要の説明があります。
- 454 ページの『コール・レベル・インターフェースの使用: リクエスター上のアクティビティー』には、Db2 ストアード・プロシージャーを使用し、VSAMSQL

CLI を介してマップ済み VSAM データを表示するためにクライアント上で ユーザーが実行しなければならないアクティビティーの説明があります。

- 454 ページの『コール・レベル・インターフェースの使用: z/VSE ホスト上のアクティビティー』には、Db2 ストアード・プロシージャーを使用し、VSAMSQL CLI を介してマップ済み VSAM データを表示するために *z/VSE* ホスト 上で ユーザーが実行しなければならないアクティビティーの説明があります。
- 456 ページの『VSAMSQL コール・レベル・インターフェースを使用するときのプログラム・フロー』には、Db2 ストアード・プロシージャーが、マップ済み VSAM データに VSAM-CLI 更新を実行するときの典型的なプログラム・フローの説明があります。
- 457 ページの『VSAMSQL コール・レベル・インターフェースによってサポートされる SQL ステートメント』には、Db2 ストアード・プロシージャーが、マップ済み VSAM データに VSAM-CLI 更新を実行するときにユーザーが使用できる SQL ステートメントがリストされています。

注: アプリケーション・プログラムが VSAMSQL CLI を使用する方法の実際的な例については、 234 ページの『ステップ 2: VSAMSQL CLI 環境の初期化』を参照してください。

概説: Db2 ストアード・プロシージャーを介して VSAM データにアクセスする

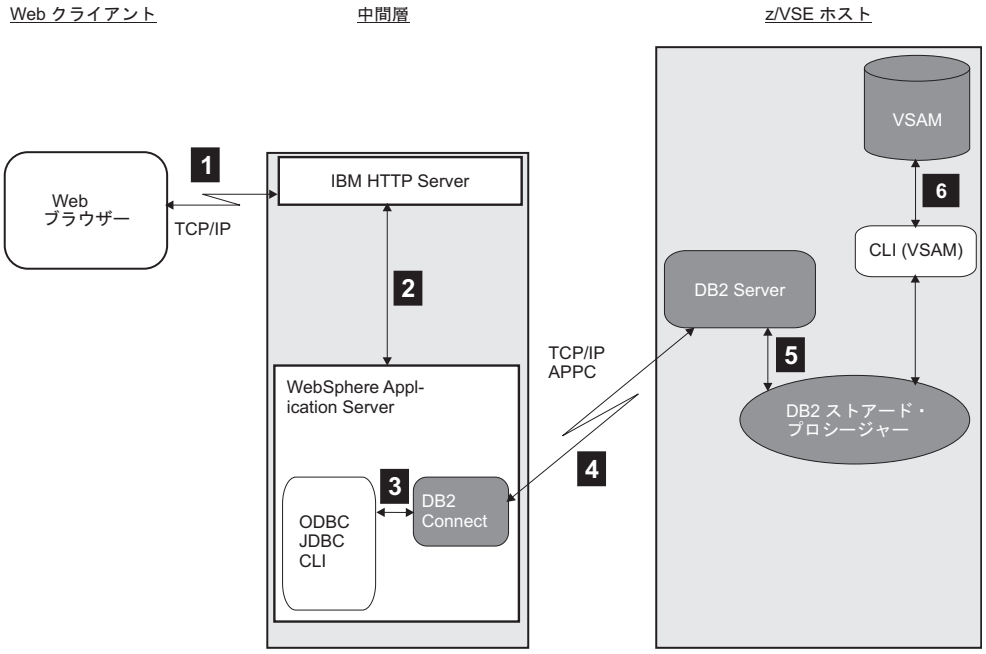


図 *154. VSAM* データにアクセスするための *Db2* ストアード・プロシージャーの使用

以下の各リスト項目の番号は、図 154 に示されているステップを説明しています。

1 クライアントの Web ブラウザーが、物理/論理中間層で実行されている IBM HTTP Server にある HTML ページを要求します。

2 IBM HTTP Server は WebSphere Application Server を呼び出して、HTML ページにある要求 (例えば、アプレットまたはサーブレットに対する要求) を求めます。

3 インターフェース (ODBC、JDBC、または CLI) が、Db2 Connect を介して Db2 Server for VSE と通信します。

4 Db2 Connect は、DRDA (Distributed Relational Database Architecture™) を介して Db2 Server for VSE と通信します。ここで使用される基礎のプロトコルは APPC または TCP/IP のどちらでもかまいません。

5 Db2 Server for VSE は、ストアード・プロシージャー・サーバーを使用して、Db2 ストアード・プロシージャーの実行を管理します。

6 Db2 ストアード・プロシージャーは、VSAMSQL CLI を介して、z/VSE ホストに保管されている VSAM データにアクセスできます。Db2 ストアード・プロシージャーには、このアクセスを実行するために使用される VSAM *CLI* 関数 (455 ページの表 9 に説明があります) への呼び出しが入っています。

次に、ステップ 1 から 6 の逆が実行されます。Db2 Server for VSE は、結果をリクエスターに受け渡します。

452 ページの図 154 に示されているように、Db2 ストアード・プロシージャーを使用して z/VSE ホストに保管されている VSAM データにアクセスするには、以下のものが必要です。

- 物理/論理中間層上:
  - Web サーバー (例えば、IBM HTTP Server)
  - WebSphere Application Server
  - Db2 Connect
  - ODBC または CLI
- *z/VSE* ホスト上:
  - Db2 Server for VSE
  - VSAMSQL CLI

以下に、Db2 ストアード・プロシージャーを使用してマップ済みの VSAM データにアクセスするアプリケーション・プログラムを開発するために実行する必要がある一般的なステップを示します。

1. リクエスターと z/VSE ホストとの間の Db2 接続を確立します。
2. アプリケーション・プログラム、およびご使用のマップを保管するために使用する VSAM クラスターを設計します。
3. VSAM クラスター用のマップとビューを定義します (117 ページの『第 12 章 リレーショナル構造への VSE/VSAM データのマッピング』を参照)。
4. VSAM および Db2 に対するロジックと要求を組み込んだ Db2 ストアード・プロシージャーを作成します (IBM 資料「*Db2 Server (VSE および VM 版) データベース管理*」(SC88-8636) および「*Db2 Server (VSE および VM 版) アプリケーション・プログラミング*」(SC88-8637) を参照)。
5. Db2 に、ユーザーの Db2 ストアード・プロシージャーのエントリーを作成します。

6. Db2 ストアード・プロシージャーへのリクエスター呼び出しを作成します。
7. アプリケーション・プログラムをテストし、実行します。

コール・レベル・インターフェースの使用: リクエスター上のアクティビティー

リクエスター上で、JDBC または ODBC/CLI などのリレーショナル・データベース・インターフェースにインプリメントされているストアード・プロシージャー呼び出しを使用して、Db2 ストアード・プロシージャーを呼び出すことができます。次に、これらのリレーショナル・データベースのストアード・プロシージャー・インターフェースのサマリーを示します。

JDBC

```
String sql="Call <proc_name> (?,?,?,?,?,?)";
statement = con.prepareCall (sql);
statement.execute ();
```

ODBC/CLI

```
CALL procedure_name (?,?,?,?,...)
SQLExecDirect() または
SQLPrepare() (SQLExecute() が続く)
```

組み込み **SQL**

```
CALL proc_name [(parm1[:parmind1],...,parmn[:parmindn])] または
CALL proc_name USING DESCRIPTOR :sqlda
```

上記のインターフェースの詳細な説明については、ストアード・プロシージャーの呼び出し方法が説明されている以下の資料を参照してください。

- Microsoft ODBC SDK Programmer's Reference
- IBM DB2 Universal Database™ Call Level Interface Guide and Reference (S10J-8159)
- IBM Embedded SQL Programming Guide (S10J-8158)
- JDBC データ・アクセス API。次の Web サイトを参照してください。

  `http://www.oracle.com/technetwork/java/javase/jdbc/`

コール・レベル・インターフェースの使用: z/VSE ホスト上のアクティビティー

z/VSE ホスト上で、VSAMSQL コール・レベル・インターフェース (CLI) は、Db2 ストアード・プロシージャーを使用してマップ済み VSAM データにアクセスするための C プログラム関数を提供します。 このインターフェースは、リレーショナル・データベースで表を使用するのと同じ方法で、VSAM ファイルにアクセスします。 したがって、Db2 ストアード・プロシージャーを使用し VSAMSQL インターフェースを介してアクセスしたいマップ済みデータがある VSAM クラスターのそれぞれに対して、リレーショナル・ビューを定義する必要があります。

すべての VSAMSQL CLI C プログラム関数には VSAMSQL という接頭部があり、Db2 CLI 関数 (Db2 CLI 関数には接頭部 SQL がある) と同じ構文と機能性を持っています。

注: CLI 関数を使用するには、C ヘッダー・ファイル *IESVSQL.h* をプログラム・ソースに組み込む必要があります。オブジェクト・ファイル *IESVSQLO.Obj* は、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。 *IESVSQL.h* および *IESVSQLO.Obj* の両方とも、VSE ライブラリー PRD1.BASE に入っています。

以下の CLI 関数がサポートされています。

表 *9.* マップ済み *VSAM* データにアクセスするために使用できる *CLI* 関数

CLI 関数	説明
VSAMSQLAllocConnect *	接続ハンドルの割り振り
VSAMSQLAllocEnv *	環境ハンドルの割り振り
VSAMSQLAllocHandle	環境、接続、ステートメント、または、記述子のそれぞれのハンドルの割り振り
VSAMSQLAllocStmt *	ステートメント・ハンドルの割り振り
VSAMSQLBindCol	アプリケーション変数への列のバインド
VSAMSQLBindParameter	バッファーへのパラメーター・マーカーのバインド
VSAMSQLCloseTable	指定した表 (クラスター) のクローズ
VSAMSQLColAttribute *	列属性の戻り
VSAMSQLColAttributes	列属性の取得
VSAMSQLColumns	表の列情報の取得
VSAMSQLDescribeCol	列の一連の属性の戻り
VSAMSQLError	エラー情報のリトリーブ
VSAMSQLExecDirect	ステートメントの直接の実行
VSAMSQLExecute	ステートメントの実行
VSAMSQLFetch	次の行の取り出し
VSAMSQLFreeConnect *	接続ハンドルの解放
VSAMSQLFreeEnv *	環境ハンドルの解放
VSAMSQLFreeHandle	ハンドル・リソースの解放
VSAMSQLFreeStmt *	ステートメント・ハンドルの解放 (またはリセット)
VSAMSQLGetDiagRec	診断レコードの複数のフィールド設定値の取得
VSAMSQLNumParams	SQL ステートメント内のパラメーター数の取得
VSAMSQLNumResultCols	結果列の数の取得
VSAMSQLPrepare	ステートメントの準備
VSAMSQLPrimaryKeys	表の主キー列の取得
VSAMSQLRowCount	行カウントの取得
VSAMSQLSetParam *	バッファーへのパラメーター・マーカーのバインド
VSAMSQLTables	表情報の取得

注**:** アスタリスク (*) が付いている関数は、最新の関数の中に組み込まれています。 詳しくは、IBM 資料「*Db2 Universal Database Call Level Interface, Guide and Reference*」(S10J-1859) を参照してください。

CLI 関数の構文の例 - VSAMSQLCloseTable

以下に、オンライン・ドキュメンテーションで、CLI 関数 *VSAMSQLCloseTable* がどのように説明されているかについて、例を示します。

VSAMSQLCloseTable - 指定した表 (クラスター) のクローズ

目的 VSAMSQLCloseTable() は、指定した VSAM クラスターを閉じます。

構文

```
VSAMSQLRETURN   VSAMSQLCloseTable (VSAMSQLHENV       henv,
                                   VSAMSQLCHAR*      szTableName,
                                   VSAMSQLSMALLINT   cbTableName);
```

関数引数 VSAMSQLCloseTable 引数

データ・タイプ	引数	使用	説明
VSAMSQLHENV	henv	入力	環境ハンドル
VSAMSQLCHAR*	szTableName	入力	クローズする表名
VSAMSQLSMALLINT	cbTableName	入力	表名の長さまたは VSAMSQL_NTS

戻りコード

```
VSAMSQL_SUCCESS
VSAMSQL_ERROR
VSAMSQL_INVALID_HANDLE
```

診断 VSAMSQLCloseTable SQLSTATEs

SQLSTATE	説明	解説
HY009	無効な引数値	引数は無効でした。

VSAMSQL コール・レベル・インターフェースを使用するときのプログラム・フロー

以下に示すのは、マップ済み VSAM データのために VSAMSQL CLI を使用するときのプログラム・ステートメントの典型的なフローです。

```
VSAMSQLRETURN  rc;         // Return Code
VSAMSQLHENV    hEnv;       // Environment Handle
VSAMSQLHDBC    hDBC;       // Connection Handle
VSAMSQLHSTMT   hStmt;      // statement Handle

//allocate Environment
rc = VSAMSQLAllocHandle(VSAMSQL_HANDLE_ENV,VSAMSQL_NULL_HANDLE,&hEnv);
//allocate Connection
rc = VSAMSQLAllocHandle(VSAMSQL_HANDLE_DBC,hEnv,&hDBC);
//allocate Statement
rc = VSAMSQLAllocHandle(VSAMSQL_HANDLE_STMT,hDBC,&hStmt);

// Prepare a Statement
rc = VSAMSQLPrepare(hStmt,
     "UPDATE VSESP.USER.CATALOG/VSAM.DISPLAY.DEMO.CLUSTER/MAP1 SET EMAIL=?, AGE=?"
     "WHERE NAME=?",VSAMSQL_NTS);

// Query the number of Parameters
rc = VSAMSQLNumParams(hStmt,&Num);

// Bind local Variables/Values to the Statement
rc = VSAMSQLBindParameter(hStmt,1,VSAMSQL_PARAM_INPUT,
     VSAMSQL_C_CHAR,VSAMSQL_VARCHAR,5,0,"Hugo",5,NULL);
 ...

// Execute the Statement
rc = VSAMSQLExecute(hStmt);
 ...
rc = VSAMSQLFreeHandle(VSAMSQL_HANDLE_STMT,hStmt);
rc = VSAMSQLFreeHandle(VSAMSQL_HANDLE_DBC,hDBC);
rc = VSAMSQLFreeHandle(VSAMSQL_HANDLE_ENV,hEnv);
```

図 *155. VSAMSQL CLI* 更新を実行するときの典型的なプログラムのフロー

VSAMSQL コール・レベル・インターフェースによってサポートされる SQL ステートメント

以下の SQL ステートメントは、マップ済み VSAM データに対して VSAMSQL CLI を使用するときにサポートされます。

```
INSERT INTO table
       (col1, col2, ...)
       VALUES (val1, val2, ...),
              (val3, val4, ...),
              ...

UPDATE table
       SET col1=val2,
           col2=val2,
           ...
       WHERE col3=val3 AND
             col4=val4 AND
             ...

DELETE FROM table
       WHERE col3=val3 AND
             col4=val4 AND ...

SELECT col1, col2, ...
       FROM table
       WHERE col3=val3 AND
             col4=val4 AND
             ...
```

WHERE ステートメントは、以下の比較演算子をサポートします。

= (等しい)
< (より小)
> (より大)
<= (以下)
>= (以上)
<> (等しくない)

注:

1. **AND** を使用して複数のフィルターを結合できます。
2. **OR** はサポートされていません。

以下のものを使用して表を指定します。

- VSAM カタログ
- クラスターのファイル ID
- マップ名
- ビュー名 (オプショナル)

次に、このステートメントの構文を示します。

```
MY.USER.CATALOG/MY.VSAM.CLUSTER/MYMAP1
または
MY.USER.CATALOG/MY.VSAM.CLUSTER/MYMAP1/MYVIEW1
```

SQL ステートメントには、パラメーターの代わりに、プレースホルダー ('**?**') を入れることができます。 パラメーターは、ステートメントを実行する前にバインドする必要があります。 パラメーターには、ステートメントの始めから、1 から始まる番号が付けられます。

以下の部分にはプレースホルダーを使用できます。

- 表名
- 列名
- 値

select ステートメントの結果セットを得るには、ローカル変数を結果セットの列にバインドできます。 結果セットの列は、以下のいずれかになります。

- 次のように、select ステートメントに指定された列:

  ```
  ("SELECT col1,col2,... FROM...")
  ```

- map/view のすべての列:

  ```
  ("SELECT * FROM ...")
  ```

RRDS クラスターの場合は、**RELRECNO** という名前の特別な列が、レコード自体の一部ではないレコード (=key) の相対レコード番号を指定するために使用されます。 **SELECT * FROM** ステートメントは、自動的に、 **RELRECNO** 列を、最後の列として結果セットに追加します。 **RELRECNO** 列は、フィルターとして、サポートされるすべての SQL ステートメントで指定できます **("UPDATE table SET col1=val1 WHERE RELRECNO=5 ...")**。

Db2 ストアード・プロシージャーによる DL/I データへのアクセス

このトピックでは、Db2 ストアード・プロシージャーを使用して DL/I データにアクセスする方法について説明します。図 156 では DL/I データへのアクセスのみが示されていますが、同じ Db2 ストアード・プロシージャーを使用して、VSE/VSAM データおよび Db2 データにアクセスできます。

Db2 ストアード・プロシージャー内から DL/I データにアクセスするには、ユーザーのプログラムで、DL/I メソッドを使用して DL/I データに直接 アクセスする AIBTDLI インターフェースを使用できます。 DL/I データは、マップしません (VSE/VSAM データの場合はマップします)。

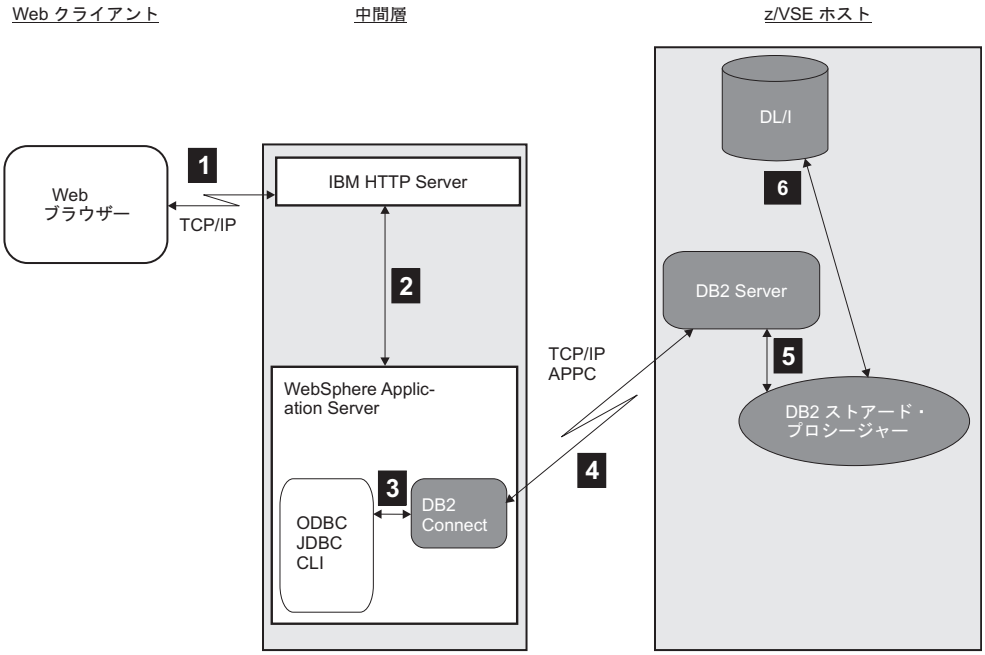


図 *156. DL/I* データにアクセスするための *Db2* ストアード・プロシージャーの使用

以下の各リスト項目の番号は、図 156 に示されているステップを説明しています。

1. クライアントの Web ブラウザーが、物理/論理中間層で実行されている IBM HTTP Server にある HTML ページを要求します。
2. IBM HTTP Server は WebSphere Application Server を呼び出して、HTML ページにある要求 (例えば、アプレットまたはサーブレットに対する要求) を求めます。
3. インターフェース (ODBC、JDBC、または CLI) が、Db2 Connect を介して Db2 Server for VSE と通信します。
4. Db2 Connect は、DRDA (分散リレーショナル・データベース体系) を介して Db2 Server for VSE と通信します。ここで使用される基礎のプロトコルは APPC または TCP/IP のどちらでもかまいません。
5. Db2 Server for VSE は、ストアード・プロシージャー・サーバーを使用して、Db2 ストアード・プロシージャーの実行を管理します。
6. これによって、Db2 ストアード・プロシージャーは、インターフェース

AIBTDLI を使用して、z/VSE ホストに保管されている DL/I データにアクセスできます (詳細については 『AIBTDLI インターフェースの概要』を参照)。

次に、ステップ 1 から 6 の逆が実行されます。Db2 Server for VSE は、結果をリクエスターに受け渡します。

459 ページの図 156 に示されているように、Db2 ストアード・プロシージャーを使用して z/VSE ホストに保管されている DL/I データにアクセスするには、以下のものが必要です。

- 物理/論理中間層上:
  - Web サーバー (例えば、IBM HTTP Server)
  - WebSphere Application Server
  - Db2 Connect
  - ODBC、JDBC、または CLI
- *z/VSE* ホスト上:
  - Db2 Server for VSE
  - DL/I VSE

以下に、Db2 ストアード・プロシージャーを使用して DL/I データにアクセスするアプリケーション・プログラムを開発するために実行する必要がある一般的なステップを示します。

1. リクエスターと z/VSE ホストとの間の Db2 接続を確立します。
2. アプリケーション・プログラムを設計します。
3. DL/I および VSAM または Db2 に対するロジックと要求を組み込んだ Db2 ストアード・プロシージャーを作成します (IBM 資料「*Db2 Server (VSE* および *VM* 版*)* データベース管理」(SC88-8636)、および「*Db2 Server (VSE* および *VM* 版*)* アプリケーション・プログラミング」(SC88-8637) を参照)。
4. Db2 に、ユーザーの Db2 ストアード・プロシージャーのエントリーを作成します。
5. Db2 ストアード・プロシージャーへのリクエスター呼び出しを作成します。
6. アプリケーション・プログラムをテストし、実行します。

AIBTDLI インターフェースの概要

AIBTDLI インターフェースを使用すると、VSE バッチ・プログラム (例えば、Db2 ストアード・プロシージャー) は、DL/I バッチ環境が DLZRRC00 または DLZMPI00 を使用して確立されていなくても、DL/I 呼び出しを出せるようになります。

注**:** AIBTDLI インターフェースを使用するためのインストール要件のリストについては、 108 ページの『ステップ 8: DL/I データ・アクセス用に Db2 ベース・コネクター をカスタマイズする』を参照してください。

AIBTDLI は DL/I 呼び出しを *DLZMPX00* に受け渡し、DLZMPX00 は、MPS バッチ・タスクのフォームで、稼働中の CICS/DLI オンライン・システムに接続します。 DLZMPX00 は、AIBTDLI を使用している各 VSE バッチ・タスクを、既知の MPS スキームで自身のために実行している、CICS/DLI オンライン・サイド上の DLZBPC00 ミラー・タスクに関連付けます。

データベースは、CICS/DLI のオンライン・サイドにあります (「オープンされています」)。 CICS/DLI オンライン・システムに受け渡されたすべての DL/I 呼び出し、および CICS/DLI オンライン・システムから戻された結果とフィードバック情報は、 VSE バッチと CICS/DLI BPC ミラー・タスクとの間の固有の XPCC 接続を使用し、DLZMPX00 を通って経路指定されます。 データベース・アクセス・コンテンション、リソース・ロギング、およびリカバリーは、既存の CICS/DLI 関数を使用して CICS/DLI オンライン・システムで実行されます。

図 157 は、3 つの既存の DL/I 環境 (バッチ、MPS バッチ、および CICS/DLI オンライン) の区画のレイアウトと処理のフローを示し、さらに、これらの環境を、AIBTDLI が使用されている環境と比較しています。

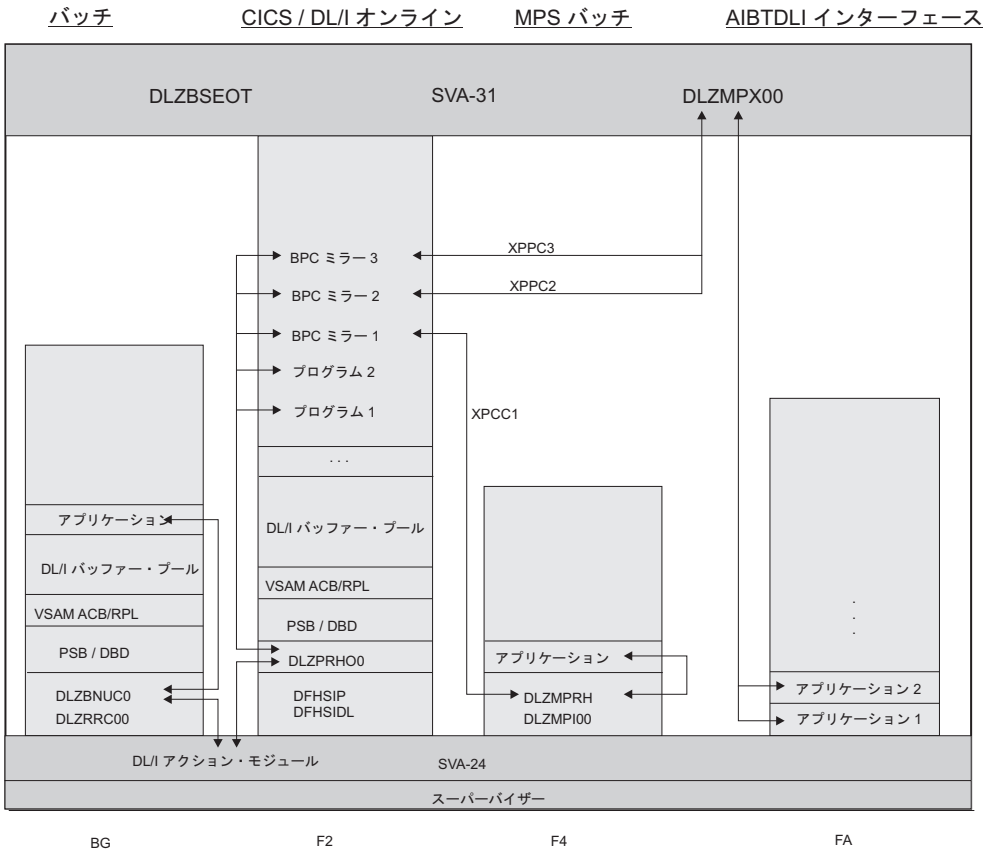


図 *157. バッチ、MPS バッチ、CICS/DLI オンライン、および AIBTDLI インターフェースの DL/I 区画のレイアウト*

BG DL/I バッチ環境は、DL/I バッチ・ルート・フェーズ *DLZRRC00* を呼び出すことによって初期化されます。 アプリケーションは DLZRRC00 によ

ってロードされ、さらに、DL/I バッチ・プログラムの要求処理プログラム *DLZBNUC0* を使用して DL/I と通信します。 DL/I リソースはすべて、DL/I バッチ区画にあります。

F2 CICS/DLI オンライン環境が、CICS ルート・フェーズ *DFHSIP* を呼び出すことによって確立され、さらに、DFHSIP が、DL/I の初期化を行うために DL/I モジュール *DFHSIDL* を呼び出します。オンライン・プログラムは、DL/I のオンライン中核 *DLZNUCxx* にある DL/I オンライン・プログラムの要求処理プログラム *DLZPRHO0* を使用して DL/I と通信します。 DL/I リソースはすべて、CICS/DLI 区画にあります。

F4 MPS バッチ環境は、DL/I MPS バッチ・ルート・フェーズ *DLZMPI00* を呼び出すことによって初期化されます。 アプリケーションは DLZMPI00 によってロードされ、さらに、MPS バッチ・プログラムの要求処理プログラム *DLZMPRH* を使用して DL/I と通信します。 DL/I リソースはすべて、CICS/DLI オンライン区画にあります。

XPCC 接続を使用して、DLZMPRH は、CICS/DLI オンライン・システムで自身のために実行を行っている CICS/DLI BPC ミラー・タスクに DL/I 呼び出しを受け渡し、そのミラー・タスクからデータを受け取ります。

FA *AIBTDLI* インターフェースが使用されている環境では、 DL/I バッチ・アプリケーションをメインプログラムとして開始することができ、そのメインプログラムは、*DLZMPX00* を使用して DL/I と通信します。 DL/I リソースはすべて、CICS/DLI オンライン区画にあります。

XPCC 接続を使用して、DLZMPX00 は、CICS/DLI オンライン・システムで自身のために実行を行っている CICS/DLI BPC ミラー・タスクに DL/I 呼び出しを受け渡し、そのミラー・タスクからデータを受け取ります。 処理のフローは、MPS バッチ環境に似ています。 違いは、DL/I バッチ・システムの初期化が必要ではない こと、および複数のバッチ・プログラム (タスク) が同時に DLZMPX00 を使用できることです。

AIBTDLI を使用するプログラムの作成

AIBTDLI インターフェースは、VSE バッチ・プログラムで使用されます。 ご使用のプログラムは、メインタスクとして開始することも、あるいは、サブタスクとして生成することもできます。 最大 128 の (サブ) タスクが同時に AIBTDLI を使用できます。

AIBTDLI を使用するプログラムは、COBOL (VSE 版)、PL/I (VSE 版)、またはアセンブラーで作成できます。 これらのプログラムは、任意の amode/rmode 特性をとることが可能です。 すべてのタイプの既存の DL/I 呼び出し関数がサポートされています。

- PCB
- GET タイプ
- ISRT
- REPL
- DLET
- CHKP
- TERM

さらに、次に説明する 2 つの新規呼び出しが提供されていることにより、データベースの変更が、DL/I CHKP 呼び出しまたは TERM 呼び出しによって、あるいは暗黙的にタスク終了によってコミットされていない場合に、バックアウトできるようになりました。

ROLB
Roll Back 呼び出しは、データベースの変更を最後の同期点まで動的にバックアウトして、制御をプログラムに戻すために使用されます。

ROLL Roll 呼び出しは、データベースの変更を最後の同期点まで動的にバックアウトして、プログラムを異常終了 (CANCEL) させるために使用されます。

ROLB 呼び出しおよび ROLL 呼び出しは、DL/I TERM 呼び出しのようにコーディングされています。 これ以上のパラメーターは不要です。

AIBTDLI インターフェースの一般呼び出しフォーマットの説明については、 464 ページの『AIBTDLI インターフェースの呼び出し』を参照してください。

DL/I の観点から見た場合、AIBTDLI を使用するアプリケーションは、CALL インターフェースを使用する CICS/DLI オンライン・プログラムと同じ方法でインプリメントされます。 DL/I データベースにアクセスするには、ご使用のアプリケーションに関して、以下のことに注意してください。

- アプリケーションは、DL/I スケジューリング呼び出しで開始する必要があります。これによって、ユーザー・プログラム (タスク) と CICS/DLI MPS システムとの間に接続が確立され、PSB がスケジュールされます。複数の PSB は順にスケジュールできます。 新規 PSB をスケジュールする前に、DL/I 終了呼び出しを使用して、前の PSB を解放する必要があります。
- アプリケーションは、任意のタイプの DL/I データベース呼び出しを実行できます。
- アプリケーションは DL/I 終了呼び出しで終了しなければなりません。これによって、ユーザー・プログラム (タスク) と CICS/DLI MPS システムとの間の接続が終了し、PSB が解放され、このタスクによって所有されていたすべての DL/I リソースが解放され、さらに、コミットおよび同期点処理が準備されます。

AIB (466 ページの『戻りコードおよび状況コード』に説明があります) に戻される DL/I 応答情報をチェックするためには、コピーブック DLIAIB を組み込む必要があります。

通常の DL/I バッチ・アプリケーションとの以下の違いに注意してください。

COBOL および **PL/I** プログラム

- ENTRY (COBOL) ステートメントまたは PROCEDURE (PL/I) ステートメントでは、PCB アドレスへの参照が不要になりました。
- PCB を指すポインターは、CICS/DLI オンライン・プログラムの場合と同じ方法で、スケジューリング呼び出しを使用して取得されます。

PL/I プログラム

- AIBTDLI インターフェースは、次のように、アセンブラー言語のエントリー・ポイントとして宣言する必要があります。

```
DCL AIBTDLI ENTRY OPTIONS(ASM);
```

- PSB は、LANG=PLI を使用して生成する必要はありません。

サンプル・プログラム DLZHLA80、DLZAIC50、および DLZAIP50 に、それぞれ、アセンブラー、COBOL、または PL/I 言語を使用したときの、AIBTDLI インターフェースを使用したプログラムの作成方法を示します。

AIBTDLI インターフェースの呼び出し

AIBTDLI インターフェースは、COBOL、PL/I、およびアセンブラー・プログラムで使用できます。このインターフェースのフォーマットとパラメーターは、それぞれ、CBLTDLI、PLITDLI、または ASMTDLI の各インターフェースと同等です。これは、「*DL/I CALL and RQDLI Interfaces*」に説明があります。 次の説明では、DL/I 呼び出しのタイプの一般フォーマットを示し、さらに、スケジューリング呼び出しの新規パラメーターについて説明します。

注**:** アセンブラー・プログラムの場合は:

- 18 フルワードのレジスター保管域をレジスター 13 に用意する必要があります。
- レジスター 1 は、パラメーター・リストを指す必要があります。

スケジューリング呼び出しのフォーマット

スケジューリング呼び出しの一般フォーマットは、次のようになります。

COBOL:

CALL 'AIBTDLI' USING [parm-count,] 'PCB ',psbname, **aibparm[,destination]**.

PL/I: CALL AIBTDLI (parm-count,'PCB ',psbname, **aibparm[,destination])**;

ASSEMBLER:

CALL AIBTDLI

次に、新規パラメーターについて説明します。 既存のパラメーターは、以前の機能性を保持しています。

aibparm

これは、DL/I が、アプリケーション・インターフェース・ブロック *(AIB)* のアドレスを戻す先のフルワードの名前です。 このパラメーターの使用は必須です。 AIB は、ユーザーに、PCB リストのアドレス、スケジューリング呼び出しに続く入出力域の最大長、それぞれの DL/I 呼び出しの後の戻りコード、およびエラーが起こった場合のメッセージ領域および区画リストを指すポインターを受け渡すために使用する新規制御ブロックです。 AIB のフォーマットと使用法についての詳細は、 466 ページの『AIB のフォーマット - ユーザー・セクション』に示されています。 AIB は、既存のオンライン UIB 制御ブロックと同等です。これは、「*DL/I CALL and RQDLI Interfaces*」に説明があります。

スケジューリング呼び出しが正常に終了すると、フィールド *AIBPCBAL* には、PCB リストのアドレスが入ります。

destination

これは、ターゲット・システムを表します。スケジューリング呼び出し (および後続のすべての呼び出し) は、ここで処理されます。 前に説明したよ

うに、AIBTDLI を使用して入力された DL/I 呼び出しは、アクティブ CICS/DLI MPS システムに経路指定されます。 同時に複数の CICS/DLI 区画で MPS がアクティブにされたときは、宛先を指定することによって、DL/I 呼び出しを送信する先の MPS サブシステムを選択できます。

PARTID=xx
... xx は、DL/I 呼び出しを処理する MPS 区画を表します。

APPLID=yyyyyyyy
... yyyyyyyy は、DL/I 呼び出しを処理する CICS (汎用) アプリケーション ID を表します。

1 つの MPS システムしか実行していない場合は、宛先の指定は必要ありません。 正しいターゲット・システムを選択する方法について詳しくは、468 ページの『単一および複数の MPS システムを使用したスケジューリング』を参照してください。

データベース呼び出しのフォーマット

データベース呼び出しの一般フォーマットは、次のようになります。

COBOL:
CALL 'AIBTDLI' USING [parm-count,]call-function, db-pcb-name,i/o-area[,ssa...].

PL/I: CALL AIBTDLI (parm-count,call-function,db-pcb-name, i/o-area[,ssa...]);

ASSEMBLER:
CALL AIBTDLI

既存のパラメーターは、以前の機能性を保持しています。

終了呼び出しのフォーマット

終了呼び出しの一般フォーマットは、次のようになります。

COBOL:
CALL 'AIBTDLI' USING [parm-count,]'TERM'.

PL/I: CALL AIBTDLI (parm-count,'TERM');

ASSEMBLER:
CALL AIBTDLI

関数コード「TERM」は「T」に短縮できます。

ロールバック呼び出しのフォーマット

ロールバック呼び出しの一般フォーマットは、次のようになります。

COBOL:
CALL 'AIBTDLI' USING [parm-count,]'ROLx'.

PL/I: CALL AIBTDLI (parm-count,'ROLx');

ASSEMBLER:
CALL AIBTDLI

ご使用のプログラムのコンパイルとリンク・エディット

COBOL、PL/I、およびアセンブラーでインプリメントされた DL/I プログラムのコンパイルおよびリンクの要件は、「*DL/I Release Guide*」に説明があります。 ただし、PL/I プログラムの場合、「*DL/I Release Guide*」にある説明と、AIBTDLI インターフェースを使用するためにユーザーが指定しなければならないものとの間に、以下のような違いがあります。

- ご使用のプログラムは、* PROCESS SYSTEM(DLI) を指定してコンパイルしてはなりません。
- 以下の 2 つのリンケージ・エディター・ステートメントを除く必要があります。

```
INCLUDE IBMRPJRA
ENTRY PLICALLB
```

サンプル・プログラム DLZHLA80、DLZAIC50、および DLZAIP50 には、それぞれ、COBOL、PL/I、およびアセンブラーの各言語で、AIBTDLI インターフェースを使用した DL/I プログラムをコンパイルし、リンク・エディットする方法が示されています。

戻りコードおよび状況コード

すべてのタイプの呼び出しについて、DL/I は、応答情報とエラー情報を、新しいアプリケーション・インターフェース・ブロック *(DLIAIB)* に入れて戻します。ユーザーは、このブロックを、前のオンライン・ユーザー・インターフェース・ブロック (DLIUIB) と同じ方法で使用できます。 DL/I データベース呼び出しの場合、状況コードは、(前と同じように) PCB を使用して戻されます。

各 DL/I 呼び出しの後で、戻された情報の検査は、AIB の検査から始めます。 AIB の戻りコードが、エラーが起こったと示していない場合は、次に、PCB 状況コードを調べます。

AIB のフォーマット - ユーザー・セクション

```
DLZAIB   DSECT
AIB      DS    0F          START OF DSECT
AIBPCBAL DS    A           PCB ADDRESS LIST
AIBRCODE DS    0XL2        DL/I RETURN CODES
AIBFCTR  DS    X           RETURN CODE
AIBDLTR  DS    X           ADDITIONAL INFORMATION
         DS    2X          RESERVED
AIBMSGPT DS    A           POINTER TO ERROR MSG AREA
AIBPLPT  DS    A           POINTER TO LIST OF PARTIDS
AIBIOLM  DS    F           MAX. LENGTH OF IOAREA
         DS    2F          RESERVED
AIBLEN   EQU   *-AIB       LENGTH OF USER AIB
```

AIB 戻りコードは、2 バイトのフィールド *AIBRCODE* に戻されます。これは、CICS/DLI オンライン・プログラムの UIBRCODE と同じ値をとります。UIBRCODE (=AIBRCODE) で発生の可能性があるすべての戻りコードのリストについては、「*DL/I Messages and Codes*」および「*DL/I Release Guide*」を参照してください。

上記の既存戻りコードに加え、新規コード *X'080A'* および *X'FF00'* が追加されています。 これらのコードの詳細については、*DL/I Release Guide* を参照してください。

また、コード *X'FF00'* については、『戻りコード X'FF00' の使用方法』(下記) を参照してください。

AIB の中のフィードバック情報を調べたら、次に、DL/I PCB 状況コードを検査してください。 すべての PCB 状況コードのリストは、「*DL/I Messages and Codes*」にあります。 新たに追加された新規の PCB 状況コードはありません。

AIB のユーザー・セクションは、メンバー DLIAIB として、アセンブラー、COBOL、および PL/I の各バージョンで送達されます。

戻りコード X'FF00' の使用方法

AIBRCODE に X'FF00' が戻されたときは、AIBTDLI インターフェースがリカバリー不能なエラーを検出しています。 この場合、AIBTDLI インターフェースは、ユーザー・タスクと CICS/DLI MPS システムとの間の接続を終了し、PSB を解放し、さらに、そのユーザー・タスクが取得していたすべての DL/I リソースを解放します。 X'FF00' 状態のエラー処理によって、DL/I 終了呼び出しが内部的にトリガーされ、この終了呼び出しによって、スケジュールされていた PSB のバックアウトと同期点処理が実行されます。

X'FF00' 状態の理由は、DL/I がコンソールに書き込むメッセージで説明されています。 アドレス・フィールド *AIBMSGPT* が、このメッセージが入るストレージ域を指します。このメッセージのフォーマットは以下のとおりです。

- 2 バイトの LL フィールド。 LL は、LLBB フィールドの長さを除いた、メッセージの長さです。
- 2 バイトの BB フィールドで、2 進ゼロにセットされます。
- メッセージのテキストが入る可変長フィールド。

戻りコードを生成しないエラー

以下の状態の場合は、DL/I は、AIB を使用して応答情報を呼び出し側に戻すことができません。

- DLZMPX00 のロードが失敗した (メッセージ DLZ150I)。
- 非互換環境 (メッセージ DLZ151I)。
- DL/I の出口ルーチン DLZBSEOT が SVA にない (メッセージ DLZ152I)。
- AIB 用の GETVIS が取得できない (メッセージ DLZ153I)。
- DL/I サブシステムの登録が失敗した (メッセージ DLZ134I)。
- AIB パラメーターが欠落しているか無効 (メッセージ DLZ154I)。

このような場合は、該当するエラー・メッセージがコンソールに書き込まれ、タスクは異常終了します (取り消されます)。

単一および複数の MPS システムを使用したスケジューリング

1 つの CICS/DLI MPS システムしか実行されていない場合、AIBTDLI インターフェースは、スケジューリング要求をこの MPS システムに送付します。 この場合、ユーザーが宛先を指定する必要はありません。

MPS が複数の CICS/DLI 区画で同時に開始された場合は、 464 ページの『スケジューリング呼び出しのフォーマット』に説明があるように、宛先の指定を使用しなければなりません。宛先の指定を使用することにより、AIBTDLI インターフェースは、それぞれの MPS システムを区別できます。

スケジューリング呼び出しが成功すると、DL/I は X'0000' を AIBRCODE に戻します。

スケジューリング・エラーが起こった場合は、AIBRCODE には、以下のいずれかが入ります。

- 「*DL/I Messages and Codes*」および「*DL/I Release Guide*」に記載された、既存の戻りコードのいずれか。
- 戻りコード X'FF00' および AIBMSGPT の中のエラー・メッセージを指すポインター。 メッセージには、スケジューリングが失敗した理由の説明があります。

2 番目の場合 (戻りコード X'FF00' および AIBMSGPT の中のエラー・メッセージを指すポインター) は、理由は、DL/I が、適切な CICS/DLI MPS ターゲット・システムを見つけられなかった可能性があります。 この場合は、以下の状態について考慮する必要があります。

- メッセージ DLZ089I ('... MPC NOT ACTIVE OR ENDING') が AIBMSGPT に戻されている場合。これは、MPS システムが開始されていないか、または、'PARTID=' または 'APPLID=' (あるいはその両方) パラメーターとして受け渡された宛先指定が、現在実行中の MPS システムのいずれとも一致しないことを意味します。
- メッセージ DLZ145I ('MORE THAN ONE MPS ACTIVE ...') が AIBMSGPT に戻されている場合。 MPS が複数の CICS/DLI 区画でアクティブになっていると、DL/I がターゲット・システムを判別できない場合があります。 これは、スケジューリング呼び出しに宛先指定が欠落しているか、あるいは、'APPLID=' は入力されているが、複数の CICS/DLI MPS システムが同じ CICS (汎用) アプリケーション ID で実行されている場合に起こります。

  DLZ145I が戻された場合は、*AIBPLPT* は、区画 ID のリストを指しています。区画 ID は、スケジューリング呼び出しで受け渡された宛先指定に関連して、アクティブな CICS/DLI MPS システムが見つかった区画を表します。
  - 宛先が指定されていなかった場合は、すべてのアクティブな MPS システムが示されます。
  - 'APPLID=' が指定されていた場合は、検索対象の CICS アプリケーション ID のもとで実行されている MPS システムのみが示されます。

  最大 10 個の区画 ID が戻されます。 これらの区画 ID は、コンマで分けられた 2 バイトの文字フィールドのリストで表されます。 コンマがないところがリストの終わりを表します。

AIBPLPT を使用して戻された区画 ID は、スケジューリング呼び出しを再試行するときの宛先指定の定義と適応に使用できます。

タスクの終了および異常終了の処理

VSE タスク終了は、最終 DL/I クリーンアップ処理を実行するために AIBTDLI インターフェースを使用したタスクのために、DL/I タスク終了出口 DLZBSEOT を呼び出します。 PSB がまだスケジュールされている場合、これは、内部 DL/I 終了呼び出しを暗黙指定します。これは、以下の理由によります。

- ご使用のプログラムを終了する前に、通常の DL/I 終了呼び出しをコーディングしていなかった。
- プログラムの異常終了が起こったときに、終了呼び出しを実行できなかった。

内部 DL/I 終了呼び出しは、タスクと CICS/DLI MPS システムとの間の接続を終了し、PSB を解放し、このタスクによって取得されていたすべての DL/I リソースを解放します。 通常の VSE タスク終了の場合は、コミットおよび同期点処理が実行されます。 VSE の異常タスク終了の場合は、バックアウトおよび同期点処理が実行されます。

AIBTDLI インターフェースがプログラム・エラーまたは異常終了条件を処理することはありません。 ご使用のプログラムが、例えば AB または PC タイプの異常終了を扱う独自の出口をセットアップして組み込む必要があります。 アセンブラー・プログラムは STXIT リンケージを使用できます。COBOL プログラムおよび PL/I プログラムは、LE TRAP ランタイム・オプションを指定して実行できます。

注**:** VSE では、出口ルーチンによって処理されたプログラム障害は、「正常」処理条件と見なします。 プログラムが「正常」処理条件で終了すると、VSE の通常タスク終了が実行されます。 処理できなかったプログラム障害のみが、結果として VSE の異常タスク終了およびバックアウト処理になります。

メッセージおよび戻りコード

AIBTDLI インターフェースは、既存の MPS バッチ関数で使用可能なメッセージのサブセットを再利用します。 それぞれの DL/I 呼び出しの後で、AIBTDLI インターフェースは、CICS/DLI オンライン・システムから戻りコードを受け取ります。これらのメッセージおよび戻りコードは、「*DL/I Messages and Codes*」および「*DL/I Release Guide*」に説明があります。

すべてのメッセージは、コンソールに書き込まれます。 また、エラー・メッセージは、フィールド AIBMSGPT を使用してユーザー・プログラムに受け渡され、戻りコードは AIBRCODE に保管されます。詳細については、 466 ページの『戻りコードおよび状況コード』を参照してください。

第 24 章 SOAP を使用したプログラム間通信

このトピックでは、Simple Object Access Protocol (SOAP と短縮します) を使用して、インターネットを介して、CICS プログラムとその他のモジュールとの間で情報を送受信する方法について説明します。

以下の項目があります。

- 『Web サービスおよび SOAP での z/VSE サポートの概要』
- 473 ページの『SOAP 構文の概要』
- 477 ページの『Web サービス (SOAP) セキュリティーの概要』
- 479 ページの『z/VSE ホストが SOAP サーバーとして機能する方法』
- 483 ページの『z/VSE ホストが SOAP クライアントとして機能する方法』
- 491 ページの『SOAP でのチャネルとコンテナーの使用』
- 485 ページの『IBM 提供の SOAP 制御ブロックの使用方法』
- 493 ページの『z/VSE SOAP エンジンの構成』
- 496 ページの『長い名前の短い名前へのマッピング (SOAP エンジン・バージョン 1)』
- 497 ページの『IBM 提供の SOAP サービス例 (getquote.c) の説明』
- 499 ページの『IBM 提供の SOAP クライアント例 (soapclnt.c) の説明』
- 501 ページの『Java SOAP クライアントの使用例』
- 502 ページの『IBM 提供の SOAP サンプルの実行』
- 506 ページの『ユーザー独自の SOAP プログラムの作成』

関連トピック:

実行する作業	参照先
CICS2WS ツールキット を使用して、CICS プログラムから Web サービス・プロキシー・コードを作成する	http://www.ibm.com/systems/z/os/zvse/products/connectors.html

Web サービスおよび SOAP での z/VSE サポートの概要

SOAP は、業界全体での標準 XML ベース・プロトコルで、アプリケーションが HTTP を介してインターネット上で情報を交換できるようにするものです。

XML は、Web 上で構造化された文書およびデータに使用される汎用フォーマットです。Web クライアントのオペレーティング・システム・プラットフォームおよび使用されるプログラム言語のいずれにも依存しません。HTTP は、すべてのインターネット Web ブラウザーおよびサーバーでサポートされます。

SOAP は、XML と HTTP の両方の利点を 1 つに合わせた標準アプリケーション・プロトコルです。 それにより、さまざまなプラットフォーム間で情報を送受信できます。

Web ブラウザーを使用すると、Web サイト上に含まれる情報を表示できます。しかし、 SOAP を使用することで、以下の作業が可能になります。

- 別の Web サイトおよびサービスのコンテンツを結合する。
- 関連するすべての情報の完全なビューを生成する。

z/VSE が SOAP プロトコルをサポートすることによって、ユーザーは Web サービスをインプリメントできます。

z/VSE SOAP サポート (z/VSE SOAP Engine) を使用して、以下を行うことができます。

- z/VSE 外部のユーザーに Web サービスを提供します。この場合、既存の、または新たに作成した CICS プログラムは Web サービス対応であるため、Web サービスとして提供できます。このシナリオでは、z/VSE は SOAP サーバーとして機能します。
- z/VSE 外部のユーザーから提供された Web サービスを使用します。この場合、CICS プログラムは z/VSE SOAP エンジンを通じて Web サービスを呼び出します。このシナリオでは、z/VSE は SOAP クライアントとして機能します。

z/VSE 5.2 以降では、サポートされている以下の 2 バージョンの z/VSE Engine があります。

- **SOAP** エンジン・バージョン **1:** VSE/ESA 2.7 以降で使用可能です。 SOAP エンコード方式のみがサポートされます。
- **SOAP** エンジン・バージョン **2:** リテラル・エンコード方式と配列サポートを備えています。

SOAP の使用例としては、旅行代理店でホテルの予約、飛行機の予約、および車のレンタルを扱う Web サービスを結合したビューを必要とする場合などがあります。旅行代理店が必要なデータを入力すると、3 つの別々のプロバイダーからの 3 つの Web サービスがすべて一度の明快なステップで処理されます。これは、「企業間取引」(B2B) 関係のインプリメント例です。

CICS2WS ツールキット

z/VSE で Web サービスを使用するには、**CICS2WS** ツールキットを使用することをお勧めします。このツールキットは、既存の CICS プログラムで Web サービスを使用するために役立つ開発ツールです。このツールは、WSDL ファイルとコピーブックを読み取り、既存のプログラムと z/VSE SOAP エンジン間のレイヤーとして使用するプロキシー・コードまたはマッピング・ルールを作成します。プロキシー・コードまたはマッピング・ルール・コードは、アセンブラー・プログラムとして生成されます。したがって、 COBOL コンパイラーや PL/I コンパイラーは不要です。このツールにより、z/VSE システムは、Web サービス・プロバイダー (サーバー) として、および Web サービス・リクエスター (クライアント) として機能できます。このツールは、Web サービス・プロバイダーと Web サービス・リクエスターの両機能用のプロキシー・コードまたはマッピング・ルール・コードを作成できます。

CICS2WS ツールは、z/VSE SOAP エンジン (バージョン 1 およびバージョン 2) をサポートしています。このツールは、z/VSE がサーバーとして機能する場合と z/VSE がクライアントとして機能する場合の両方のシナリオで使用できます。

CICS2WS は無料であり、z/VSE ホーム・ページ (http://www.ibm.com/systems/z/os/zvse/downloads/#cics2ws) からダウンロードできます。

注: CICS2WS ツールキットは現状のまま提供され、サポートや保証はありません。質問や問題がある場合については、メールを **zvse@de.ibm.com** にお送りください。

SOAP 構文の概要

通常、ここで説明されているタグ付けについて、気遣う必要はありません。タグ付けは、以下のいずれかの方法によって、自動的に 行われるからです。

- SOAP クライアントによって生成され、SOAP サーバーによってネイティブ・データに変換される。
- SOAP サーバーによって生成され、SOAP クライアント・プロセッサーによってネイティブ・データに変換される。

ただし、デバッグの目的には、SOAP タグ付けに関する知識が必要です。 この場合の詳細説明については、前のトピックで述べた URL にある Apache Web サイトを参照してください。 以下の説明は概要のみです。

SOAP メッセージは、以下の主な部分が入っている標準 XML 文書です。

- SOAP エンベロープ。メッセージの内容を定義します。
- SOAP ヘッダー (オプショナル)。ヘッダー情報が入ります。
- SOAP 本体。呼び出し情報および返信情報が入ります。

以下に、SOAP メッセージで使用されるエレメントのタイプについて説明します。

- <Envelope> エレメントは SOAP メッセージのルート・エレメントであり、XML 文書が SOAP メッセージであると定義します。
- <Header> エレメントは、SOAP メッセージについてまたはセキュリティー固有の情報についての追加のアプリケーション固有の情報を組み込むために使用できます。 この情報はユーザーが定義します。例えば、メッセージに使用する言語を定義するために使用できます。
- <Body> エレメントは、メッセージ自体を定義するために使用します。
- <Fault> エレメントはオプションで <Body> エレメント内で使用できます。またこのエレメントは、SOAP メッセージの処理中に発生する可能性があるエラーの情報を提供するために使用されます。

次に、SOAP 構文の例を示します。

```
<soap:Envelope>
  <soap:Header>
   ...
  </soap:Header>
  <soap:Body>
    <GetStock>
```

```
        <Company>IBM</Company>
      </GetStock>
    </soap:Body>
  </soap:Envelope>
```

この例は、IBM の株価を求めるために使用される SOAP XML 文書を示しています。 もちろん、これは非常に単純化されています。

リテラルとエンコード、RPC スタイルと文書スタイル

WSDL 文書 (WSDL = Web Service Description Language) は Web サービスを記述します。 WSDL バインディングは、メッセージング・プロトコル、特に SOAP メッセージング・プロトコルにサービスをバインドする方法を記述します。 WSDL SOAP バインディングは、リモート・プロシージャー・コール (RPC) スタイル・バインディングまたは文書スタイル・バインディングのいずれかです。 SOAP バインディングでは、エンコード使用またはリテラル使用も可能です。したがって、以下の 4 つのスタイル/使用モデルがあります。

- RPC/エンコード。
- RPC/リテラル。
- 文書/エンコード (実際には使用されません)。
- 文書/リテラル。

WSDL は、文書と RPC の 2 つのメッセージ・スタイルを区別します。メッセージ・スタイルは、SOAP 本体の内容に作用します。

- 文書スタイル: SOAP 本体にはパーツと呼ばれる 1 つ以上の子エレメントが含まれています。本体の内容に関する SOAP フォーマット規則はありません。本体には送信側と受信側が合意したあらゆるものが含まれます。
- **RPC** スタイル: RPC は、呼び出されるメソッドまたは操作の名前を持つエレメントが SOAP 本体に含まれていることを暗黙指定します。このエレメントには、そのメソッド/操作の各パラメーターのエレメントが含まれています。

データ・ワイヤー・フォーマットを除外するために逐次化/非逐次化を使用するアプリケーションの場合、逐次化フォーマットというもう 1 つのオプションがあります。よく使用される逐次化フォーマットには以下の 2 つがあります。

- **SOAP** エンコード方式: SOAP エンコード方式は一連の逐次化です。この規則は、どのようにオブジェクト、構造、配列、およびオブジェクト・グラフを逐次化するかを指定します。一般に、SOAP エンコード方式を使用するアプリケーションはリモート・プロシージャー・コールを重点的に扱っており、多くの場合、RPC メッセージ・スタイルを使用します。 SOAP エンコード方式を使用する場合、SOAP メッセージには SOAP メッセージ内のデータ型情報が含まれます。これにより、各パラメーターのデータ型がそのパラメーターで指示されるため、逐次化 (データ変換) が容易になります。
- リテラル: データはスキーマに従って逐次化されます。実際には、このスキーマは通常、W3C XML スキーマで表されます。 SOAP メッセージには、データ型情報が直接含まれているわけではありません。使用されているスキーマへの参照 (名前空間) だけが含まれています。適切な逐次化 (データ変換) を行うには、送信側と受信側の両方がスキーマを認識し、データの変換に同じ規則を使用する必要があります。

例: RPC/エンコード SOAP メッセージ

以下の SOAP メッセージでは、RPC スタイルと SOAP エンコード方式を使用しています。

```
<soap:envelope>
    <soap:body>
        <myMethod>
            <x xsi:type="xsd:int">5</x>
            <y xsi:type="xsd:float">5.0</y>
        </myMethod>
    </soap:body>
</soap:envelope>
```

SOAP メッセージにはオペレーション名 (myMethod) が含まれています。メソッドは 2 つのパラメーター x と y を転送します。これらはいずれも、そのデータ型 (int および float) を type 属性で指定します。

ここで使用されている名前空間は xsi と xsd の 2 つがあります。どちらも SOAP エンベロープ内で定義されます (この例には示されていません)。 xsi 名前空間はスキーマ・インスタンス (http://www.w3.org/2001/XMLSchema-instance) であり、type 属性を定義します (ほかにもあります)。 xsd はスキーマ名前空間 (http://www.w3.org/2001/XMLSchema) であり、int や float などのデータ型の意味を定義します。

例: RPC/リテラル SOAP メッセージ

以下の SOAP メッセージでは、RPC スタイルとリテラルを使用しています。

```
<soap:envelope>
    <soap:body>
        <myMethod>
            <x>5</x>
            <y>5.0</y>
        </myMethod>
    </soap:body>
</soap:envelope>
```

SOAP エンコード方式とは対照的に、パラメーター x と y は SOAP メッセージ内のデータ型を指定しません。送信側と受信側は、パラメーターのデータ型を「認識している」必要があります。この情報は通常、WSDL で使用可能です。

例: 文書/リテラル SOAP メッセージ

以下の SOAP メッセージでは、文書スタイルとリテラルを使用しています。

```
<soap:envelope>
    <soap:body>
        <x>5</x>
        <y>5.0</y>
    </soap:body>
</soap:envelope>
```

ここには、body のメソッド名またはオペレーション名の部分がありません。代わりに、body にはパラメーターが直に含まれています。また、SOAP メッセージに含まれているデータ型情報もありません。 body 内にあるものはすべて、スキーマで定義されます。

例: 文書/リテラル・ラップ SOAP メッセージ

以下の SOAP メッセージでは、文書スタイルとリテラル・ラップを使用しています。

```
<soap:envelope>
    <soap:body>
        <myMethod>
            <x>5</x>
            <y>5.0</y>
        </myMethod>
    </soap:body>
</soap:envelope>
```

この SOAP メッセージは RPC/リテラルの例とほぼ同じように見えます。ここでは、myMethod タグはメソッド名を指定していませんが、1 つの入力メッセージの部分の参照先である、ラッパー・エレメントを指定しています。

z/VSE SOAP エンジンでの使用のための推奨事項

z/VSE SOAP エンジンは SOAP エンコード方式とリテラルの両方の SOAP メッセージをサポートしています。ただし、z/VSE SOAP エンジンは文書スタイルの SOAP メッセージをサポートしていません。このメッセージにはメソッド/操作の名前が含まれていないためです。 CICS2WS ツールキットは、RPC スタイル、SOAP エンコード方式、およびリテラルの WSDL のみをサポートしています。 z/VSE は SOAP 1.1 プロトコルをサポートしています。

z/VSE SOAP エンジンは、SOAP メッセージ (z/VSE が、SOAP サーバーとして動作している場合は要求、SOAP クライアントとして動作している場合は応答) を受け取ると、以下を行います。

- パラメーターを TS キュー項目またはパラメーター・コンテナーに変換します。 **z/VSE SOAP** エンジン・バージョン **1** は可能であれば、データを適切なネイティブ・データ型に非逐次化 (変換) しようと試みます。
- パラメーターを定義された規則に従ってメモリーへ直接変換します。 **z/VSE SOAP** エンジン・バージョン **2** は、マッピング規則を使用して、データを適切なデータ型に変換します。

エンコードされた SOAP メッセージには、SOAP メッセージの一部としてデータ型情報が含まれているため、データ変換を実行可能です (型が認識され、サポートされている場合)。リテラル SOAP メッセージの場合、SOAP メッセージにはデータ型は含まれていないため、z/VSE SOAP エンジン・バージョン 1 はパラメーターをネイティブ・フォーマットに変換できません。したがって、z/VSE SOAP エンジン・バージョン 1 は、未変換データを使用して TS キュー項目またはパラメーター・コンテナーを作成します。データは、SOAP メッセージに示される場合のように、TS キュー項目またはパラメーター・コンテナーにテキスト・フォーマットで示されます。

z/VSE SOAP エンジン・バージョン 2 は SOAP エンコード方式にもリテラル SOAP メッセージにも対応し、どちらの場合もデータ変換が可能です (型が認識され、サポートされている場合)。

Web サービス (SOAP) セキュリティーの概要

Web サービス・セキュリティーは、次のように分類できます。
- トランスポート層セキュリティー
- メッセージ層セキュリティー

トランスポート層セキュリティーとメッセージ層セキュリティーの両方が以下を対象とするセキュリティー機能を提供します。
- 認証/許可
- データ暗号化およびシグニチャー

トランスポート層セキュリティーとメッセージ層セキュリティーの比較

トランスポート層セキュリティーでは、ネットワーク上を伝送されるデータを暗号化することで、通信パートナーとの間のネットワーク通信を保護します。さらに、データ保全性、認証、および機密性も実現できます。トランスポート層セキュリティーでは、一般に、デジタル署名、PKI 証明書、およびセキュア・ハッシュ機能を使用して、メッセージの「偽装」、パスワードのハッキング、およびトランザクションの拒否を防ぎます。

複数ホップで構成される環境の場合には、各ホップ間の通信を、トランスポート層セキュリティーの観点から別々に考慮する必要があります。

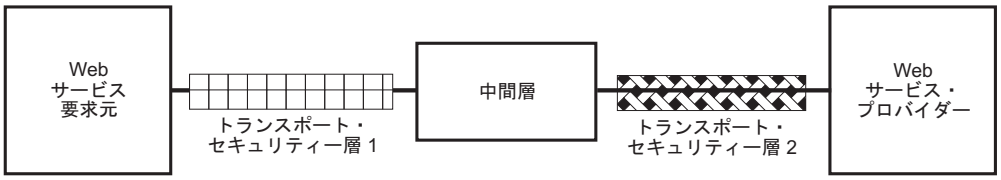


図 *158. Web* サービス・リクエスターと *Web* サービス・プロバイダーの間の接続

図 158 に示すように、各ホップ間で異なる トランスポート層セキュリティー方式を使用できます (または、一部の接続でトランスポート・セキュリティーを使用しない ようにすることも可能です)。トランスポート層セキュリティーは複数ホップに「またがり」ません。つまり、中間ホップで、メッセージを読み取ることができます。したがって、エンドツーエンド・セキュリティーを実現するには、メッセージ層セキュリティー を使用する必要があります。メッセージ層セキュリティーを使用すると、メッセージ自体が保護されるため、複数ホップにまたがって送信された場合に変更されません。

トランスポート層セキュリティーは業界に普及している以下のような任意のプロトコルを使用して実装できます。
- SSL/TLS (Secure Socket Layer/Transport Layer Security)、これは HTTP**S** として示されます。
- VPN/IPSec (これは、アプリケーションに対して透過的です)。

メッセージ層セキュリティーでは、セキュリティー関連の情報が SOAP メッセージに含められます (具体的に言うと、SOAP ヘッダー内に含められます)。

Web サービス・セキュリティーでの認証の使用

認証を使用することで、要求されたサービスの使用者をサービス・プロバイダーが確認できます。さらに、サービス・プロバイダーは、この情報を使用することで、特定のユーザー ID およびこれに関連付けられたアクセス権限 (許可) によってサービスを実行できます。

認証と許可を完全に理解するには、以下の概念を理解することが重要です。

認証　個人の資格情報を使用して、個人を識別する処理です。

許可　認証されたクライアントが、セキュリティー・ドメイン内のリソースにアクセスしたり、タスクを実行したりすることが許可されるかどうかを決定する処理です。許可では、クライアントの ID やロールまたはこの両方を使用して、クライアントがアクセスできるリソースや実行できるタスクを決定します。

資格情報
　クライアントの ID を証明するための一式のクレームです。資格情報は、クライアントの ID と、このクライアント ID を証明するもの、例えばパスワードを含みます。シグニチャーなどの情報も含めることで、発行者が資格情報内のクレームを認証していることを示すこともできます。

ID　システムで特定のサブジェクトを認識して、システムの他のユーザーと区別できるようにする用途での ID です。

認証を実行する方式は 2 つ考えられます。

- トランスポート層認証。この場合は、トランスポート層がサービスの要求者に関する情報を含んでいます。以下の方法による実装が可能です。
  - HTTP 認証 (基本およびダイジェスト・アクセス許可、RFC 2617 を参照)。
  - HTTPS での SSL/TLS クライアント認証の使用。
- メッセージ層認証。この場合は、 SOAP メッセージ自体がサービスの要求者に関する情報を含んでいます。以下の方法による実装が可能です。
  - プレーン・テキスト・パスワードまたはパスワード・ダイジェストを使用した直接認証。
  - X.509 証明書、Kerberos、セキュリティー・トークン・サービス、または SAML アサーションを使用するブローカー認証。X.509 証明書を使用するブローカー認証では、SOAP ヘッダーの一部として X.509 証明書を含んでいます。次に、例を示します。

```
<soap:Header>
  <Security xmlns="...secext-1.0.xsd>
    <BinarySecurityToken EncodigType= "wsse:Base64Binary"
                ValueType= "wsse:X509v3">
      MIICuzCCAiQCBF...
          ...
    </BinarySecurityToken>
  </Security>
  ...
```

直接認証 では、パスワードを転送する方法が 2 つ定義されています。

- プレーン・テキスト・パスワード では、UsernameToken を使用して実際のパスワードが転送されます。プレーン・テキスト・パスワード構成を使用する場合は、セキュアな転送方式 (HTTPS など) を使用する必要があります。次に、例を示します。

```
<soap:Header>
  <Security xmlns="...secext-1.0.xsd>
    <UsernameToken>
      <Username>John Smith</Username>
      <Password>Pass12wd</Password>
    </UsernameToken>
  </Security>
  ...
```

- パスワード・ダイジェスト では、通信当事者 (リクエスターおよびサービス) が、セキュアでないトランスポート・チャネルを使用します。パスワードが他者に公開されないようにする手順を実行する必要があります。ここでは、リクエスターが、一式のランダム・バイト (nonce フィールド) と、作成時刻に依存するもう 1 つの他の値 (created フィールド) を、実際のパスワードに連結して**ダイジェスト**を作成します。このダイジェストは、次のように計算されます。

  ```
  digest = Base64_encode(SHA-1(nonce+created+password))
  ```

  サービスでは、要求を認証するために、受信した username に結びつけられたパスワードを使用してダイジェスト値を計算します。サービスでは、受信した**ダイジェスト**値と計算した**ダイジェスト**値を比較します。次に、例を示します。

  ```
  <soap:Header>
    <Security xmlns="...secext-1.0.xsd>
      <UsernameToken>
        <Username>John Smith</Username>
        <Password Type="...#PasswordDigest">AFHHF23wger=</Password>
        <Nonce>ksSDGFljdfD=</Nonce>
        <Created>2010-07-15T07:12:19.573Z>/Created>
      </UsernameToken>
    </Security>
    ...
  ```

z/VSE 4.2 以降の z/VSE では、以下をサポートします。

- 以下を使用するトランスポート層認証
  - HTTP 認証 (基本およびダイジェスト・アクセス許可)
  - HTTPS による SSL/TLS クライアント認証
- 以下を使用するメッセージ層認証
  - UsernameToken (プレーン・テキスト・パスワードまたはパスワード・ダイジェスト)
  - X.509 証明書 (BinarySecurityToken)

z/VSE ホストが SOAP サーバーとして機能する方法

480 ページの図 159 は、着信 Web サービス要求が z/VSE SOAP エンジンによって処理されるときに関係するモジュールを示したものです。

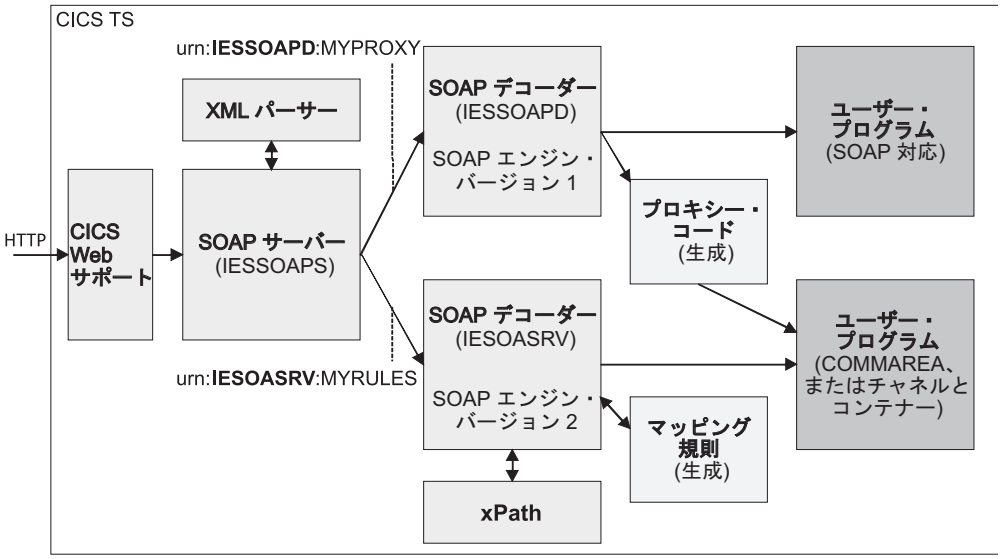


図 *159. z/VSE* が *SOAP* サーバーとして機能するときに関係するモジュール

z/VSE SOAP エンジンの SOAP サーバー部分は CICS Web サポート (CWS) に基づきます。 CWS は、着信 SOAP 要求に使用される HTTP サーバーとして機能します。 CWS は、さらに何らかの処理を行う z/VSE SOAP エンジンに要求を渡します。

CWS は TCPIPSERVICE で HTTP 要求を受け取ると、その URL を使用して、呼び出すプログラムを決定します。 Web サービス要求の場合、URL は必ず次のようになります。

```
http://hostname:port/cics/CWBA/IESSOAPS
```

この URL により CICS は CICS 提供のトランザクション CWBA の下でプログラム IESSOAPS (z/VSE SOAP エンジンのサーバー部分) を呼び出します。 IESSOAPS は HTTP コンテンツ (すなわち SOAP 要求) を取得して処理し、SOAP 応答を HTTP 応答として送り返します。

SOAP サーバーは SOAP メッセージ (XML エンコード) を受け取ると、XML パーサーを呼び出して、そのメッセージを解析します。さらに、SOAP サーバーは、その SOAP メッセージを分析して検証します。 SOAP サーバーは URN (メソッド・ネームスペース) に基づいて、使用する z/VSE SOAP エンジン・バージョンを決定します。

urn:**IESSOAPD**:MYPROXY

urn:**IESOASRV**:MYRULES

z/VSE SOAP エンジン・バージョン **1:**

urn:**IESSOAPD**:MYPROXY の場合、SOAP サーバー・プログラムはプログラム IESSOAPD (SOAP デコーダー・プログラム) を呼び出します。

SOAP デコーダーは SOAP メッセージを調査し、実行するメソッドとそのパラメーターに関する情報を抽出します。また、この SOAP デコーダーはパラメーターを XML 表現から z/VSE 固有の表現に非直列化します。各パラメーターは TS キュー項目内やパラメーター・コンテナー内に配置されます。

SOAP デコーダーは URN の 2 番目の部分 (上記の例では MYPROXY) に基づいて、次に呼び出すプログラムを決定します。これは、SOAP 対応のユーザー・プログラム (すなわち、z/VSE SOAP エンジン・プログラミング・インターフェースが直接使用される) でも、CICS2WS ツールキットによって生成されたプロキシー・プログラムでもかまいません。プロキシー・プログラムの場合は、入力パラメーターが COMMAREA に変換され、ユーザー・プログラムが呼び出されます。

SOAP 対応プログラム**:**

SOAP 対応プログラムは、COMMAREA とともに、または SOAP_PROG_PARAM 制御ブロックから提供される情報を含むコンテナーとともに、EXEC CICS LINK を使用して呼び出されます。この制御ブロックについて詳しくは、 485 ページの『IBM 提供の SOAP 制御ブロックの使用方法』セクションを参照してください。その後、SOAP 対応プログラムは TS キュー項目またはパラメーター・コンテナーから入力パラメーターを取得し、要求を処理して出力パラメーターを TS キュー項目またはパラメーター・コンテナーに戻します。

生成されたプロキシー・プログラム**:**

CICS2WS ツールキットによって生成されたプロキシー・コードが使用される場合は、そのプロキシー・コードにより、入力パラメーターが TS キュー項目またはパラメーター・コンテナーから取得され、データがユーザー・プログラム COMMAREA の入力フィールドに取り込まれます。その際、このユーザー・プログラムは EXEC CICS LINK を使用して呼び出され、COMMAREA が (COMMAREA として直接、またはコンテナー内に入れて) 渡されます。ユーザー・プログラムからの戻り時にプロキシー・コードにより、返された COMMAREA からの出力フィールドが処理されて SOAP デコーダー・プログラムに TS キュー項目またはパラメーター・コンテナーで出力パラメーターとして戻されます。

生成されたプロキシー・プログラムを使用するメリットとして、ユーザー・プログラムはユーザー・プログラム自体が SOAP によって Web サービスとして呼び出されたという事実を認識する必要がないということがあります。この概念は、EXEC CICS LINK を使用して呼び出すことができる既存の CICS プログラムを Web サービス対応にするために使用できます。そのようなプログラムは、Web サービスとして呼び出されたときに端末が割り当てられないため、端末固有の機能は使用できません。

z/VSE SOAP エンジン・バージョン **2:**

urn:**IESOASRV**:MYRULES の場合、SOAP サーバー・プログラムはプログラム IESOASRV (SOAP エンジン・バージョン 2 の SOAP デコーダー・プログラム) を呼び出します。

新しい SOAP デコーダーは SOAP メッセージを調査し、実行するメソッドとそのパラメーターに関する情報を抽出します。また、この SOAP デコーダーはパラメーターを XML 表現から z/VSE 固有の表現に非直列化します。

この場合、URN の 2 番目の部分 (上記の例では MYRULES) は IESOASRV によってマッピング規則リポジトリーとして (ユーザー・プロ

グラムによって使用される COMMAREA をマップするコピーブックに従って SOAP データ・タイプと SOAP メッセージをユーザー・プログラム・データ・タイプに変換するために) 使用されます。 MYRULES はテーブルのみで構成されるアセンブラー・プログラムであり、CICS2WS ツールキット・バージョン 2.6 以降で生成されています。

その際、ユーザー・プログラムは EXEC CICS LINK を使用して呼び出され、COMMAREA が (COMMAREA として、またはコンテナー内に入れて) 渡されます。

ユーザー・プログラムは戻るときに同じ流れをさかのぼります。このとき、出力パラメーターは IESSOAPD/IESOASRV によって直列化され、SOAP 応答メッセージが作成されます。最後に IESSOAPS が HTTP を使用して応答を送り返します。

z/VSE が SOAP サーバーとして機能するときの Web サービス・セキュリティー機能の使用

以下の領域について考慮する必要があります。

- トランスポート層の暗号化。トランスポート層認証を使用するとき、z/VSE は HTTP サーバーとして機能します。これを実装するために、 CICS Web Support (CWS) が HTTP サーバーとして使用されます。CWS では、さらに処理するために SOAP 要求を z/VSE SOAP エンジンに渡します。CWS には HTTPS (HTTP over SSL/TLS) のサポートが実装されているため SOAP エンジンでは、CWS からセキュリティー機能を継承します。HTTPS を使用するためには、以下を実行する必要があります。
  - SSL/TLS で使用するよう CICS に `TCPIPSERVICE` を構成する。
  - 必要な鍵および証明書を作成する。
- トランスポート層認証。CWS では、SSL クライアント認証 (HTTPS) および HTTP 基本認証をサポートしているため、z/VSE SOAP エンジンでは、CWS からセキュリティー機能を継承します。クライアントに HTTP 基本認証の使用を強制するには、TCPIPSERVICE を構成して CICS 提供のコンバーター・プログラム DFH$WBSB を使用するようにする必要があります (`URM=DFH$WBSB` を指定します)。さらに、z/VSE SOAP エンジンは、認証情報 (HTTP 基本認証用のユーザー ID とパスワードまたは SSL クライアント認証用のマップされたユーザー ID) を抽出します。この情報をコンバーター・コードで使用してトランスポート層認証が使用されたかどうかを検査できます。認証が使用されなかった場合は、コンバーター・コードで要求を拒否できます。
- メッセージ層認証。メッセージ層認証をサポートするために、z/VSE SOAP エンジン (つまり、z/VSE SOAP サーバー) では、 XML データ・ストリームを構文解析した後で SOAP ヘッダーから認証トークンを抽出します。`UsernameToken` の場合は、ユーザー ID およびパスワードを、ローカル ID ストアと照合して検査する必要があります。これを実行するには、ID ストアは、パスワード・ダイジェストのプレーン・テキスト・パスワードを、保管されたパスワードと比較できる必要があります。追加でユーザー許可を実行する場合は、受け取ったユーザー名を z/VSE ユーザー ID の 1 つにマップするユーザー・マッピングを実行する必要があります。また、このマップされたユーザーを使用して、CICS SIGNON を実行することで、トランザクションがこのユーザーの下で実行されるようにする必要もあります。`UsernameToken` に加え、認証に証明書を使用することも可能です。この場合は、コンバーター・コードが受け取った証明

書を z/VSE ユーザーにマップします。z/VSE では、SSL/TLS クライアント認証の一部として、この機能をサポートします。

z/VSE SOAP エンジン自体が認証を実行するのではないことに注意してください。SOAP エンジンでは、セキュリティー情報の抽出を行い、これをコンバーター・コードまたはユーザー・アプリケーションに渡すのみです。許可検査またはサインオン処理の実行はユーザー・アプリケーションが担当します。

z/VSE ホストが SOAP クライアントとして機能する方法

図 160 は、発信 Web サービス要求が z/VSE SOAP エンジンによって処理されるときに関係するモジュールを示したものです。

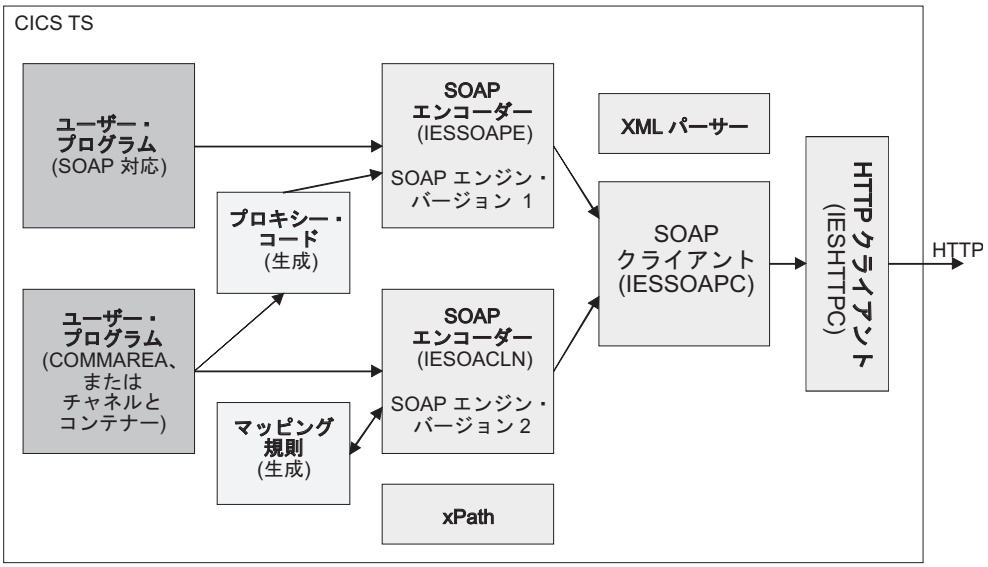


図 *160. z/VSE* が *SOAP* クライアントとして機能するときに関係するモジュール

ユーザー・プログラムは外部 Web サービスを呼び出す必要があるときは、z/VSE SOAP エンジンのクライアント部分を呼び出します。ユーザー・プログラムができることは以下のとおりです。

- SOAP 対応のプログラムになることができます (すなわち、z/VSE SOAP エンジン・プログラミング・インターフェースが直接使用されます)。
- z/VSE SOAP エンジン・バージョン 1 と一緒に使用するために CICS2WS ツールキットによって生成されたプロキシー・プログラムを呼び出すことができます。
- CICS2WS ツールキットによって生成されたマッピング規則を指定して SOAP エンジン・バージョン 2 を呼び出すことができます (z/VSE 5.2 以降)。

SOAP 対応プログラム **(z/VSE SOAP** エンジン・バージョン **1):**
SOAP 対応プログラムは、COMMAREA を渡したり、SOAP_DEC_PARAM 制御ブロックから提供される情報を含むコンテナーを渡したりして、SOAP エンコーダー・プログラム IESSOAPE を直接呼び出します。この制御ブロックについて詳しくは、 485 ページの『IBM 提供の

SOAP 制御ブロックの使用方法』セクションを参照してください。 SOAP 対応プログラムは TS キュー項目またはパラメーター・コンテナーに入力パラメーターを配置し、SOAP エンコーダー・プログラムを呼び出します。呼び出された SOAP エンコーダー・プログラムが Web サービス呼び出しを行います。 SOAP 対応プログラムは戻り時に、返された出力パラメーターを TS キュー項目またはパラメーター・コンテナーから取得します。

生成されたプロキシー・プログラム **(z/VSE SOAP** エンジン・バージョン **1):**
プロキシー・プログラムを使用する場合、ユーザー・プログラムは COMMAREA を渡したり COMMAREA を含むコンテナーを渡したりして EXEC CICS LINK で、生成されたプロキシー・プログラムを呼び出します。その後で、プロキシー・プログラムが COMMAREA の入力フィールドのデータを TS キュー項目またはパラメーター・コンテナーに配置し、SOAP エンコーダー・プログラムを呼び出します。呼び出された SOAP エンコーダー・プログラムが Web サービス呼び出しを行います。 SOAP エンコーダー・プログラムからの戻り時にプロキシー・コードにより、TS キュー項目またはパラメーター・コンテナーから返されたすべての出力パラメーターが COMMAREA の出力フィールドに配置され、制御がユーザー・プログラムに戻ります。

z/VSE SOAP エンジン・バージョン **2** の使用**:**
z/VSE SOAP エンジン・バージョン 2 を使用する場合、ユーザー・プログラムは COMMAREA を渡したり COMMAREA を含むコンテナーを渡したりして EXEC CICS LINK でプログラム IESOACLN を呼び出します。COMMAREA は 8 文字のフィールドで始まり、CICS2WS ツールキット・バージョン 2.6 以降で生成されたマッピング規則プログラムの名前 (例: MYRULES) を含んでいなければなりません。この規則プログラムは IESOACLN によってマッピング規則リポジトリーとして (ユーザー・プログラムによって使用される COMMAREA をマップするコピーブックに従ってユーザー・プログラム・データ・タイプを SOAP データ・タイプと SOAP メッセージに変換するために) 使用されます。

Web サービス呼び出しには、SOAP エンコーダー・プログラム (IESSOAPE/IESOACLN) で行われる SOAP 処理が関係します。 SOAP エンコーダーは Web サービスの入力パラメーターを z/VSE 固有の表現から XML 表現に直列化して SOAP クライアント・プログラム (IESSOAPC) を呼び出します。

SOAP クライアント・プログラムは、SOAP 本体やエンベロープだけでなく、Web サービスの入力パラメーターや、呼び出すメソッドも含まれている SOAP メッセージを生成します。その後で、SOAP クライアント・プログラムは Web サービスの URL を使用して HTTP クライアント・プログラムを呼び出します。

HTTP クライアントは、URL に示されたサーバーへの TCP 接続をオープンし、SOAP メッセージを含む XML データとともに HTTP 要求を送信します。 HTTP クライアントは HTTP 応答を受信して、その応答を SOAP クライアント・プログラムに戻します。

SOAP クライアント・プログラムは XML データを解析し、SOAP 応答メッセージを分析し、Web サービスの出力パラメーターを SOAP エンコーダー・プログラムに渡します。

z/VSE が SOAP クライアントとして機能するときの Web サービス・セキュリティー機能の使用

以下の領域について考慮する必要があります。

- トランスポート層の暗号化。トランスポート層認証を使用するとき、z/VSE は HTTP クライアントとして機能します。次に、z/VSE に実装されている HTTP クライアントが HTTPS をサポートします。HTTPS を使用するには、以下を実行します。
  - URL で `https://` を指定する必要があります。
  - 公開鍵と秘密鍵のペアおよび証明書を指定する必要があります。鍵を指定する方法の詳細については、VSE/ICCF ライブラリー 59 に含まれているスケルトン SKSOAPOP を参照してください。
- トランスポート層認証。z/VSE 4.2 以降の HTTP クライアントでは、SSL/HTTPS をサポートします。したがって、証明書を使用する SSL クライアント認証を使用できます。SSL プロトコルでは、要求された場合に、クライアントの証明書をサーバー (サービス・プロバイダー) に送信できます。必要な場合、サーバーではクライアントの証明書を使用して、認証および許可を実行できます。さらに、HTTP 基本認証が z/VSE HTTP クライアントによってサポートされます。
- メッセージ層認証。z/VSE 4.2 以降では、z/VSE SOAP エンジン (つまり、z/VSE SOAP クライアント) で、SOAP ヘッダー内の認証トークンをサポートします。サービスを要求しているユーザー・アプリケーション (またはコンバーター・コード) では、`UsernameToken` の `username` および `password` を SOAP エンジンに渡す必要があります。証明書を使用して認証を実行するときは、証明書名を指定する必要があります。この情報を渡すためのコードは、ユーザー・アプリケーションに含めるか、コンバーター・コードに含めることができます。

IBM 提供の SOAP 制御ブロックの使用方法

このトピックでは、C 言語ヘッダー・ファイル IESSOAPH.H に定義されている IBM 提供の SOAP 制御ブロックについて説明します。その制御ブロックは SOAP エンジン・バージョン 1 にのみ適用されます。 **IESSOAPH.H** は **PRD1.BASE** にあります。また、ヘッダー・ファイルは VSE コネクター・クライアントの一部としてディレクトリー **...¥<install-directory>¥samples¥soap¥vseSoapClient** にも提供されています。

ファイル IESSOAPH.H は、z/VSE ホストで実行される、以下のすべての SOAP プログラムで使用されます。

- 480 ページの図 159 の SOAP デコーダー、SOAP サーバー、およびユーザー・プログラム。
- 483 ページの図 160 の SOAP エンコーダー、SOAP クライアント、およびユーザー・プログラム。

独自の SOAP エンコーダー/デコーダーを作成する場合 (IBM 提供のエンコーダー/デコーダーが自分の要件を満たさない場合) は、他の制御ブロック (ここでは説明されていません) が使用されます。独自の SOAP エンコーダーまたはデコーダーを作成する場合は、制御ブロックの詳細について、ヘッダー・ファイル文書を参照してください。

SOAP_PARAM_HDR 制御ブロックの使用方法

SOAP_PARAM_HDR 制御ブロックは、以下の項目から成る、各パラメーターのデータを指定するために使用します。

- 名前
- 値 (データ自体)
- 値の長さ
- 値のタイプ

以下の例に示します。

```
char         name[16];      // parameter name
char         typename[16];  // data type name
unsigned int length;        // length of block (inc. header)
unsigned int type;          // type (see SOAP_TYPE_xxx)
```

図 *161. SOAP* パラメーターの内容

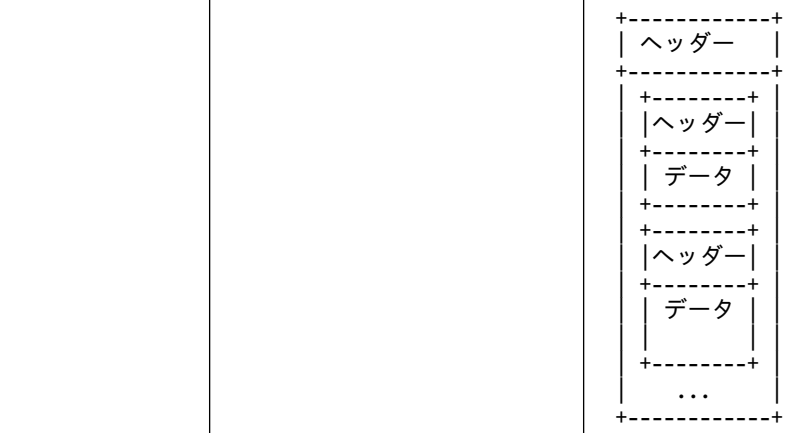
1. パラメーターのデータは、
   - CICS ユーザー・トランザクションに受け渡されます。または、
   - CICS ユーザー・トランザクションによって生成されます。
2. 次に、データは SOAP コンバーターによって、
   - ネイティブ・データ (図 161 に示したデータ) から XML に変換されます。または、
   - XML からネイティブ・データに変換されます。

以下に、図 161 の値フィールドの**タイプ**として指定可能なすべての値のリストを示します。

```
// Values for type field in SOAP_PARAM_HDR
#define SOAP_TYPE_UNSPECIFIED   0     // unknown/unspecified type
#define SOAP_TYPE_PRIVATE       1     // private type
#define SOAP_TYPE_STRUCT        2     // hirachical structure
#define SOAP_TYPE_ARRAY         3     // hirachical array structure
#define SOAP_TYPE_STRING        10    // String
#define SOAP_TYPE_INTEGER       11    // Integer (4 bytes)
#define SOAP_TYPE_SHORT         12    // Short (2 bytes)
#define SOAP_TYPE_BYTE          13    // Byte (1 byte)
#define SOAP_TYPE_BOOLEAN       14    // Boolean (1 byte)
#define SOAP_TYPE_BINARY        15    // Binary
#define SOAP_TYPE_LONG          16    // Long (8 bytes)
#define SOAP_TYPE_UINTEGER      17    // Unsigned Integer (4 bytes)
#define SOAP_TYPE_USHORT        18    // Unsigned Short (2 bytes)
#define SOAP_TYPE_UBYTE         19    // Unsigned Byte (1 byte)
#define SOAP_TYPE_ULONG         20    // Unsigned Long (8 bytes)
// Decimal data types (represented as Packed Decimal COMP-3)
#define SOAP_TYPE_DECIMAL       21    // Decimal (with decimal places)
#define SOAP_TYPE_DECINT        22    // Integer
// Date and Time specific types
#define SOAP_TYPE_DATETIME     100    // Date and Time in ISO 8601
#define SOAP_TYPE_DATE         101    // Date  (e.g. "2002-10-10")
#define SOAP_TYPE_TIME         102    // Time  (e.g. "13:20:00")
#define SOAP_TYPE_GYEAR        103    // Year  (e.g. "2005")
#define SOAP_TYPE_GMONTH       104    // Month (e.g. "--05")
#define SOAP_TYPE_GDAY         105    // Day   (e.g. "---01")
#define SOAP_TYPE_GYEARMONTH   106    // Year and Month (e.g. "1999-05")
#define SOAP_TYPE_GMONTHDAY    107    // Month and Day  (e.g. "--05-01")
```

図 *162.* 値フィールドの**タイプ**の可能な値

486 ページの図 162 の項目は、以下に説明する 3 つのフィールドを除いて自明です。

タイプ	説明
SOAP_TYPE_UNSPECIFIED	**タイプ**が SOAP_TYPE_UNSPECIFIED の場合、SOAP コンバーターは、任意のネームスペース URL なしの 不明なタイプを検出します。 この場合、データは変換されず、プレーン・テキストになります。 したがって、ご使用のプログラムでタイプを認識できるか (ネームスペースがない場合でも) を検査し、次に、自身で、データを変換する必要があります。
SOAP_TYPE_PRIVATE	**タイプ**が SOAP_TYPE_PRIVATE の場合、SOAP コンバーターはネームスペース URL を持つ不明のタイプを検出します。 この場合、データは変換されず、プレーン・テキストになります。 この場合、ご使用のプログラムで、ネームスペース URL/タイプ・ペアが既知のものであるかを検査し、自身でデータを変換する必要があります。ネームスペース URL は、SOAP_PROG_PARAM 制御ブロックにあります。
SOAP_TYPE_STRUCT	**タイプ**が SOAP_TYPE_STRUCT の場合、SOAP コンバーターは階層構造を受け取ります。 これは、配列の 1 つのタイプ、あるいは、エンクロージング配列 (階層の深さが無限) のメンバーである配列の可能性があります。 階層構造の例を、次に示します。  「外側」のヘッダーには、このヘッダーのデータの長さが入ります。 タイプは SOAP_TYPE_STRUCT です。 この例では、「外側」のヘッダーのデータには、2 つのパラメーター (それぞれは、ヘッダーとデータで構成される) が入ります。 各パラメーターは任意の長さでかまいません。 プログラムは、外側のヘッダーのデータ・ブロックの長さを現在のパラメーター長に照らして検査することによって、まだパラメーターがあるかを認識できます。

タイプ	説明
SOAP_TYPE_ARRAY	配列は、SOAP_TYPE_STRUCT に似た処理が行われます。 配列内の項目数が、SOAP_PARAM_HDR のフィールド type の上位ハーフワードで指定されます。 フィールド typename で、内部パラメーター (配列項目) の typename を指定します。
SOAP_TYPE_DECIMAL	10 進数は、パック 10 進数 (COBOL の COMP-3) で示されます。 暗黙の小数点位 (小数部の桁数) は、SOAP_PARAM_HDR のフィールド type の上位ハーフワードで指定されます。

SOAP_PROG_PARAM 制御ブロックの使用方法

トランザクションが *SOAP* サービス として呼び出されると、SOAP_PROG_PARAM 制御ブロックが CICS ユーザー・トランザクションに渡されます (例えば、 480 ページの図 159 では、ホスト上で実行されるユーザー・プログラムは SOAP デコーダーによって呼び出されると SOAP_PROG_PARAM を受け取ります)。

CICS ユーザー・トランザクションは、コンバーター・プログラム (例えば、IBM 提供の IESSOAPE) によって呼び出されます。 この制御ブロックには、要求された SOAP メソッド名、使用する入力待ち行列および出力待ち行列の名前、さらに、場合により、含まれているパラメーター・タイプのネームスペース URL が含まれます (上記の SOAP_PARAM_HDR を参照)。戻すパラメーターとしてプライベート・タイプ (SOAP_TYPE_PRIVATE) を使用する場合は、独自のネームスペース URL をここで指定できます。 コンバーターは、ご使用のネームスペース/タイプ・ペアを使用して、XML を構築します。

次に、SOAP_PROG_PARAM 制御ブロックに入っているフィールドのリストを示します。

```
char  method[16];        // (in)     method name
char  inqueue[8];        // (in)     input params
char  outqueue[8];       // (in)     output params
char  namespaceurl[128]; // (in/out) private namespace url
int   retcode;           // (out)    return code
char  methodlong[128];   // (in)     long method name
// Authentication
int   authtype;          // (in) Authetication type (AUTH_xxx)
char  authuser[64];      // (in) user id for authentication
char  authpwd[64];       // (in) password for authentication

// Authentication types
#define AUTH_NONE               0  // No authentication
#define AUTH_HTTP_BASIC         1  // basic HTTP authentication
#define AUTH_WS_PLAIN           2  // authentication using plain pwd
#define AUTH_WS_DIGEST          3  // authentication using digest
#define AUTH_WS_CERT            4  // authentication using certificate
#define AUTH_SSL_CERT           5  // SSL client authentication
```

図 *163. SOAP_PROG_PARAM* 制御ブロックに入っているフィールド

注:

1. methodlong フィールドは、16 文字より大きいメソッド名をサポートします。

2. 以前のリリースとの互換性のため、method フィールドには、メソッド名の最初の 16 文字が含まれます。
3. AUTH_HTTP_BASIC および AUTH_WS_PLAIN の場合は、指定された方式を使用してクライアント・サイドから受け取ったユーザー ID とパスワードがフィールド authuser および authpwd に指定されます。 AUTH_WS_DIGEST の場合は、フィールド authuser にユーザー ID が含まれ、authpwd には、以下の 3 つの項目を保持する TS-QUEUE の名前が指定されます。
   a. base64 形式のパスワード・ダイジェスト
   b. テキスト形式で作成されたタイム・スタンプ
   c. base64 形式の nonce 値

   チャネルおよびコンテナーが使用されるときは、TS キューの代わりに 3 つの追加コンテナーが渡されます。
   a. コンテナー「AUTHINFO-DIGEST」には base64 形式のパスワード・ダイジェストが含まれます。
   b. コンテナー「AUTHINFO-CREATED」には、作成されたタイム・スタンプがテキストとして含まれます。
   c. コンテナー「AUTHINFO-NONCE」には base64 形式の nonce 値が含まれます。
4. AUTH_WS_CERT および AUTH_SSL_CERT の場合は、証明書にマップされたユーザー ID が authuser フィールドに含まれ、ユーザーがマップされなかった場合はブランクになります。authpwd フィールドは、以下の項目を保持する TS-QUEUE の名前を含みます。
   a. バイナリー証明書データ

   チャネルおよびコンテナーが使用されるときは、TS キューの代わりに 1 つの追加コンテナーが渡されます。
   a. コンテナー「AUTHINFO-CERT」にはバイナリー証明書データが含まれます。

SOAP_DEC_PARAM 制御ブロックの使用方法

SOAP_DEC_PARAM 制御ブロックは、SOAP コンバーター IESSOAPE を呼び出すために、SOAP クライアントとして機能する CICS ユーザー・トランザクションによって使用されなければなりません。次に、SOAP コンバーターが SOAP クライアント・プロセッサーを呼び出し、SOAP クライアント・プロセッサーが、SOAP サーバーに対して SOAP 呼び出しを実行します。 詳細については、 483 ページの図 160を参照してください。

フィールド proxytype が HTTP_TYPE_DIRECT (0) でない場合は、ユーザーが、このタイプに必要なプロキシー・フィールドを充てんする必要があります。

```
char  url[128];         // (in) the servers url
char  method[16];       // (in) method name
char  urn[128];         // (in) the urn
char  inqueue[8];       // (in) input queue name
char  outqueue[8];      // (in) output queue name
char  namespaceurl[128]; // (in/out) namespace url
// proxy
int   proxytype;        // (in) proxy type (HTTP_TYPE_xxx)
char  proxy[128];       // (in) proxy server
int   proxyport;        // (in) port number
char  userid[16];       // (in) userid for socks
char  password[16];     // (in) password for socks 5
// return code
int   retcode;          // (out) return code
// SOAP Action
char  soapaction[128]; // (in) SOAPAction value
char  methodlong[128]; // (in) long method name
// Authentication
int   authtype;       // (in) Authetication type (AUTH_xxx)
char  authuser[64];    // (in) user id for authentication
char  authpwd[64];     // (in) password for authentication

// Authentication types
#define AUTH_NONE               0  // No authentication
#define AUTH_HTTP_BASIC         1  // basic HTTP authentication
#define AUTH_WS_PLAIN           2  // authentication using plain pwd
#define AUTH_WS_DIGEST          3  // authentication using digest
#define AUTH_WS_CERT            4  // authentication using certificate
#define AUTH_SSL_CERT           5  // SSL client authentication
```

図 *164. SOAP_DEC_PARAM* 制御ブロックに入っているフィールド

注:

1. methodlong フィールドは、16 文字超のメソッド名をサポートするために使用します。
2. methodlong フィールドに含まれている値がすべてゼロまたはブランクの場合、あるいは渡される COMMAREA またはコンテナーの長さが、新規フィールドが存在しないことを示している場合、メソッド名は前に使用されたフィールド・メソッドから取得されます。これによって、既存プログラムとの後方互換性を提供します。
3. COMMAREA またはコンテナーが十分大きく methodlong がゼロやブランクでない場合、メソッド名は methodlong から取られます。
4. AUTH_HTTP_BASIC、AUTH_WS_PLAIN および AUTH_WS_DIGEST の場合は、選択された方式を使用してサーバーに送信されるユーザー ID とパスワードがフィールド authuser および authpwd に指定されます。AUTH_WS_CERT の場合は、使用する証明書メンバーの名前、例えば CRYPTO.KEYRING(CERTNAME) がフィールド authuser に指定されます。この場合は、フィールド authpwd は使用されず、無視されます。AUTH_SSL_CERT は SOAP クライアント操作ではサポートされません。ただし、サーバー・サイドで、SSL に使用されるクライアント証明書を使用することに決めてクライアント認証を実行することはできます。
5. 渡された COMMAREA またはコンテナーの長さが、フィールド authtype、authuser、および authpwd が存在しないことを示している場合、認証は使用されません (後方互換性)。

以下は、SOAP サーバーへの直接接続が使用可能でない場合にユーザーが使用できる定義済みのプロキシー・タイプです。

```
// Proxy types
#define HTTP_TYPE_DIRECT      0 // direct connection
#define HTTP_TYPE_PROXY       1 // connection through a proxy
#define HTTP_TYPE_SOCKS4      2 // connection through Socks V4
#define HTTP_TYPE_SOCKS5      3 // connection through Socks V5
```

図 *165. SOAP_DEC_PARAM* 制御ブロックで使用できるプロキシー・タイプ

SOAP でのチャネルとコンテナーの使用

このトピックでは、z/VSE SOAP エンジンでのチャネルとコンテナーの使用方法について説明します。チャネルとコンテナーは、z/VSE V6.1 以降、および z/VSE V2.1 以降向けの CICS Transaction Server でサポートされています。

z/VSE が SOAP サーバーとして動作する場合

SOAP エンジン・バージョン **1:**

SOAP サーバーとしての z/VSE の場合、チャネルとコンテナーの使用は SOAP 要求の受信時に SOAP エンジンによって決定されます。チャネルとコンテナーを使用する場合、URN には末尾に「?CC=yes」標識が含まれている必要があります (例:「urn:IESSOAPD:userprog**?CC=yes**」)。 SOAP エンジンは、使用するチャネルとコンテナーを認識すると、EXEC CICS LINK を使用してユーザー・プログラムを呼び出し、各種コンテナーを含む「IESSOAPD-CHANNEL」というチャネルを渡します。
SOAP_PROG_PARAM 制御ブロックを使用して COMMAREA を渡す代わりに、「CALLINFO」というコンテナーがユーザー・プログラムに渡されます。 CALLINFO コンテナーには SOAP_PROG_PARAM 制御ブロックが含まれています。ユーザー・プログラムは、SOAP_PROG_PARAM 制御ブロックの更新を行う際、リターン時にコンテナー CALLINFO をチャネルに戻す必要があります。

注**:** SOAP_PROG_PARAM 制御ブロックの inqueue フィールドと outqueue フィールドは、チャネルとコンテナーの使用時には使用されません。代わりに、input パラメーターと output パラメーターもコンテナーとして渡されます (以下のセクションを参照)。

CICS2WS ツールキットを使用してプロキシー・コードを生成し、チャネルおよびコンテナー・サポートの使用を選択した場合、末尾の「?CC=yes」によって URN が自動的に生成されます。プロキシー・コード自体も、チャネルとコンテナーを利用してユーザー・プログラムを呼び出します。プロキシー・コードは EXEC CICS LINK を使用してユーザー・プログラムを呼び出し、「COMMAREA」というコンテナーを含む「SOAPCALL」というチャネルを渡します。このコンテナーには、チャネルとコンテナーが使用されない場合に COMMAREA に含まれるものと同じ情報が入っています。チャネルとコンテナーの使用時には、COMMAREA コンテナーのサイズは 32KB に制限されません。

SOAP エンジン・バージョン **2:**

SOAP サーバーとしての z/VSE の場合、チャネルとコンテナーの使用は規則内部のフラグによって制御されます。 CICS2WS ツールキットを使用して規則を作成する際に、ユーザーはチャネルとコンテナーを使用するかどうかを指定します。チャネルとコンテナーを使用する場合、z/VSE SOAP エンジンは、「RULESINFO」というコンテナーを含む「IESOASRV-CHANNEL」というチャネルを使用して、EXEC CICS LINK によってユーザー・プログラムを呼び出します。このコンテナーには、チャネルとコンテナーが使用されない場合に COMMAREA に含まれるものと同じ情報が入っています。

z/VSE が SOAP クライアントとして動作する場合

SOAP エンジン・バージョン **1:**

SOAP クライアントとしての z/VSE の場合、チャネルとコンテナーの使用は実行時に SOAP エンジンによって決定されます。 SOAP エンジンは、SOAP 対応のユーザー・プログラムが COMMAREA を使用して、またはチャネルとコンテナーを使用して SOAP エンジン(IESSOAPE など) を呼び出したかどうかを検出します。チャネルとコンテナーが使用された場合、SOAP エンジンは、渡されたチャネルに、SOAP_DEC_PARAM 制御ブロックの入った「CALLINFO」というコンテナーが含まれていると想定します。

注: SOAP_DEC_PARAM 制御ブロックの inqueue フィールドと outqueue フィールドは、チャネルとコンテナーの使用時には使用されません。代わりに、input パラメーターと output パラメーターもコンテナーとして渡されます (以下のセクションを参照)。

CICS2WS ツールキットを使用してプロキシー・コードを生成し、チャネルおよびコンテナー・サポートの使用を選択した場合、ユーザー・プログラムは、EXEC CICS LINK によってプロキシー・プログラムを呼び出し、「COMMAREA」というコンテナーを含むチャネルを渡す必要があります。ユーザー・プログラムへのリターン時には、プロキシー・コードはこのコンテナーを、Web サービスから返された出力データで更新しています。

SOAP エンジン・バージョン **2:**

SOAP クライアントとしての z/VSE の場合、チャネルとコンテナーの使用は実行時に SOAP エンジンによって決定されます。 SOAP エンジンは、ユーザー・プログラムが COMMAREA を使用して、またはチャネルとコンテナーを使用して SOAP エンジン (IESOACLN など) を呼び出したかどうかを検出します。チャネルとコンテナーが使用された場合、SOAP エンジンは、渡されたチャネルに、チャネルとコンテナーが使用されない場合に COMMAREA に含まれるものと同じ情報の入った「RULESINFO」というコンテナーが含まれていると想定します。

パラメーター・データの受け渡し方法 (SOAP エンジン・バージョン 1 のみ)

チャネルとコンテナーを使用する場合、パラメーター・データは、TS キュー項目ではなくコンテナーで渡されます。 TS キュー項目には 32KB のサイズ制限があるの

に対し、コンテナーにはサイズ制限はありません。そのため、SOAP Web サービスでは大きなパラメーター・データを渡すことが可能です。 TS キュー項目で渡されるパラメーター・データと同様に、パラメーター・コンテナーにはパラメーターを記述する SOAP_PARAM_HDR 制御ブロックが含まれます。

入力パラメーターは「INPUT---nnnnnnnn」というコンテナーで渡されます。nnnnnnnn は左側にゼロが埋められた、1 から始まる 10 進数です。したがって、最初の入力パラメーター・データは「INPUT---00000001」というコンテナーに入れられ、2 番目の入力パラメーター・データは「INPUT---00000002」というコンテナーに入れられ、以降も同様に続きます。 SOAP エンジンはすべてのパラメーターを、入力パラメーターがなくなるまで順番に処理します。

出力パラメーターは「OUTPUT--nnnnnnnn」というコンテナーで渡されます。nnnnnnnn は左側にゼロが埋められた、1 から始まる 10 進数です。したがって、最初の出力パラメーター・データは「OUTPUT--00000001」というコンテナーに入れられ、2 番目の出力パラメーター・データは「OUTPUT--00000002」というコンテナーに入れられ、以降も同様に続きます。 SOAP エンジンはすべてのパラメーターを、出力パラメーターがなくなるまで順番に処理します。

z/VSE SOAP エンジンの構成

z/VSE SOAP エンジンは、IBM によりプリインストールおよび事前構成されて出荷されています。通常は、構成を変更する必要はありません。

ただし、固有の要件がある場合 (例えば、異なるコード・ページを使用する場合) は、VSE/ICCF ライブラリー 59 に含まれるスケルトン SKSOAPOP を使用して SOAP オプション・フェーズ **IESSOAPO** を生成する必要があります。

SOAP 構成内の以下の設定を調整できます。

- ASCII/EBCDIC 変換に (グローバルに、または TCPIPSERVICE ごとに) 使用するコード・ページ。
- SSL/TLS の設定 (z/VSE が SOAP Client として稼働する場合の HTTPS 用)。
- SOAP パラメーター名のための長い名前から短い名前へのマッピング (z/VSE SOAP エンジン・バージョン 1 にのみ適用)。

次に、SOAP エンジンの構成の例を示します。

```
* $$ JOB JNM=SOAPOPT,CLASS=A,DISP=D
// JOB SOAPOPT GENERATE SOAP OPTION PHASE
* ********************************************************************
* ASSEMBLE AND LINK THE SOAP OPTION PHASE                            *
* ********************************************************************
// LIBDEF *,CATALOG=PRD2.CONFIG
// LIBDEF *,SEARCH=PRD1.BASE
// OPTION ERRS,SXREF,SYM,NODECK,CATAL,LISTX
   PHASE IESSOAPO,*,SVA
// EXEC ASMA90,SIZE=(ASMA90,64K),PARM='EXIT(LIBEXIT(EDECKXIT)),SIZE(MAXC
               -200K,ABOVE)'
IESSOAPO CSECT
IESSOAPO AMODE ANY
IESSOAPO RMODE ANY
* ********************************************************************
* General settings.
* ********************************************************************
*
```

```
* TRACE FLAGS:   USED TO ACTIVATE TRACING
*
TRSYSLOG DC     XL2'001F'   TRACE TO SYSLOG
TRSYSLST DC     XL2'001F'   TRACE TO SYSLST
TRC_SERV EQU    X'0001'
TRC_CLNT EQU    X'0002'
TRC_DEC  EQU    X'0004'
TRC_ENC  EQU    X'0008'
TRC_HTTP EQU    X'0010'
*
* ********************************************************************
* Codepage specific settings.
* ********************************************************************
*
* SOAP SERVER EBCDIC CODEPAGE (IBM CCSID format)
* POSSIBLE VALUES ARE DESCRIBED IN THE MANUAL
* CICS Family: Communicating from CICS on System/390.
* See chapter Charater data
*
SERV_ECP DC     CL8'037'
*
* SOAP SERVER ASCII CODEPAGE
* POSSIBLE VALUES ARE DESCRIBED IN THE MANUAL (HTML coded format)
* CICS Transaction Server for VSE/ESA - Internet Guide
* See appendix: HTML-coded character sets
*
SERV_ACP DC     CL40'iso-8859-1'
*
*
* SOAP CLIENT EBCDIC CODEPAGE (ICONV FORMAT)
* POSSIBLE VALUES ARE DESCRIBED IN THE MANUAL
* LE/VSE C Run-Time Programming Guide
* See chapter Internationalization
*
CLNT_ECP DC     CL16'IBM-1047'
*
* SOAP CLIENT ASCII CODEPAGE (ICONV FORMAT)
* POSSIBLE VALUES ARE DESCRIBED IN THE MANUAL
* LE/VSE C Run-Time Programming Guide
* See chapter Internationalization
*
CLNT_ACP DC     CL16'UTF-8'
*
* ********************************************************************
* SSL specific settings. Used for VSE as SOAP client when the URL
* starts with https:// (secure HTTP).
* ********************************************************************
*
* Keyring library for SSL
*
SSLKEYR  DC     CL16'CRYPTO.KEYRING'
*
* Keyname for SSL
*
SSLKEYNM DC     CL8'HTTPSKEY'
*
* Cipher Specs for SSL.
* Specify all blanks to use the SSL libraries's supported ciphers.
*
SSLCIPSP DC     CL64'352F' 0A 09 08 02 01 ARE DEPRECATED
* For OpenSSL:
* SSLCIPSP  DC  CL64'C027C014C013C0126B673933163D3C352F'
*
* Session timeout for SSL
*
SSLTIMEO DC     F'86400'
*
```

```
* ********************************************************************
* Parameter name mapping (short names to long names):
* When receiving SOAP messages with paramater names that are longer
* than 16 chars (and would not fit into the SOAP_PROG_PARAM header),
* the following table is searched and the long name is replaced by the
* corresponding short name.
* When sending out SOAP messages, the names of all parameters are
* checked against the table, if the short name starts with the '#'
* character. If a matching short name is found, the parameter name
* is replaced by the corresponding long name.
* ********************************************************************
*
* Address of mapping table (see below)
*
PNAMETAB DC     A(NAMETAB)
*
* Short name indication (usually '#') for outgoing parameters
*
SNAMEIND DC     CL1'#'
*
* ********************************************************************
* Additional SSL specific settings. Used for VSE as SOAP client
* when the URL starts with https:// (secure HTTP).
* ********************************************************************
*
* SSL Type (e.g. 'SSL30' or 'TLSV1'. OpenSSL also supports
*           'SSLV3', 'TLS31', 'TLSV1.2' or 'ALL'
*           CSI also supports 'TLS12'.)
*
SSLTYPE  DC     CL15'TLSV1'
*
* ********************************************************************
* Support for specifying codepages per TCPIPSERVICE:
* In case different codepages are to be used for different TCPIP-
* SERVCIE definitions, you must supply table SERVTAB with the
* names of the TCPIPSERVICEes and the coresponding codepages.
* Codepage of a service overrule the default codepages specified
* in CLNT_ECP and CLNT_ACP.
* If no mathcing entry is found in the service table, then the
* default codepages from CLNT_ECP and CLNT_ACP are used.
* ********************************************************************
*
* Address of service table (see below)
*
PSERVTAB DC     A(SERVTAB)
*
* ********************************************************************
* Macro definition used for mapping table below.
* ********************************************************************
         MACRO
&NAME    ENTRY &SHORT=,&LONG=
&NAME    DC    Y(N&SYSNDX-*)    LENGTH OF THIS ENTRY
         DC    Y(L'S&SYSNDX)    LENGTH OF SHORT NAME
         DC    Y(L'L&SYSNDX)    LENGTH OF LONG NAME
S&SYSNDX DC    C&SHORT          SHORT NAME
L&SYSNDX DC    C&LONG           LONG NAME
N&SYSNDX DC    0H               START OF NEXT ENTRY
         MEND
*
* ********************************************************************
* Start of the mapping table. Entries are specified as follows:
*      ENTRY SHORT='short name',LONG='long name'
* ********************************************************************
*
NAMETAB  DS     0F              START OF TABLE
*
          ENTRY SHORT='#shortin',                                  X
```

```
              LONG='ThisIsAVeryLongInputWithMoreThan16Chars'
          ENTRY SHORT='#shortout',                                          X
              LONG='ThisIsAVeryLongOutputWithMoreThan16Chars'
*
NAMETEND DC     H'0'            END OF TABLE
*
* ********************************************************************
* Macro definition used for service table below.
* ********************************************************************
         MACRO
&SERV    SERVICE &NAME=,&ASCIICP=,&EBCDICCP=
&SERV    DC    Y(L&SYSNDX-*)     LENGTH OF THIS ENTRY
         DC    CL8&NAME          NAME OF TCPIPSERVICE
         DC    CL16&EBCDICCP     EBCDIC CODEPAGE (ICONV FORMAT)
         DC    CL16&ASCIICP      ASCII CODEPAGE (ICONV FORMAT)
L&SYSNDX DC    0H                START OF NEXT ENTRY
         MEND
*
* ********************************************************************
* Start of the service table. Entries are specified as follows:
*      SERVICE NAME='service',ASCIICP='asciicp',EBCDICCP='ebcdiccp'
* ********************************************************************
*
SERVTAB  DS     0F              START OF TABLE
*
          SERVICE NAME='CWS-US',                                        X
              ASCIICP='UTF-8',                                          X
              EBCDICCP='IBM-1047'  US ENGLISH
          SERVICE NAME='CWS-DE',                                        X
              ASCIICP='UTF-8',                                          X
              EBCDICCP='IBM-1141'  GERMANY
*
SERVTEND DC     H'0'            END OF TABLE
*
         END
/*
// IF $MRC GT 4 THEN
// GOTO NOLINK
// EXEC LNKEDT,PARM='MSHP'
* ********************************************************************
* DONT FORGET TO NEWCOPY THE IESSOAPO IN ORDER TO ACTIVATE IT:       *
*   CEMT SET PROG(IESSOAPO) NEWCOPY                                  *
* ********************************************************************
/. NOLINK
/*
/&
* $$ EOJ
```

長い名前の短い名前へのマッピング (SOAP エンジン・バージョン 1)

注: 本項は SOAP エンジン・バージョン 1 にのみ該当します。

TS キュー項目のサイズ制限のため、SOAP パラメーターでは、(SOAP_PROG_PARAM 制御ブロックに示されているように) 16 文字までの名前のみが可能です。

16 文字を超える SOAP パラメーターを使用するために、(16 文字を超える) 長い名前を (16 文字以下の) 短い名前にマップする独自のマッピングを提供できます。z/VSE SOAP エンジン・バージョン 1 は以下のように変換を行います。

- 長い名前を持つパラメーターを含んだ SOAP メッセージを受信したとき、長い名前を対応する短い名前に変換します。

- 長い名前を持つパラメーターを含んだ SOAP メッセージを送信するとき、短い名前を対応する長い名前に変換します。

長い名前に対応する短い名前は、『#』 文字で開始される必要があります。これにより z/VSE SOAP エンジンは、これが変換を必要とする名前であることを認識します。

SOAP エンジン・バージョン 1

このセクションでは、SOAP エンジン・バージョン 1 で使用可能な一連の例について説明します。

IBM 提供の SOAP サービス例 (getquote.c) の説明

IBM 提供の SOAP サービスのソース・コードは `getquote.c` です。このサンプルは VSE コネクター・クライアントの一部として提供されます。 VSE コネクター・クライアントのインストール方法については、 28 ページの『VSE コネクター・クライアントのインストール』のセクションを参照してください。この C (CICS 版) プログラムは、ディレクトリー **...¥<install-directory>¥samples¥soap¥vseSoapService** にあります。また、このディレクトリーには、 `getquote.c` をコンパイルしてリンクするためのジョブがあります (ジョブ `compile.job` およびジョブ `link.job`)。

1. SOAP サーバーによって SOAP サービスが呼び出される: 480 ページの図 159 に示されているように、プログラム `getquote` が SOAP サーバーによって (SOAP デコーダーを使用して) 呼び出されます。各要求には、以下のものが入っています。

- TargetObjectURI。これは、この要求のために、どのオブジェクトを呼び出すかを定義します。
- このオブジェクトで呼び出すメソッドのメソッド名。

`getquote` で使用される URI (Uniform Resource Identifier) は、`urn:iessoapd:getquote` です。 URI は常に `urn` で始まり、次のものの名前が続きます。

1. SOAP コンバーター (IBM 提供の `iessoapd`)
2. SOAP サービス (IBM 提供の C (CICS 用) プログラム `getquote`)

`iessoapd` の代わりに、ユーザー独自の SOAP コンバーターを使用することもできます。 IBM 提供のサンプルのメソッド名は `getQuote` で、これは、SOAP サービスに付けられた名前です。 SOAP サービスはこのメソッド名を使用して、実行する操作を決定します。 したがって、各 SOAP サービス (すなわち CICS プログラム) は、複数のメソッドを処理できます。

2. COMMAREA を SOAP_PROG_PARAM にマップする: SOAP サービスは、開始されると、提供されている CICS COMMAREA を指すポインターを取得する必要があります。COMMAREA は、 498 ページの図 166 に示すように、C 構造 SOAP_PROG_PARAM にマップしなければなりません。

```
SOAP_PROG_PARAM* call;
EXEC CICS ADDRESS COMMAREA(call);
```

図 *166. SOAP_PROG_PARAM* 制御ブロックへの *COMMAREA* のマッピング

3. 必要なメソッドを検査する**:** ここで、SOAP サービスは、図 167 に示されているように、要求されたメソッドがあるかどうかを検査できます。IBM 提供の SOAP サービスは 1 つのメソッド `getQuote` しかありません。その他のメソッド名はすべて、エラー・コード 1 を生成します。正しいメソッド名が要求された場合は、SOAP サービスは副次機能を呼び出し、 SOAP サーバーによって構築された 2 つの CICS 待ち行列を提供して、以下の 2 つのことを行います。

1. 指定された SOAP パラメーターを SOAP サービスに受け渡す (`inqueue` を使用する)
2. SOAP サービスから出力パラメーターを受け取る (`outqueue` を使用する)

```
// check if the SOAP method name is 'getQuote'
if(strncmp(call->method, "getQuote", 8) == 0 )
    rc = ProcessGetQuote(call->inqueue, call->outqueue);
else
    rc = 1;
```

図 *167.* どの *SOAP* メソッドが要求されたかの検査

4. 入力パラメーターを取得する**:** SOAP サービスは、CICS `inqueue` から入力パラメーターを読み取ることにより、これを順番に取得します。次に、SOAP サービスは、図 168 に示すように、読み取られた値を SOAP_PARAM_HEADER 構造にマップします。`pName` はパラメーターのエレメント名、 `pPtr` はデータを指すポインター、さらに、`pLen` はデータの長さです。

```
SOAP_PARAM_HDR* param;
unsigned short length;
EXEC CICS READQ TS QUEUE(inqueue)
    SET(param) LENGTH(len) NEXT
    RESP(resp) RESP2(resp2);
if(param->type != SOAP_TYPE_STRING)
    return 5; // invalid type (for this service)
*pName = (char*)&param>name;
*pPtr  = (char*)&param[1];
*pLen  = param->length - sizeof(SOAP_PARAM_HDR);
```

図 *168. CICS* 待ち行列からの入力パラメーターの取得

5. 値を **CICS** 出力待ち行列に入れる**:** パラメーターを CICS `outqueue` に入れるために、SOAP サービスは、 499 ページの図 169 に示すように、パラメーターを SOAP_PARAM_HEADER 構造にコピーし、その構造を CICS `outqueue` に書き込みます。

```
int SetNextOutParameter(char*outqueue,
        char *typename, unsigned int type,
        char* name, char* ptr,int len)
{
  SOAP_PARAM_HDR* param;
  int resp,resp2;
  param = (SOAP_PARAM_HDR*)malloc(sizeof(SOAP_PARAM_HDR)+len);
  if(param==NULL)
      return(-1);
  memset(param->name,' ',sizeof(param->name));
  memcpy(param->name,name,strlen(name));
  memset(param->typename,' ',sizeof(param->typename));
  memcpy(param->typename,typename,strlen(typename));
  param->type = type;
  param->length = len + sizeof(SOAP_PARAM_HDR);
  memcpy(&param[1],ptr,len);
  EXEC CICS WRITEQ TS QUEUE(outqueue)
            FROM(param) LENGTH(len+sizeof(SOAP_PARAM_HDR))
            RESP(resp) RESP2(resp2);
  free(param);
  return(0);
};
```

図 *169. CICS* 出力待ち行列へのパラメーターの配置

残りの処理**:** 残りの処理は、きわめて単純です。

1. SOAP サービスは、最初のパラメーターを CICS inqueue から読み取り、以下のことを検査します。
   - パラメーター名が symbol かどうか。
   - タイプが SOAP_TYPE_STRING かどうか。

   そうでない場合、SOAP サービスは、エラー・コードを SOAP サーバーに戻します。
2. 図示するために、SOAP サービスは、ハードコーディングされたシンボル値を使用し、この値 (パラメーターとして) を名前 data および SOAP_TYPE_STRING と一緒に CICS outqueue に入れます。
3. SOAP サービスはここで終了し、ゼロというエラー・コードを SOAP サーバーに (SOAP デコーダーを使用して) 戻します。SOAP サーバーは、戻された値の XML 表現を CICS outqueue に作成し、これを SOAP クライアントに戻します (480 ページの図 159 を参照)。

IBM 提供の SOAP クライアント例 (soapclnt.c) の説明

IBM 提供の SOAP クライアントのソース・コードは soapclnt.c です。このサンプルは VSE コネクター・クライアントの一部として提供されます。 VSE コネクター・クライアントのインストール方法については、 28 ページの『VSE コネクター・クライアントのインストール』のセクションを参照してください。この C (CICS 版) プログラムは、ディレクトリー **...¥<install-directory>¥samples¥soap¥vseSoapClient** にあります。また、このディレクトリーには、soapclnt.c をコンパイルしてリンクするためのジョブがあります (ジョブ compile.job およびジョブ link.job)。

1. SOAP サービスの呼び出しを準備する**:** プログラム soapclnt.c は、 483 ページの図 160 に示されているように、z/VSE ホストにある SOAP サービスを呼び出します。

各呼び出しには、以下のものが入っています。
- SOAP サーバーの URL (Uniform Resource Locator)
- SOAP サーバーによって呼び出されるプログラムの名前 (URI)
- 呼び出し先プログラムから要求されるメソッド (METHOD)

図 170 に例を示します。

```
char *URL = "http://9.164.155.95:1080/cics/CWBA/IESSOAPS";
char *URN = "urn:iessoapd:getquote";
char *METHOD = "getQuote";
```

図 *170. SOAP* クライアントの呼び出しパラメーターの準備

2. SOAP_DEC_PARAM 構造を準備する**:** SOAP クライアントは、図 171 に示されているように、呼び出しに必要な値を SOAP_DEC_PARAM タイプの構造にコピーすることによって、SOAP_DEC_PARAM 構造を準備します。

```
// prepare the call
strncpy(call.url, URL, strlen(URL));
strncpy(call.method, METHOD, strlen(METHOD));
strncpy(call.urn, URN, strlen(URN));
strncpy(call.inqueue, "INPUT   ",8);
strncpy(call.outqueue,"OUTPUT  ",8);
call.proxytype = 0;
call.proxytype = HTTP_TYPE_DIRECT;
call.authtype  = AUTH_NONE;
```

図 *171. SOAP* クライアントが *SOAP_DEC_PARAM* 構造を準備する

3. パラメーターを **SOAP** サーバーの **CICS** 入力待ち行列に挿入する**:** SOAP クライアントは、(SOAP サービスと同じように) CICS の inqueue および outqueue を使用して、パラメーターを伝送します。IBM 提供の例では、SOAP クライアントは、銘柄シンボル IBM の株価を要求します。応答で、図 172 に示すように、IBM という値を持った symbol というパラメーターおよび SOAP_TYPE_STRING というタイプが、CICS inqueue に挿入されます。

```
SetNextOutParameter("INPUT   ",
    "string", SOAP_TYPE_STRING,
    "symbol", "IBM", 3);
```

図 *172. SOAP* クライアントが、値を *SOAP* サーバーの入力待ち行列に挿入する

4. SOAP コンバーターを呼び出して要求を処理する**:** 呼び出しの準備が完了したので、SOAP クライアントは、図 173 に示されているように、SOAP コンバーター (IESSOAPE) を呼び出して SOAP クライアントの要求を処理することができます。SOAP コンバーターは次に、 483 ページの図 160 に示されているように、SOAP クライアント・プロセッサーを呼び出します。

```
EXEC CICS LINK PROGRAM("IESSOAPE")
    COMMAREA(&call) LENGTH(sizeof(call))
    RESP(rc) RESP2(rc2);
```

図 *173. SOAP* クライアントが、要求を処理するために *SOAP* コンバーター *(IESSOAPE)* を呼び出す

5. SOAP 呼び出しの結果を取得する**:** SOAP クライアントは、SOAP コンバーター (IESSOAPE) を呼び出した後で、 501 ページの図 174 に示されているように、こ

の呼び出しの結果を取得することができます。変数 *call.namespaceurl* には、返信の内容をエンコードするために呼び出された SOAP サービスによって使用されたネームスペース URL が入ります。 *namespace url* に値が入っている場合、URL はデコーダーに認知されていません。 この場合は、SOAP クライアントは、パラメーター自体の値を非直列化しなければなりません。

```
rc = GetNextInParameter("OUTPUT  ",&name,&val,&len);
if(rc!=0)
    break;
cicsprintf(  "name/type = %.32s", name );
cicsprintf(  "len       = %d", len);
sprintf(temp,"val       = %s%ds", "%.", len);
cicsprintf(temp,val);
```

図 *174. SOAP* クライアントが *SOAP* 呼び出しの結果を取得する

6. 作成された **CICS** 待ち行列を削除する**:** SOAP 呼び出しが完了すると、SOAP クライアントは、呼び出しの準備の際に自動的に作成された一時的な CICS 待ち行列をここで削除する必要があります。これによって、すべてのメモリーが必ず CICS に戻されます。 これは、図 175 に示します。

```
EXEC CICS DELETEQ TS QUEUE("INPUT   ")
    RESP(rc) RESP2(rc2);
EXEC CICS DELETEQ TS QUEUE("OUTPUT  ")
    RESP(rc) RESP2(rc2);
```

図 *175. SOAP* クライアントが *CICS* 待ち行列を削除する

Java SOAP クライアントの使用例

C (CICS 用) で作成された IBM 提供の SOAP クライアント (soapclnt.c) を使用する代わりに、Java SOAP クライアントを使用できます。

SOAP クライアント GetQuote.java は、Apache SOAP の配布と一緒に提供されます。 このサンプルは、Apache SOAP パッケージに入っている Apache AXIS ディレクトリー **axis-bin-1_4¥axis-1_4¥samples¥stock¥GetQuote.java** にあります (詳細については 503 ページの『ステップ 1: Java SOAP クライアント・パッケージのクライアントでのダウンロードおよびインストール』を参照)。

また、IBM で変更した GetQuote.java のバージョンが、ディレクトリー **...¥<install-directory>¥samples¥soap¥javasample** にあります。これは、このプログラムをコンパイルして実行するバッチ・ファイルと一緒に入っています。 このサンプルは VSE コネクター・クライアントの一部として提供されます。 VSE コネクター・クライアントのインストール方法については、 28 ページの『VSE コネクター・クライアントのインストール』のセクションを参照してください。このサンプルは、 502 ページの図 176 の強調表示したステートメントで示されているように、IBM 提供の getQuote SOAP サービスを呼び出せるように変更されています。

```
...
Call     call    = (Call) service.createCall();
call.setTargetEndpointAddress( url );
call.setOperationName( new QName("urn:iessoapd:getquote", "getQuote") );
call.addParameter( "symbol", XMLType.XSD_STRING, ParameterMode.IN );
call.setReturnType( XMLType.XSD_STRING );
Object ret = call.invoke( new Object[] {symbol} );
...
```

図 *176. SOAP* クライアントの *getQuote* サービスの呼び出し

おわかりのように、この呼び出しのセットアップは、SOAP クライアントが z/VSE ホストで実行されるときのセットアップに似ています (500 ページの図 171 を参照)。 Java クライアントのサンプルは、z/VSE ホストで実行される SOAP クライアントのサンプル (soapclnt) と同じ処理を実行します。

ユーザーは、run.bat ファイル (ディレクトリー **...¥<install-directory>¥samples¥soap¥javasample** にあります) を変更して、SOAP サーバーの URL および要求されたシンボルがコマンド行上でプログラムに指定されるようにしなければなりません。 詳しくは、『IBM 提供の SOAP サンプルの実行』を参照してください。

IBM 提供の SOAP サンプルの実行

IBM 提供の SOAP サンプルは、以下の 3 つのプログラムから成っています。

- GetQuote.java (Java 使用可能プラットフォームで実行される SOAP クライアント)
- getquote.c (z/VSE ホスト上で SOAP サービスをインプリメントする)
- soapclnt.c (z/VSE ホストが SOAP クライアントとして機能するときに使用される)

これらのサンプルは VSE コネクター・クライアントの一部として提供されます。 VSE コネクター・クライアントのインストール方法については、 28 ページの『VSE コネクター・クライアントのインストール』のセクションを参照してください。

プログラム getquote.c は、GetQuote.java または soapclnt.c のどちらかから、SOAP サービスとして呼び出すことができます。 IBM 提供の SOAP サンプルは、z/VSE ホストが、以下のものとして機能するシナリオの両方を示します。

- SOAP クライアント (soapclnt.c が SOAP クライアントであるとき)
- SOAP サーバー (GetQuote.java が SOAP クライアントであるとき)

IBM 提供の SOAP サンプルを実行するには、このトピックで説明するステップを実行する必要があります。ただし、この SOAP サンプルを実行するには、以下の条件を満たす必要があります。

- Java Development Kit (JDK) 1.5 以降がインストールされていること。JDK 1.5 以降をインストールしていない場合のインストール方法について詳しくは、 24 ページの『Java のインストールと構成』を参照してください。
- IBM *C* コンパイラー *(VSE/ESA* 版*)* がインストールされている。

ステップ 1: Java SOAP クライアント・パッケージのクライアントでのダウンロードおよびインストール

必要なファイルをまだインストールしていない場合は、さまざまなパッケージをダウンロードする必要があります。 必要なパッケージは、Apache AXIS パッケージです。これは、次の URL から入手できます。

```
http://ws.apache.org/axis/
```

この Web サイトで、最新バージョンがあるダウンロード・ディレクトリーに進み、axis-bin パッケージ (例えば、**axis-bin-1_4.zip**) をダウンロードします。

ステップ 2: 必要な Java プログラムを抜き出し、インストールする

ステップ **2.1:** 新規ディレクトリーを作成する**:** このステップでは、新規ディレクトリーを作成します。 CLASSPATH の定義を単純化するために、SOAP サンプルの実行に必要なすべての .JAR ファイルは、このディレクトリーに保管します。 以下のステップでは、このディレクトリーは **work** と呼びます。

ステップ **2.2: JAR** ファイルを抜き出し、ご使用のディレクトリーに入れる**:** このステップでは、以下に示すように、必要な **.JAR** ファイルをダウンロードされた ZIP ファイル **axis-bin-1_4.zip** から抜き出し、抜き出された JAR ファイルをディレクトリー **work** に入れます。

- axis.jar
- commons-discovery-0.2.jar
- commons-logging-1.0.4.jar
- jaxrpc.jar
- log4j-1.2.8.jar
- saaj.jar
- wsdl4j-1.5.1.jar

ステップ 3: サンプル C プログラムをコンパイル/リンクし、CICS に定義する

IBM 提供の C (CICS 用) プログラム (SOAP クライアントおよび SOAP サービス) の使用を始めるには、これらのプログラムのソースをコンパイルしてリンクし、そのプログラムを CICS に定義しなければなりません。 したがって、以下の作業が必要になります。

1. SOAP クライアント (soapclnt.c) の中の URL を、ご使用の z/VSE システムに一致するように変更します。サンプルの URL は、以下のとおりです。

   ```
   http://9.164.123.23:1080/cics/CWBA/IESSOAPS
   ```

   (通常は、IP アドレスを変更するだけで済みます。)
2. 以下のソース
   - ディレクトリー **...¥<install-directory>¥samples¥soap¥vseSoapClient** にある soapclnt.c
   - ディレクトリー **...¥<install-directory>¥samples¥soap¥vseSoapService** にある getquote.c

   を、z/VSE ライブラリーにアップロードします。 TCP/IP **ftp** ユーティリティー、または、3270 エミュレーターで提供されているファイル転送機能を使用します。

3. ソース・プログラム soapclnt.c および getquote.c を、IBM 提供のジョブ compile.job および link.job を使用してコンパイルします。 これらのジョブは、関係のあるソース・プログラム (soapclnt.c または getquote.c) と同じディレクトリーにあります。これらのジョブは、ご使用の環境に合わせて変更する (例えば、ソースおよび宛先のライブラリー名など) 必要がある場合があります。 コンパイル・ジョブは、以下の処理を実行します。
   a. CICS プリコンパイラーを呼び出す。
   b. C コンパイラーを呼び出す。
   c. オブジェクト・デックを、コンパイル・ジョブの中で指定した z/VSE ライブラリーにカタログする。

   リンク・ジョブは、以下の処理を実行します。
   a. LE プリリンカーを呼び出す。
   b. リンケージ・エディターを呼び出す。
   c. フェーズを、リンク・ジョブの中で指定した z/VSE ライブラリーにカタログする。
4. CEDA トランザクションを使用して、SOAPCLNT プログラムおよび GETQUOTE プログラムを CICS に定義します。また、上記で作成したフェーズは、(CICS の LIBDEF ステートメントを使用して) CICS で探し出せるようにする必要があります。 図 177 に、GETQUOTE プログラムの場合の CEDA の使用方法を示します。

```
CEDA DEFINE PROGRAM
OVERTYPE TO MODIFY
 CEDA  DEFine PROGram(          )
  PROGram      ==> GETQUOTE
  Group        ==> VSESPG
  DEscription  ==> SAMPLE SOAP SERVICE GETQUOTE
  Language     ==> C                 COBol | Assembler | C | Pli
  RELoad       ==> No                No | Yes
  RESident     ==> No                No | Yes
  USAge        ==> Normal            Normal | Transient
  USEsvacopy   ==> No                No | Yes
  Status       ==> Enabled           Enabled | Disabled
  RSl            : 00                0-24 | Public
  Cedf         ==> Yes               Yes | No
  DAtalocation ==> Any               Below | Any
  EXECKey      ==> User              User | Cics
 REMOTE ATTRIBUTES
  REMOTESystem ==>
  REMOTEName   ==>
  Transid      ==>
```

図 *177.* サンプルの *SOAP* サービスを *CICS* に定義するための *CEDA* の使用

ステップ 4: CICS Web サポートを使用するように CICS を構成する

ここで、CICS Web サポートを構成および活動化して、z/VSE で Web サービスを使用するようにする必要があります。

これを実行する方法について詳しくは、「*CICS Transaction Server for VSE/ESA* インターネット・ガイド」 (SC88-8667) の『CICS Web サポートの構成』を参照してください。

ステップ 5: SOAP サーバーを CICS に定義する

IBM 提供の SOAP サーバーをアクティブにするには、TCP/IP サービスを CICS に定義する必要があります。 すべての SOAP クライアント (C (CICS 用) または Java のどちらで作成されていても) は、ユーザーがこのステップを実行することを必要とします。 以下に示す CEDA DEFINE を実行します。

```
CEDA   DEFine TCpipservice( SOAP     )
 TCpipservice   : SOAP
 Group          : VSESPG
 Description  ==> TCP/IP SERVICE FOR SOAP SEVRER
 Urm          ==> DFHWBADX
 Portnumber   ==> 01080              1-65535
 Certificate  ==>
 STatus       ==> Open               Open | Closed
 SSl          ==> No                 Yes | No | Clientauth
 Attachsec    ==> Local              Local | Verify
 TRansaction  ==> CWXN
 Backlog      ==> 00001              0-32767
 TSqprefix    ==>
 Ipaddress    ==>
 SOcketclose  ==> No                 No | 0-240000
```

図 *178. SOAP* サーバーを *CICS* に定義するための *CEDA* の使用

ステップ 5: ASCII から EBCDIC へのコンバーターをアクティブにする

ASCII / EBCDIC コンバーター (DFHCNV) は、VSE/ICCF ライブラリー 59 に提供されています。これは、以下の変換を行うために CICS Web サポートによって (したがって、さらに SOAP サーバーによって) 使用されます。

- 着呼 XML データを ASCII から EBCDIC に
- 発信 XML データを EBCDIC から ASCII に

ASCII / EBCDIC コンバーターをアクティブにするには、スケルトン DFHCNV を VSE/POWER 読み取り待ち行列にサブミットします。

ステップ 6: Java サンプルをコンパイルする

1. 以下のファイルを、ディレクトリー **...¥<install-directory>¥samples¥soap¥javasample** から、ご使用のディレクトリー **work** にコピーします。
   - compile.bat
   - run.bat
   - GetQuote.java
2. ファイル `compile.bat` を実行します。

ステップ 7: Java SOAP クライアントのサンプルを実行する

1. テキスト・エディターでファイル `run.bat` をオープンし、URL を、ご使用の z/VSE システムに一致するように変更します。 サンプルの URL は、以下のとおりです。

   ```
   http://9.164.123.23:1080/cics/CWBA/IESSOAPS
   ```

   (通常は、IP アドレスを変更するだけで済みます。)

2. run.bat ファイルを開始します。 すると、SOAP 呼び出しが、SOAP クライアントから、z/VSE ホスト上で実行中の SOAP サーバーに送信され、銘柄シンボル IBM の株価が要求されます。 すると、z/VSE ホストで実行中のサンプルの SOAP サービスは、ハードコーディングされた銘柄シンボルの株価を Java プログラムに戻します。
3. Java SOAP クライアントは、株価 (プログラムに「ハードコーディング」されています) を画面に出力します。

ステップ 8: C プログラムの SOAP クライアントのサンプルを実行する

C プログラム soapclnt.c を実行するには、以下を実行する必要があります。

1. 503 ページの『ステップ 3: サンプル C プログラムをコンパイル/リンクし、CICS に定義する』で定義した CICS プログラム SOAPCLNT を呼び出す CICS トランザクションを定義する (これも CEDA を使用する)。 注: SOAP クライアント (soapclnt.c) の中の URL を、ご使用の z/VSE システムに一致するように変更したことを確認してください。
2. 上記で定義した CICS トランザクションを開始します。 すると、SOAP 呼び出しが、SOAP クライアントから、z/VSE ホスト上で実行中の SOAP サーバーに送信され、銘柄シンボル IBM の株価が要求されます。 すると、z/VSE ホストで実行中のサンプルの SOAP サービスは、ハードコーディングされた銘柄シンボルの株価を C プログラムに戻します。

ユーザー独自の SOAP プログラムの作成

ユーザー独自のプログラムを作成する際に、ユーザーがテンプレートとして使用できる IBM 提供のサンプルは、以下のものから成っています。

SOAP C 言語ヘッダー・ファイル IESSOAPH.H
: 485 ページの『IBM 提供の SOAP 制御ブロックの使用方法』に説明があります。 z/VSE SOAP 実装で使用される COMMAREA またはコンテナーとメモリー・マッピングは、IESSOAPH.H で定義されています。 SOAP サービスまたはクライアントを別の言語 (COBOL など) で作成したい場合は、ユーザー自身がこれらの定義をこの言語に適応させる必要があります。 アセンブラー言語用のマッピングは、コマンドとして IESSOAPH.H に組み込まれています。

SOAP サービス getquote.c
: CICS 用に C で作成されています。 497 ページの『IBM 提供の SOAP サービス例 (getquote.c) の説明』に説明があります。

SOAP クライアント soapclnt.c
: CICS 用に C で作成されています。 499 ページの『IBM 提供の SOAP クライアント例 (soapclnt.c) の説明』に説明があります。

SOAP クライアント GetQuote.java
: Java で作成されています。 501 ページの『Java SOAP クライアントの使用例』に説明があります。

SOAP コンバーターである IESSOAPD または IESSOAPE (あるいは両方) がユーザーの要件に合わない場合は、これらを、ユーザー独自のプログラムで置き換えることができます。 z/VSE インバウンド方向の SOAP コンバーターの名前を、

SOAP 要求の URI に指定します。 詳細については、 497 ページの『IBM 提供の SOAP サービス例 (getquote.c) の説明』を参照してください。

第 25 章 REST を使用したプログラム間通信

このトピックでは、Representational State Transfer (省略形 REST) を使用して、CICS プログラムと他のモジュール間で情報を送受信し、インターネット経由で RESTful Web サービスを構築し使用する方法について説明します。

以下の項目があります。

- 『REST のための z/VSE サポートの概要』
- 510 ページの『z/VSE ホストが REST サーバーとして機能する方法』
- 511 ページの『z/VSE ホストが REST クライアントとして機能する方法』
- 512 ページの『IBM 提供の REST 制御ブロックの使用方法』
- 527 ページの『REST エンジンの構成』

実行する作業	参照先
REST について詳しくは、次の URL にあるウィキペディア記事を参照してください。	https://en.wikipedia.org/wiki/Representational_state_transfer

REST のための z/VSE サポートの概要

Representational State Transfer (REST) は、Web サービスのガイドラインとベスト・プラクティスから構成される、ソフトウェア・アーキテクチャー・スタイルです。 REST は、SOAP や WSDL ベースの Web サービスに対する最もシンプルな代替として、Web で幅広い支持を受けています。 RESTful システムは一般的に、Hypertext Transfer Protocol (HTTP) を介し、Web ブラウザーで使用されるものと同じ HTTP verb (GET、POST、PUT、 DELETE など) を使用して通信を行います。

RESTful Web サービスによって転送されるペイロード (メッセージ) にはさまざまなタイプ (コンテンツ・タイプ) があります。よく使用されるのは JSON と XML ですが、プレーン・テキストまたはバイナリー・データも使用できます。

RESTful Web サービスは一般的に、サーバー上の特定の「オブジェクト」に対して操作を行います。オブジェクトは一般的に、http://example.com/resource のような URI (URL の一部) によってアドレス指定されます。

このようなリソースに対するアクションは通常、GET、PUT、POST、DELETE などの HTTP 要求タイプによって指示されます。 GET は一般的にリソースを読み取ります。 PUT はリソースを更新/置換し、POST はリソースを作成し、DELET はリソースを削除します。

RESTful Web サービスは一般的にステートレスです。クライアントからの要求にはそれぞれ、要求に対してサービスを提供するために必要なすべての情報が含まれています。そのため、セッション状態がクライアントに保持されます。

上記の説明の「一般的に」という語で示されるように、説明されている特性に関してはいずれも厳格な要件はありません。

z/VSE REST サポートにより、z/VSE は RESTful Web サービス環境に加わることができます。 z/VSE CICS TS アプリケーションは、RESTful Web サービスとして外部世界に提供でき (サーバーとしての z/VSE)、またどこか他の場所でホストされている RESTful Web サービスのコンシューム/使用/呼び出しを行うこともできます (クライアントとしての z/VSE)。

z/VSE REST エンジンは、要求と応答の送受信、ASCII/EBCDIC 変換 (必要な場合)、XML または JSON の解析 (XML または JSON タイプのペイロードの場合) など、すべての HTTP 関連アクションを実行します。

お客様提供のプログラムは、HTTP 要求タイプ、URI、およびペイロードに基づいて必要なアクションを実行する役割を果たします。

z/VSE ホストが REST サーバーとして機能する方法

図 179 は、RESTful Web サービスを提供する REST サーバーとして z/VSE ホストが機能するときに CICS TS 環境において REST がどのように使用できるのかを示しています。

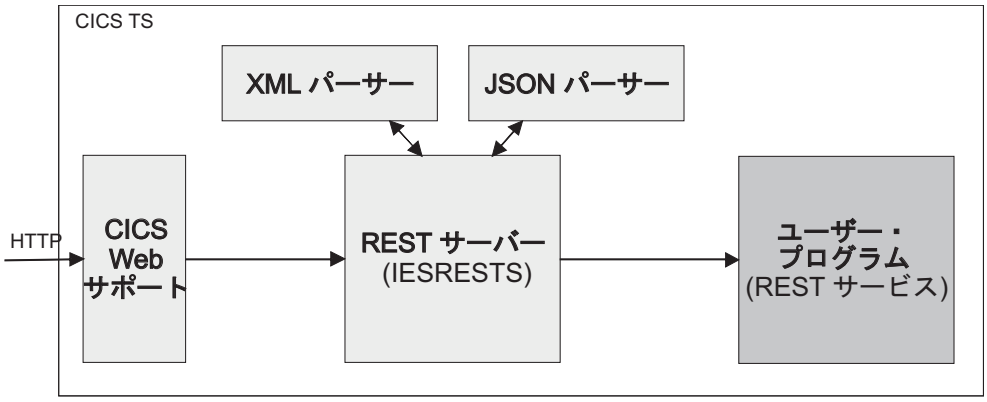


図 *179. z/VSE* が *REST* サーバーとして機能するときに関係するモジュール

RESTful Web サービス・プロバイダー (サーバー) としての z/VSE の場合、HTTP 要求は CICS Web サポート (CWS) によって受信されます。 z/VSE 提供の RESTful Web サービスを開始する URL は必ず以下のようになります。

http://host:port/cics/CWBA/IESRESTS/userprog[/recource-uri][?query-string][#fragment]

この URL により CICS Web サポートは CICS 提供のトランザクション CWBA の下でプログラム IESRESTS (z/VSE REST エンジンのサーバー部分) を呼び出します。 RESTful Web サービスを実施するユーザー提供のプログラムの名前は上記のように URL の一部として指定されます。 REST エンジンは、コンテンツ・タイプに基づいてデータ変換を行い、さらに解析も行います (XML または JSON データ・タイプの場合)。その後、REST エンジンはユーザー提供のプログラムを呼び出し、HTTP 要求タイプ (GET、PUT、POST、DELETE など)、リソース URI (指定されている場合)、URL パラメーターを含む照会ストリング (指定されている場合)、追加 HTTP ヘッダー (指定されている場合)、ペイロードのコンテンツ・タイ

プ、およびペイロード自体 (メモリー内のバッファーとして、または解析された XML/JSON ツリーとして) を渡します。ユーザー提供のプログラムは、渡された情報を使用して、必要なアクションを実行し、最後に REST エンジンに応答を返します。ユーザー提供のプログラムは、応答コード、応答コンテンツ・タイプ、および応答ペイロードを返します。その後で REST エンジンは応答コンテンツ・タイプに基づいてデータ変換を行い、XML/JSON データ生成を行い、CICS Web サービスを使用して応答をリモート・システム上の REST クライアントに送り返します。

IBM 提供の REST サーバー・サンプル COBRESTS.c

IBM 提供の REST サーバー・サンプル COMRESTS.c は、z/VSE REST エンジンを使用して RESTful Web サービスを提供する方法を示します。このサンプルは VSE コネクター・クライアントの一部として提供されます。 VSE コネクター・クライアントのインストール方法については、 28 ページの『VSE コネクター・クライアントのインストール』のセクションを参照してください。 VSE コネクター・クライアント・インストールのサンプル・ディレクトリーには、COBRESTS.c をコンパイルしてリンクするジョブ (ジョブ COBRESTS.job) もあります。

z/VSE ホストが REST クライアントとして機能する方法

図 180 は、z/VSE ホストが REST クライアントとして機能するときに CICS 環境において RESTful Web サービスをどのように使用できるのかを示しています。

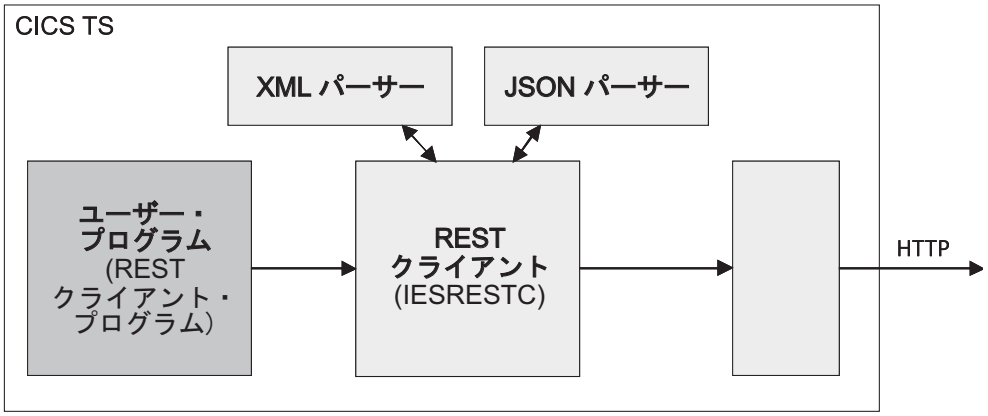


図 *180. z/VSE* が *REST* クライアントとして機能するときに関係するモジュール

RESTful Web サービス・コンシューマー (クライアント) としての z/VSE の場合、ユーザー提供のプログラムは REST エンジン (プログラム IESRESTC) を呼び出し、呼び出す RESTful Web サービスの URL、HTTP 要求タイプ (GET、PUT、POST、DELETE など)、照会ストリング URL パラメーター (要求された場合)、追加 HTTP ヘッダー (要求された場合)、ペイロードのコンテンツ・タイプ、およびペイロード自体 (メモリー内のバッファーとして、または解析された XML/JSON ツリーとして) を REST エンジンに渡します。その後で REST エンジンはコンテンツ・タイプに基づいてデータ変換を行い、XML/JSON データ生成を行い、z/VSE HTTP クライアントを呼び出してサーバーとの HTTP 通信を行います。応答が受信され、REST エンジンに返されます。次に REST エンジンは、応答

コンテンツ・タイプに基づいてデータ変換を行い、さらに解析も行います (XML データ・タイプまたは JSON データ・タイプの場合)。その後、REST エンジンはユーザー提供のプログラムを再び呼び出し、HTTP 応答コード、追加 HTTP 応答ヘッダー、ペイロードのコンテンツ・タイプ、およびペイロード自体 (メモリー内のバッファーとして、または解析された XML/JSON ツリーとして) を返します。

IBM 提供の REST クライアント・サンプル COBRESTC.c

IBM 提供の REST クライアント・サンプル COBRESTC.c は、z/VSE REST エンジンを使用して外部 RESTful Web サービスを呼び出す方法を示します。このサンプルは VSE コネクター・クライアントの一部として提供されます。 VSE コネクター・クライアントのインストール方法については、 28 ページの『VSE コネクター・クライアントのインストール』のセクションを参照してください。 VSE コネクター・クライアント・インストールのサンプル・ディレクトリーには、COBRESTC.c をコンパイルしてリンクするジョブ (ジョブ COBRESTC.job) もあります。

IBM 提供の REST 制御ブロックの使用方法

このトピックでは、C 言語ヘッダー・ファイル IESRESTH.H、COBOL コピーブック IESRESTL.C、PL/1 コピーブック IESJSONP.P、またはアセンブラー・マクロ IESRESTA.A に定義されている IBM 提供の REST 制御ブロックについて説明します。これらのメンバーは PRD1.BASE 内にあります。また、ヘッダー・ファイル、コピーブック、およびマクロは VSE コネクター・クライアントの一部としてディレクトリー ...¥<install-directory>¥samples¥rest にも提供されています。

ユーザー・プログラム (REST サーバーとしての z/VSE – プログラム IESRESTS) を呼び出したりユーザー・プログラム (REST クライアントとしての z/VSE – プログラム IESRESTC) によって呼び出されたりするために REST エンジンによって使用される COMMAREA レイアウトについて次の表で説明します。

注: REST サーバーとしての z/VSE の場合、COMMAREA で使用される TS キュー名は REST エンジンによって生成されます (ただし、すべての TS キューが実際に存在して項目が含まれているとは限りません)。 REST クライアントとしての z/VSE の場合、TS キュー名は、出力として指定されているフィールドに関してもユーザー・プログラムによって COMMAREA で生成されて指定されなければなりません。そのような出力タイプ TS キューについては、REST エンジンが項目を TS キューに投入します (名前が指定されている場合のみ)。

フィールド	タイプ	説明
Version	int	COMMAREA レイアウトのバージョン (現行バージョンは 1)

EBCDICCodepage	char[16]	EBCDIC コード・ページ (オプション、ブランク可) • サーバーとしての z/VSE の場合は、REST オプションによって、使用されるコード・ページが決まります。このフィールドには、現在使用されている EBCDIC コード・ページが含まれています。 • クライアントとしての z/VSE の場合、ユーザーは、使用するコード・ページを呼び出しごとに設定できます。このフィールドがブランクのままにされた場合は、REST オプションで指定されたコード・ページが使用されます。 REST オプションが指定されていない場合は、デフォルトの IBM-1047 が使用されます。
Flags	uint	処理フラグ • X'00000001': CLEANUP 呼び出しを要求します。 – サーバーとしての z/VSE の場合、ユーザー・プログラムは、ツリー/バッファーを解放するためにトランザクションの終わりに Action=CLEANUP で再びユーザー・プログラム自体が呼び出されることを要求するために、このビットを設定できます。 CLEANUP 呼び出しは、エラー状態に関係なく最後の呼び出しとして実行されます。 – クライアントとしての z/VSE の場合、REST エンジンは、この REST エンジンがツリー/バッファーを解放できるようにユーザー・プログラムが REST エンジンを Action=CLEANUP で再び呼び出すことを要求するために、このビットを設定できます。ユーザー・プログラムがその CLEANUP 呼び出しを実行しないと、CICS がトランザクションの終わりに CICS GETMAIN 記憶域を自動的に解放するまでは、その記憶域はそのまま解放されない可能性があります。 • X'00000002': ERROR 呼び出しを要求します。 – サーバーとしての z/VSE の場合、ユーザー・プログラムは、応答を送り返すときにエラーが発生した場合に備えて Action=ERROR で再びユーザー・プログラム自体が呼び出されることを要求するために、このビットを設定できます。 ERROR 呼び出しは CLEANUP 呼び出しの前に行われます (CLEANUP 呼び出しも要求された場合)。 – これは、クライアントとしての z/VSE には使用されません。そこでは、会話が失敗したことは、RetCode フィールドと RespHTTPStatusCode フィールドの一方または両方にゼロ以外の戻りコードが返されることで示されます。その際、ユーザー・プログラムは、その失敗に対して、必要な処置を行うことができます。

RetCode	int	戻りコード: • サーバーとしての z/VSE の場合: 戻りコードは REST エンジンへの戻り時にユーザー・プログラムによって設定されます。戻りコードがゼロ以外の場合、REST エンジンは HTTP 状況「500 Server Error」を送り返します。 • クライアントとしての z/VSE の場合: REST エンジン戻りコード (ユーザー・プログラムへの戻り時に REST エンジンによって設定される)。注: 戻りコードがゼロ以外の場合、通常はサーバーとの通信で障害が発生しています (例: ホストが見つからない)。エラーを示す HTTP 状況コードでの応答の場合、RetCode フィールドはゼロですが、HTTP 状況コードはフィールド RespHTTPStatusCode に返されます。
Private	ptr	ユーザー・プライベート・フィールド • サーバーとしての z/VSE の場合: このフィールドは、ユーザー割り振り制御ブロックのトークンまたはアドレスを保管するために REST エンジンへの戻り時にユーザー・プログラムによって設定できます。 REST エンジンはこのフィールドにアクセスすることもこのフィールドを変更することもしませんが、このフィールドを、後に続く当該会話の CLEANUP 呼び出しや ERROR 呼び出しに渡します。このため、ユーザー・プログラムは最初の呼び出しから後続のすべての呼び出しの情報を保存できます。 • クライアントとしての z/VSE の場合: REST エンジンはこのフィールドをユーザー・プログラムへの戻り時に任意の値に設定できます。ユーザー・プログラムはこのフィールドにアクセスすることもこのフィールドを変更することもしませんが、このフィールドを、後に続く当該会話の任意の CLEANUP 呼び出しや ERROR 呼び出しで REST エンジンに渡します。
以下のフィールドには要求に関する情報が含まれています。それらの要件は、次のとおりです。 • サーバーとしての z/VSE の場合は REST エンジンによって設定される • クライアントとしての z/VSE の場合はユーザー・プログラムによって設定される		
ReqAction	int	実行するアクション (HTTP メソッド): 1 = GET 2 = PUT 3 = POST 4 = DELETE 5 = HEAD 6 = OPTIONS 7 = PATCH 特別なアクション: 0 = CLEANUP – フラグの要求を受けての終結処理呼び出し。 -1 = ERROR – フラグの要求を受けてのエラー呼び出し。

ReqURL	char[2048]	• サーバーとしての z/VSE の場合: リソース URI。 REST エンジンは URI の、照会ストリング (存在する場合) が除外されたユーザー・プログラム名 (http://host:port/cics/CWBA/IESRESTS/user-program/recource-uri) の後の部分のみを渡します。照会ストリングは REST エンジンによって取り出されて解析されます。 URL エンコードのパラメーターが、フィールド ReqUrlParamsTSQ によって示される TS キューでユーザー・プログラムに渡されます。 • クライアントとしての z/VSE の場合: http または https の URL (プロトコル (例: 「http://」)、ホスト、ポート (オプション)、およびリソース URI を含む)。 URL パラメーターは除外され、代わりにフィールド ReqUrlParamsTSQ で示される TS キューによって渡される必要があります。この URL には、URL パラメーターが TS キューによって指定されるときに保持される (つまり、末尾に移動される) フラグメント (#nnn) を含むことができます。
ReqContentType	char[128]	HTTP コンテンツ・タイプ (ストリング) (例: 「text/xml; charset=UTF-8」) (オプション、ブランク可)。 • サーバーとしての z/VSE の場合: これは通知のためだけにユーザー・プログラムに渡されます。結果として、コンテンツ・タイプは REST エンジンによって処理され、コンテンツ (存在する場合) が受信されて変換されます。フィールド ReqDataType には、ユーザー・プログラムに渡されるデータのタイプが含まれます。 • クライアントとしての z/VSE の場合: これが指定されている (非ブランクである) 場合は、フィールド ReqDataType に指定されているデータ・タイプに一致しなければなりません。例えば、ReqDataType=3 (XML) の場合、コンテンツ・タイプは「text/xml」か「application/xml」、あるいは適合する他の任意の XML データ用コンテンツ・タイプでなければなりません。コンテンツ・タイプが指定されていない場合 (すべてブランクの場合)、REST エンジンは ReqDataType フィールドに基づいてコンテンツ・タイプを作成します。コンテンツ・タイプに文字セット (例: 「...;charset=UTF-8」) が指定されている場合、その情報は REST エンジンによって使用されます。文字セットが指定されていないは、REST オプションで指定されているコード・ページが使用されます。 REST オプションが指定されていない場合は、デフォルトの UTF-8 が使用されます。その場合、REST エンジンは、サーバーに送信されるコンテンツ・タイプに、適切な文字セットの指定を追加します。

ReqDataType	int	要求データ・タイプ: 0 = データなし 1 = バイナリー・バッファー (ReqDataPtr はバッファーを指します) 2 = テキスト・バッファー (ReqDataPtr はバッファーを指します) 3 = XML ツリー (ReqDataPtr は XML ツリー・ルートを指します) 4 = JSON ツリー (ReqDataPtr は JSON ツリー・ルートを指します) 5 = URL エンコードのフォーム・フィールド (ReqDataPtr は NULL ですが、ReqFormFieldsTSQ にはフォーム・フィールドが含まれます) 6 = マルチパート・データ (ReqDataPtr はマルチパート構造体の配列を指します)
ReqDataPtr	ptr	要求データを指すポインター (詳しくは、フィールド ReqDataType を参照してください)
ReqDataLength	uint	ReqDataPtr によって指し示されるバッファーの長さ。使用可能なデータがない場合はゼロ、XML/JSON ツリー・ルートの場合は -1 です。 マルチパート・データの場合、これは、ReqDataPtr によって指し示される配列内のエレメントの数を指定します。
ReqUrlParamsTSQ	char[8]	URL エンコードのパラメーターを含む TS キューの名前 (オプション、ブランク可)。 TS キュー項目レイアウトについて詳しくは、以下を参照してください。 TS キュー名はすべてのアクティブ・タスクにわたって固有でなければなりません。そのため、同一タスクの異なる TS キューを識別するために、7 桁の CICS タスク ID (EIBTASKN) に 1 文字 (例:「U」) を加えて名前を作成する必要があります。 • サーバーとしての z/VSE の場合: URL 照会ストリングは REST エンジンによって解析されます。 URL エンコードのパラメーターが、このフィールドによって示される TS キューでユーザー・プログラムに渡されます。 • クライアントとしての z/VSE の場合: TS キューが指定されていて存在し、そこに URL パラメーターが含まれている場合、そのパラメーターは、フィールド URL で指定されている URL に (URL エンコードのパラメーターとして) 追加されます。フラグメント (#nnn) はいずれも URL パラメーターの後ろにある URL の末尾に移動されます。

ReqFormFieldsTSQ	char[8]	フォーム・フィールドを含む TS キューの名前 (オプション、ブランク可)。 TS キュー名はすべてのアクティブ・タスクにわたって固有でなければなりません。そのため、同一タスクの異なる TS キューを識別するために、7 桁の CICS タスク ID (EIBTASKN) に 1 文字 (例:「F」) を加えて名前を作成する必要があります。 • サーバーとしての z/VSE の場合: 要求コンテンツ・タイプが「application/x-www-form-urlencoded」の場合、REST エンジンはコンテンツからフォーム・フィールドを抽出し、それを、このフィールドで示される TS キューでユーザー・プログラムに渡します。この場合、フィールド ReqDataType は 5 (URL エンコードのフォーム・フィールド) に設定されます。 • クライアントとしての z/VSE の場合: フィールド ReqDataType が 5 (URL エンコードのフォーム・フィールド) を指定していて、TS キューが指定されていて存在し、そこにフォーム・フィールド項目が含まれている場合、REST エンジンはフォーム・フィールドを送信します。フィールド ReqContentType はすべてブランクであるか、または「application/x-www-form-urlencoded」を指定していなければなりません。
ReqHTTPHeadersTSQ	char[8]	HTTP ヘッダーを含む TS キューの名前 (オプション、ブランク可)。 TS キュー項目レイアウトについて詳しくは、以下を参照してください。 TS キュー名はすべてのアクティブ・タスクにわたって固有でなければなりません。そのため、同一タスクの異なる TS キューを識別するために、7 桁の CICS タスク ID (EIBTASKN) に 1 文字 (例:「H」) を加えて名前を作成する必要があります。 • サーバーとしての z/VSE の場合: REST エンジン自体で処理されない HTTP ヘッダーはユーザー・プログラムに渡されます。 • クライアントとしての z/VSE の場合: TS キューが指定されていて存在する場合、そこに含まれる HTTP ヘッダーはいずれも HTTP 要求とともに送信されます。ただし、REST エンジン自体で作成された HTTP ヘッダー (「Content-Type」や「Content-Length」など) はオーバーライドできず無視されます。

ReqCookiesTSQ	char[8]	Cookie を含む TS キューの名前 (オプション、ブランク可)。 TS キュー項目レイアウトについて詳しくは、以下を参照してください。 TS キュー名はすべてのアクティブ・タスクにわたって固有でなければなりません。そのため、同一タスクの異なる TS キューを識別するために、7 桁の CICS タスク ID (EIBTASKN) に 1 文字 (例:「C」) を加えて名前を作成する必要があります。 • サーバーとしての z/VSE の場合: 要求に「Cookie」HTTP ヘッダーが含まれている場合、REST エンジンは情報を抽出して Cookie を TS キューでユーザー・プログラムに渡します。各 Cookie は独自の TS キュー項目で表されます。この場合、Cookie の名前と値のみが TS キュー項目で設定されます。 • クライアントとしての z/VSE の場合: TS キューが指定されていて存在する場合、そこに含まれる Cookie はいずれも、ドメイン/パスに一致していて有効期限切れでなければ、サーバーに「Cookie」HTTP ヘッダーとして送信されます。 TS キューに含まれる Cookie には、適切な属性情報 (Domain/Path や Expiration/MaxAge など) が含まれていなければなりません。 REST エンジンはこのような属性を使用して、一致する Cookie (サーバーに送信すべき Cookie) を検索します。
ReqAuthType	int	認証タイプ: 0 = 認証なし 1 = 基本 HTTP 認証 2 = SSL クライアント認証 (サーバーとしての z/VSE の場合にのみ利用可能)
ReqAuthUser	char[64]	基本 HTTP 認証用のユーザー名 (オプション、ブランク可)。 • サーバーとしての z/VSE の場合: 要求に認証情報が含まれている場合、REST エンジンは情報を抽出してユーザー・プログラムに渡します。 – ReqAuthType=1 (基本 HTTP 認証) の場合、フィールド AuthReqUser にはユーザー名が含まれ、ReqAuthPassword にはパスワードが含まれます。 – ReqAuthType=2 (SSL クライアント認証) の場合、フィールド ReqAuthUser には、証明書に対してマップされたユーザー ID が含まれるか、ユーザーがマップされていなければブランクが含まれます。フィールド ReqAuthPassword には、バイナリー証明書データが含まれた 1 つの項目を保持する TS キューの名前が含まれます。 • クライアントとしての z/VSE の場合: 指定されたユーザー名とパスワードがサーバーに「Authorization: Basic ...」HTTP ヘッダーとして送信されます。
ReqAuthPassword	char[64]	基本 HTTP 認証用のパスワード (オプション、ブランク可)。詳しくは、上のフィールド ReqAuthUser を参照してください。

ReqAccept	char[128]	HTTP Accept 指定 (オプション、ブランク可)。 • サーバーとしての z/VSE の場合: 要求に Accept HTTP ヘッダーが含まれている場合、REST エンジンは情報を抽出してこのフィールドに挿入します。 • クライアントとしての z/VSE の場合: これが指定されている (非ブランクである) 場合、REST エンジンはこの情報をサーバーに Accept HTTP ヘッダーとして送信します。これが指定されない場合は、Accept ヘッダー「*/*」が送信されます。
以下のフィールドには応答に関する情報が含まれています。それらの要件は、次のとおりです。 • サーバーとしての z/VSE の場合はユーザー・プログラムによって設定される • クライアントとしての z/VSE の場合は REST エンジンによって設定される		
RespHTTPStatusCode	int	3 桁の HTTP 状況コード (例: OK の場合は 200)。有効な状況コードについては、RFC2616 を参照してください。
RespHTTPStatusText	char[128]	状況テキスト (例: 状況コード 200 の場合は「OK」)。(オプション、ブランク可)。 • サーバーとしての z/VSE の場合: これがブランクのままである場合、REST エンジンは (RFC2616 の勧告に従って) 適切な状況テキストを指定します。
RespContentType	char[128]	HTTP コンテンツ・タイプ (ストリング) (例: 「text/xml; charset=UTF-8」) (オプション、ブランク可)。 • サーバーとしての z/VSE の場合: これが指定されている (非ブランクである) 場合は、フィールド RespDataType に指定されているデータ・タイプに一致しなければなりません。例えば、RespDataType=3 (XML) の場合、コンテンツ・タイプは「text/xml」か「application/xml」、あるいは適合する他の任意の XML データ用コンテンツ・タイプでなければなりません。コンテンツ・タイプが指定されていない場合 (すべてブランクの場合)、REST エンジンは RespDataType フィールドに基づいてコンテンツ・タイプを作成します。コンテンツ・タイプに文字セット (例: 「...;charset=UTF-8」) が指定されている場合、その情報は REST エンジンによって使用されます。文字セットが指定されていないは、REST オプションで指定されているコード・ページが使用されます。使用可能な REST オプションがない場合は、デフォルトの UTF-8 が使用されます。 その場合、REST エンジンは、サーバーに送信されるコンテンツ・タイプに、適切な文字セットの指定を追加します。 • クライアントとしての z/VSE の場合: これは通知のためだけにユーザー・プログラムに渡されます。結果として、コンテンツ・タイプは REST エンジンによって処理され、コンテンツ (存在する場合) が受信されて変換されます。フィールド RespDataType には、ユーザー・プログラムに渡されるデータのタイプが含まれます。

RespDataType	int	応答データ・タイプ: 0 = データなし 1 = バイナリー・バッファー (RespDataPtr はバッファーを指します) 2 = テキスト・バッファー (RespDataPtr はバッファーを指します) 3 = XML ツリー (RespDataPtr は XML ツリー・ルートを指します) 4 = JSON ツリー (RespDataPtr は JSON ツリー・ルートを指します) 5 = URL エンコードのフォーム・フィールド (RespDataPtr は NULL ですが、RespFormFieldsTSQ にはフォーム・フィールドが含まれます) 6 = マルチパート・データ (RespDataPtr はマルチパート構造体の配列を指します)
RespDataPtr	ptr	応答データを指すポインター (詳しくは、フィールド RespDataType を参照してください)
RespDataLength	uint	RespDataPtr によって指し示されるバッファーの長さ。使用可能なデータがない場合はゼロ、XML/JSON ツリー・ルートの場合は -1 です。 マルチパート・データの場合、これは、RespDataPtr によって指し示される配列内のエレメントの数を指定します。
RespFormFieldsTSQ	char[8]	フォーム・フィールドを含む TS キューの名前 (オプション、ブランク可)。 TS キュー名はすべてのアクティブ・タスクにわたって固有でなければなりません。そのため、同一タスクの異なる TS キューを識別するために、7 桁の CICS タスク ID (EIBTASKN) に 1 文字 (例:「R」) を加えて名前を作成する必要があります。 • サーバーとしての z/VSE の場合: TS キューが存在し、そこにフォーム・フィールドが含まれていて、フィールド RespContentType がすべてブランクであるか「application/x-www-form-urlencoded」を指定していて、フィールド RespDataType が 5 (フォーム・フィールド) である場合、REST エンジンはフォーム・フィールドをコンテンツ・タイプ「application/x-www-form-urlencoded」として送信します。 • クライアントとしての z/VSE の場合: 応答コンテンツ・タイプが「application/x-www-form-urlencoded」であり、TS キューが指定されている場合、REST エンジンはコンテンツからフォーム・フィールドを抽出し、それを、このフィールドで示される TS キューでユーザー・プログラムに渡します。この場合、フィールド RespDataType は 5 (フォーム・フィールド) に設定されます。

RespHTTPHeadersTSQ	char[8]	HTTP ヘッダーを含む TS キューの名前 (オプション、ブランク可)。 TS キュー項目レイアウトについて詳しくは、以下を参照してください。 TS キュー名はすべてのアクティブ・タスクにわたって固有でなければなりません。そのため、同一タスクの異なる TS キューを識別するために、7 桁の CICS タスク ID (EIBTASKN) に 1 文字 (例:「E」) を加えて名前を作成する必要があります。 • サーバーとしての z/VSE の場合: TS キューが存在する場合、そこに含まれる HTTP ヘッダーはいずれも HTTP 応答とともに送信されます。ただし、REST エンジン自体で作成された HTTP ヘッダー (「Content-Type」や「Content-Length」など) はオーバーライドできず無視されます。 • クライアントとしての z/VSE の場合: REST エンジン自体で処理されない HTTP ヘッダーはユーザー・プログラムに渡されます (TS キュー名が指定されている場合)。
RespCookiesTSQ	char[8]	Cookie を含む TS キューの名前 (オプション、ブランク可)。 TS キュー項目レイアウトについて詳しくは、以下を参照してください。 TS キュー名はすべてのアクティブ・タスクにわたって固有でなければなりません。そのため、同一タスクの異なる TS キューを識別するために、7 桁の CICS タスク ID (EIBTASKN) に 1 文字 (例:「O」) を加えて名前を作成する必要があります。 • サーバーとしての z/VSE の場合: TS キューが存在する場合、そこに含まれる Cookie はいずれも「Set-Cookie」HTTP ヘッダーとしてクライアントに送信されます。 TS キューに含まれる Cookie には、適切な属性情報 (Domain/Path や Expiration/MaxAge など) が含まれていなければなりません。 REST エンジンはこのような属性を使用して、一致する Cookie (サーバーに送信すべき Cookie) を検索します。 • クライアントとしての z/VSE の場合: 応答に「Set-Cookie」HTTP ヘッダーが含まれていて TS キューが指定されている場合、REST エンジンは情報を抽出して TS キューでユーザー・プログラムに渡します。各 Cookie は独自の TS キュー項目で表されます。同じ名前の Cookie が TS キューに既に含まれている場合、その名前は REST エンジンによって更新されます。
RespLocation	char[2048]	Location HTTP ヘッダー。通常、これは応答「201 Created」、「301 Moved permanently」、「302 Found」、「303 see other」、「305 Use proxy」、または「307 Temporary redirect」に使用されます。 • サーバーとしての z/VSE の場合: これが指定されている (非ブランクである) 場合、このフィールドの値はクライアントに「Location」HTTP ヘッダーとして送信されます。 • クライアントとしての z/VSE の場合: 応答に「Location」HTTP ヘッダーが含まれている場合、REST エンジンは情報を抽出してこのフィールドでユーザー・プログラムに渡します。ただし、REST エンジンは自動的にはリダイレクト (例: 「301 Moved permanently」) を追跡しません。要求を新規ロケーションに再送するのはユーザー・プログラムの責任となります。

HTTP ヘッダー、URL パラメーター、および Form フィールドに使用される TS キュー項目のレイアウトについて次の表で説明します。

フィールド	タイプ	説明
名前	char[128]	HTTP ヘッダー、URL パラメーター、または Form フィールドの名前。 HTTP ヘッダー名に大/小文字の区別はありませんが、URL パラメーター名やフォーム・フィールド名では大/小文字が区別されます (実装に依存)。末尾ブランクはこの名前には含まれず、REST エンジンによって自動的に削除されます。
値	char[*]	HTTP ヘッダー、URL パラメーター、または Form フィールドの値。この値の長さ (1 文字から 32639 文字まで) は TS キュー項目の長さによって決定されます。末尾ブランクはこの値には含まれず、REST エンジンによって自動的に削除されます。

Cookie に使用される TS キュー項目のレイアウトについて次の表で説明します。

フィールド	タイプ	説明
名前	char[128]	Cookie の名前。末尾ブランクはこの名前には含まれず、REST エンジンによって自動的に削除されます。 Cookie 名では大/小文字が区別されます。
Domain	char[2048]	Domain 属性は Cookie の有効範囲を定義します (オプション、ブランク可)。 Cookie のドメインやパスがサーバーで指定されていない場合は、要求されたリソースのドメインとパスがデフォルトで使用されます。
Path	char[2048]	Path 属性は Cookie の有効範囲を定義します (オプション、ブランク可)。 Cookie のドメインやパスがサーバーで指定されていない場合は、要求されたリソースのドメインとパスがデフォルトで使用されます。
Expires	char[48]	Expires 属性は、HTTP クライアントが Cookie を削除しなければならない具体的な日時を定義します。この日時は「Wdy, DD Mon YYYY HH:MM:SS GMT」の形式で指定されます (オプション、ブランク可)。
Secure	char	Secure 属性は、Cookie による通信を暗号化された送信に限定するためのものです。この属性により、クライアントはセキュアな/暗号化された接続 (すなわち HTTPS) を使用する場合にのみ Cookie を使用することになります。このフィールドに指定できる値は「Y」または「N」です。 (オプション、ブランク可)
HttpOnly	char	HttpOnly 属性は、クライアントが HTTP (および HTTPS) 要求以外のチャネルでは Cookie を公開しないようにするものです。この属性を持つ Cookie には、HTTP 以外の方法ではアクセスできません。このフィールドに指定できる値は「Y」または「N」です。 REST エンジンは HTTP (または HTTPS) チャネルのみをサポートするため、この属性は実際には何の効果もありません。(オプション、ブランク可)
値	char[*]	Cookie の値。この値の長さ (1 文字から 28493 文字まで) は TS キュー項目の長さによって決定されます。末尾ブランクはこの値には含まれず、REST エンジンによって自動的に削除されます。

マルチパート・データに使用される配列項目のレイアウトについて次の表で説明します。

フィールド	タイプ	説明
PartContentType	char[128]	当該部分の HTTP コンテンツ・タイプ (ストリング) (例: 「text/xml; charset=UTF-8」) (オプション、ブランク可)。 このフィールドの使用法は ReqContentType や RespContentType (マルチパート・データが要求に使用されるのか応答に使用されるのかによって決まる) の場合と同じです。
PartDataType	int	当該部分のデータ・タイプ: 0 = データなし 1 = バイナリー・バッファー (PartDataPtr はバッファーを指します) 2 = テキスト・バッファー (PartDataPtr はバッファーを指します) 3 = XML ツリー (PartDataPtr は XML ツリー・ルートを指します) 4 = JSON ツリー (PartDataPtr は JSON ツリー・ルートを指します) 5 = URL エンコードのフォーム・フィールド (PartDataPtr は NULL ですが、PartFormFieldsTSQ にはフォーム・フィールドが含まれます)
PartDataPtr	ptr	部分データを指すポインター (詳しくは、フィールド PartDataType を参照してください)
PartDataLength	uint	PartDataPtr によって指し示されるバッファーの長さ。使用可能なデータがない場合はゼロ、XML/JSON ツリー・ルートの場合は -1 です。
PartFormFieldsTSQ	char[8]	フォーム・フィールドを含む TS キューの名前 (オプション、ブランク可)。 TS キュー名はすべてのアクティブ・タスクにわたって固有でなければなりません。そのため、同一タスクの異なる TS キューを識別するために、7 桁の CICS タスク ID (EIBTASKN) に 1 文字 (例:「a」) を加えて名前を作成する必要があります。 • 発信マルチパート・データの場合: TS キューが存在し、そこにフォーム・フィールドが含まれていて、フィールド PartContentType がすべてブランクであるか「application/x-www-form-urlencoded」を指定していて、フィールド PartDataType が 0 (データなし) である場合、REST エンジンはフォーム・フィールドをコンテンツ・タイプ「application/x-www-form-urlencoded」として送信します。 • 着信マルチパート・データの場合: 応答コンテンツ・タイプが「application/x-www-form-urlencoded」の場合、REST エンジンは TS キュー名を生成し、コンテンツからフォーム・フィールドを抽出し、それを、このフィールドで示される TS キューでユーザー・プログラムに渡します。この場合、フィールド PartDataType は 0 (データなし) に設定されます。
PartContentDisposition	char[128]	当該部分のコンテンツ配置。

REST エンジンが XML 内容を扱う方法

要求または応答の内容の一部として受け取った XML データはすべて、REST エンジンによって (z/VSE XML パーサーを使用して) 解析され、相互にポイントされる制御ブロックの形式で XML ツリーとしてユーザー・プログラムに渡されます。ユーザー・プログラムが XML データを送信するために REST エンジンに渡す場合は、このような制御ブロックの XML ツリーを作成し、そのツリー・ルートを REST エンジンに渡すことも必要です。

XML 関連制御ブロックのマッピングについては、PRD1.BASE にある C ヘッダー・ファイル IESXMLAH.H、COBOL コピーブック IESXMLCB.C、 PL/1 コピーブック IESXMLPL.P、またはアセンブラー・マクロ IESXMLAS.A を参照してください。

下図は、次の XML 内容を表す XML ツリーを示しています。

```
<?xml version="1.0" encoding="UTF-8"?>
<VSAMmap type="temporary" status="test">
  <!-- VSAM map layout -->
  <mapname>mapname12</mapname>
  <field>
    <name>str</name>
    <type>STRING</type>
  </field>
  <field>
    <name>sign</name>
    <type>SIGNED</type>
  </field>
</VSAMmap>
```

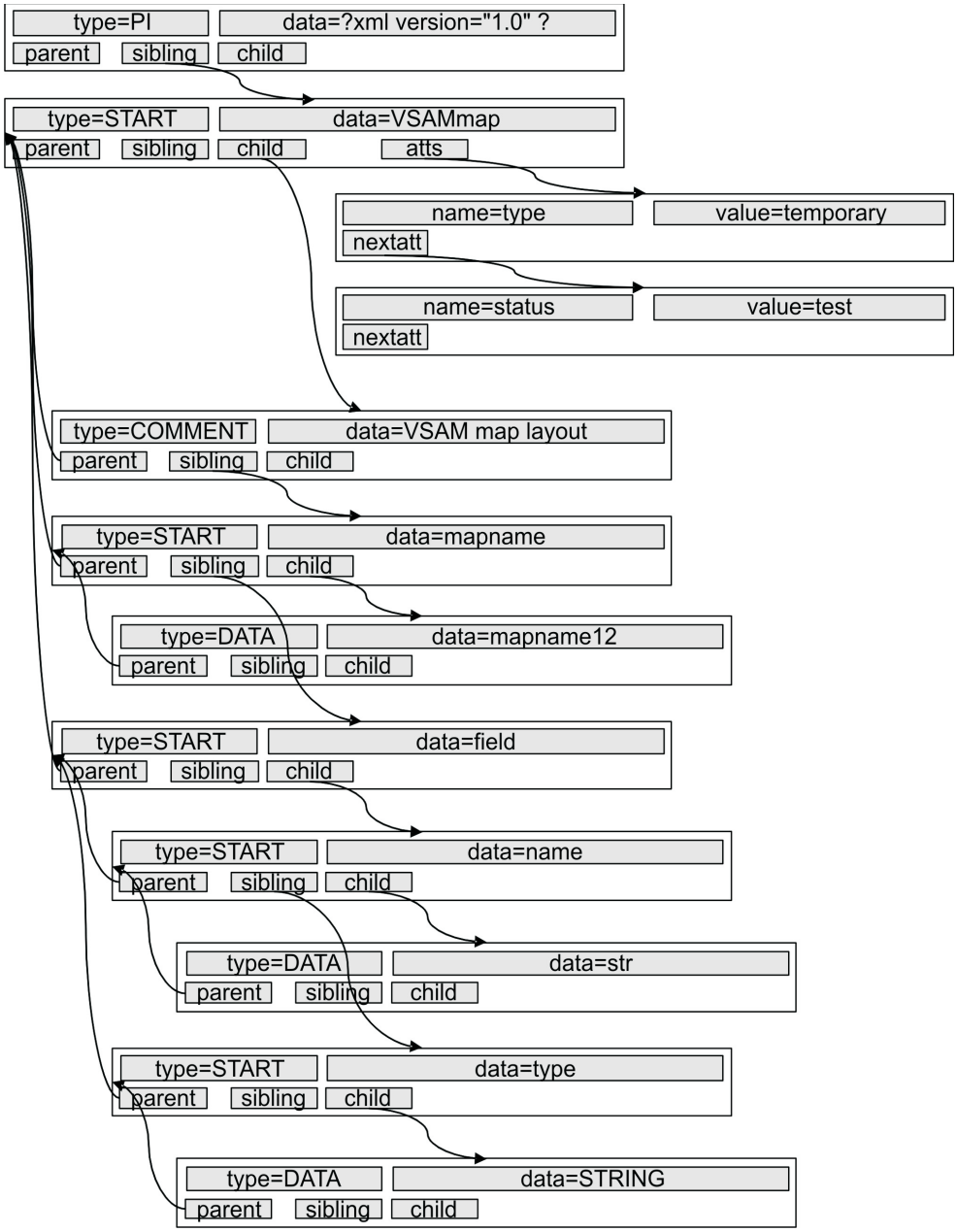


図 *181.* 上記 *XML* 内容を表す *XML* ツリーの図

REST エンジンが JSON 内容を扱う方法

要求または応答の内容の一部として受け取った JSON データはすべて、REST エンジンによって (z/VSE JSON パーサーを使用して) 解析され、相互にポイントされる制御ブロックの形式で JSON ツリーとしてユーザー・プログラムに渡されます。ユーザー・プログラムが JSON データを送信のために REST エンジンに渡す場合は、このような制御ブロックの JSON ツリーを作成し、そのツリー・ルートを REST エンジンに渡すことも必要です。

詳しくは、PRD1.BASE にある C ヘッダー・ファイル IESJSONH.H、COBOL コピーブック IESJSONC.C、PL/1 コピーブック IESJSONP.P、またはアセンブラー・マクロ IESJSONA.A を参照してください。

下図は、次の JSON 内容を表す JSON ツリーを示しています。

```
{"menu": {
  "id": "file",
  "popup": {
    "menuitem": [
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

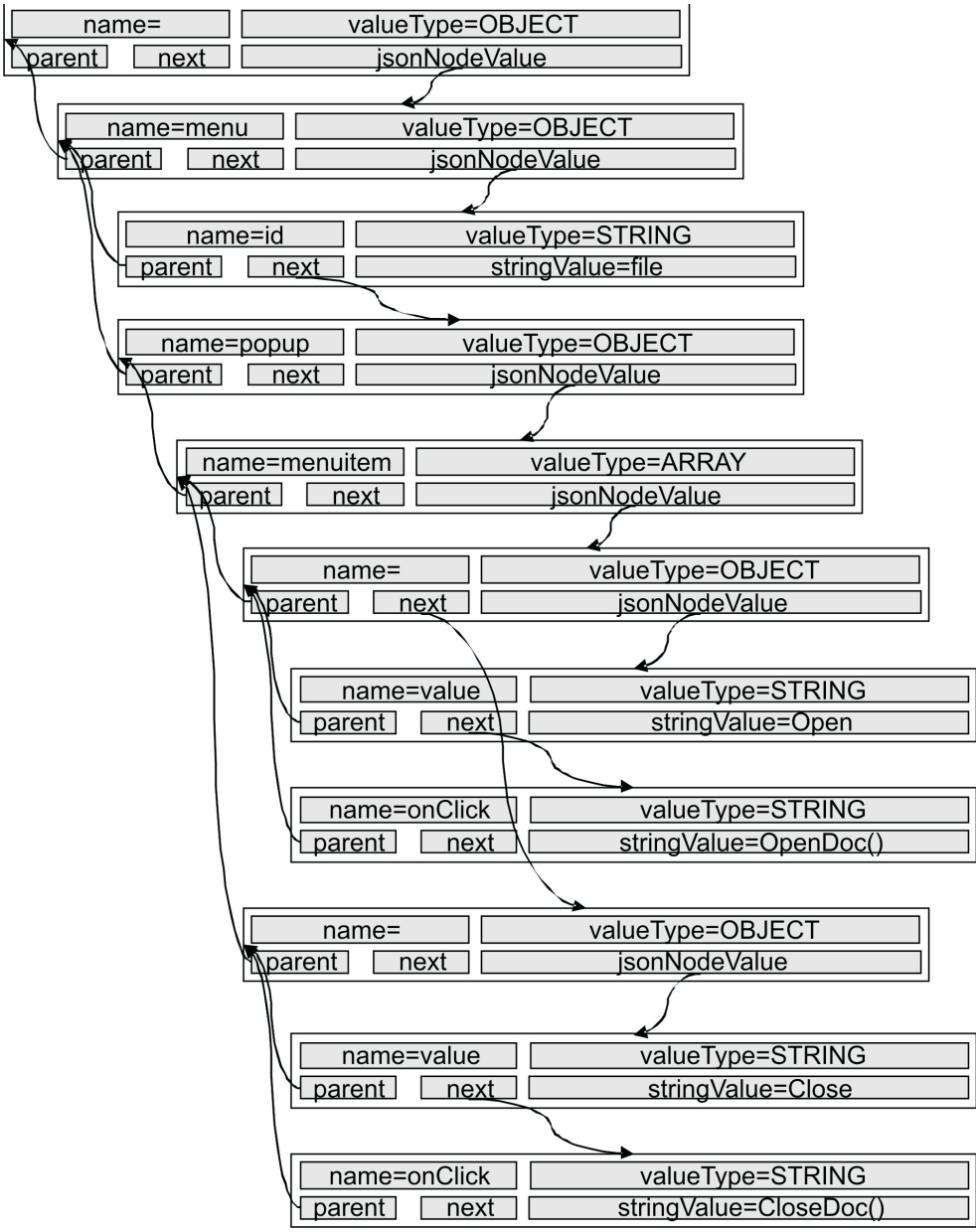


図 *182.* 上記 *JSON* 内容を表す *JSON* ツリーの図

REST エンジンの構成

REST エンジンを使用すると、オプション・フェーズ (IESRESTO) を生成することによって複数のオプションと設定を構成することができます。 REST オプションは、実行時に REST エンジンによってロードされます。 IESRESTO が使用できない場合は、ハードコーディングされたデフォルトが代わりに使用されます。 REST オプション・フェーズは、ICCF ライブラリー 59 にあるスケルトン SKRESTOP によって生成できます。

以下のオプションを構成できます。

- トレース・フラグ: REST エンジンの各種サブコンポーネントでトレースをオンにできます。
- デフォルト **EBCDIC** コード・ページ: ASCII から EBCDIC への変換に使用する EBCDIC コード・ページ (ICONV フォーマット) を指定します。デフォルトは「IBM-1047」です。サーバーとしての z/VSE の場合、コード・ページは TCPIPSERVICE ごとに構成することもできます (下記参照)。クライアントとしての z/VSE の場合、ユーザー・プログラムは、REST エンジンの呼び出しごとに必要な EBCDIC コード・ページを指定できます (COMMAREA フィールド EBCDICCodepage)。
- デフォルト **ASCII** コード・ページ: ASCII から EBCDIC への変換に使用する ASCII コード・ページ (ICONV フォーマット) を指定します。デフォルトは「UTF-8」です。サーバーとしての z/VSE の場合、コード・ページは TCPIPSERVICE ごとに構成することもできます (下記参照)。ユーザー・プログラムは、必要な文字セットをコンテンツ・タイプ・フィールドで指定できます (COMMAREA フィールド ReqContentType および RespContentType)。
- **SSL/TLS** 固有の設定: HTTPS URL 使用時のクライアントとして z/VSE の場合、これらの SSL/TLS 設定はデータの SSL/TLS 暗号化を制御します。
  - **SSL** タイプ: SSL/TLS タイプ (プロトコル・バージョン)。デフォルトは「TLSV1」です。
  - **SSL** 鍵リング: 鍵と証明書を含む鍵リング・ライブラリー。デフォルトは「CRYPTO.KEYRING」です。
  - **SSL** 鍵名: 使用する鍵素材の名前。デフォルトはありません。 SSL/TLS (HTTPS) を使用するには、このオプションを指定する必要があります。
  - **SSL** 暗号仕様: SSL/TLS 接続を確立するために使用する暗号仕様。デフォルトは、SSL 実装がサポートされている暗号として定義する内容です。
  - **SSL** セッション・タイムアウト: SSL/TLS セッションがタイムアウトになり、再ネゴシエーションされるまでの時間 (秒)。デフォルトは 86400 です。
- プロキシー/ソケット固有の設定: クライアントとしての z/VSE の場合のみ:
  - プロキシー・タイプ: プロキシーを使用するかどうか、およびそのタイプ (プロキシーなし、HTTP プロキシー、SocksV4、または SocksV5)。
  - プロキシー・ホスト: プロキシー/Socks サーバーの IP またはホスト名。プロキシーを使用する場合に必要です。
  - プロキシー・ポート: プロキシー/Socks サーバーのポート番号。プロキシーを使用する場合に必要です。
  - プロキシー・ユーザー: Socks サーバーでの認証に使用するユーザー名。SocksV4 または SockV5 を使用する場合に必要です。
  - プロキシー・パスワード: Socks サーバーでの認証に使用するパスワード。SockV5 を使用する場合に必要です。
- **TCPIPSERVICE** ごとのコード・ページ設定 **(**サーバーとしての **z/VSE** にのみ該当**):** TCPIPSERVICE 名のリストを提供し、使用する ASCII コード・ページと EBCDIC コード・ページを指定できます。ここで指定しない TCPIPSERVICE については、上で構成されたデフォルトの ASCII コード・ページおよび EBCDIC コード・ページが使用されます。
  - サービス名: この設定が適用される TCPIPSERVICE の名前。

- **EBCDIC** コード・ページ**:** ASCII から EBCDIC への変換に使用する EBCDIC コード・ページ (ICONV フォーマット) を指定します。すべてブランクにした場合は、上で指定されたデフォルトの EBCDIC コード・ページが使用されます。
- **ASCII** コード・ページ**:** ASCII から EBCDIC への変換に使用する ASCII コード・ページ (ICONV フォーマット) を指定します。すべてブランクにした場合は、上で指定されたデフォルトの ASCII コード・ページが使用されます。

REST エンジン構成の例**:**

```
* $$ JOB JNM=RESTOPT,CLASS=A,DISP=D
// JOB RESTOPT GENERATE REST OPTION PHASE
* ********************************************************************
* ASSEMBLE AND LINK THE REST OPTION PHASE                            *
* ********************************************************************
// LIBDEF *,CATALOG=PRD2.CONFIG
// LIBDEF *,SEARCH=PRD1.BASE
// OPTION ERRS,SXREF,SYM,NODECK,CATAL,LISTX
   PHASE IESRESTO,*,SVA
// EXEC ASMA90,SIZE=(ASMA90,64K),PARM='EXIT(LIBEXIT(EDECKXIT)),SIZE(MAXC
               -200K,ABOVE)'
IESRESTO CSECT
IESRESTO AMODE ANY
IESRESTO RMODE ANY
*
OPT_LEN  DC    A(OPT_END)       Lengtgh of fixed part
*
* ********************************************************************
* General settings.
* ********************************************************************
*
* TRACE FLAGS:   USED TO ACTIVATE TRACING
*
TRSYSLOG DC     XL2'0017'   TRACE TO SYSLOG
TRSYSLST DC     XL2'0017'   TRACE TO SYSLST
TRC_SERV EQU    X'0001'
TRC_CLNT EQU    X'0002'
TRC_DETA EQU    X'0004'
TRC_HTTP EQU    X'0010'
*
* ********************************************************************
* Codepage specific settings.
* ********************************************************************
*
* REST DEFAULT EBCDIC CODEPAGE (ICONV FORMAT)
* POSSIBLE VALUES ARE DESCRIBED IN THE MANUAL
* LE/VSE C Run-Time Programming Guide
* See chapter Internationalization
*
DFLT_ECP DC     CL16'IBM-1047'
*
* REST DEFAULT ASCII CODEPAGE (ICONV FORMAT)
* POSSIBLE VALUES ARE DESCRIBED IN THE MANUAL
* LE/VSE C Run-Time Programming Guide
* See chapter Internationalization
*
DFLT_ACP DC     CL16'UTF-8'
*
* ********************************************************************
* SSL specific settings. Used for VSE as REST client when the URL
* starts with https:// (secure HTTP).
* ********************************************************************
*
```

```
* SSL Type (e.g. 'SSL30' or 'TLS31')
*
SSLTYPE  DC     CL16'TLS31'
*
* Keyring library for SSL
*
SSLKEYR  DC     CL16'CRYPTO.KEYRING'
*
* Keyname for SSL
*
SSLKEYNM DC     CL8'SAMPLE'
*
* Cipher Specs for SSL.
* Specify all blanks to use the SSL libraries's supported ciphers.
*
SSLCIPSP DC     CL64'352F' 0A 09 08 02 01 ARE DEPRECATED
* For OpenSSL:
* SSLCIPSP  DC  CL64'C027C014C013C0126B673933163D3C352F'
*
* Session timeout for SSL
*
SSLTIMEO DC     F'86400'
*
* ********************************************************************
* HTTP Proxy/Socks specific settings.
* Used for VSE as REST client only.
* ********************************************************************
*
* Proxy type
*
PRXYTYPE DC    F'0'
PRXYNONE EQU   0          No Proxy
PRXYHTTP EQU   1          HTTP Proxy
PRXYSKS4 EQU   2          Socks V4
PRXYSKS5 EQU   3          Socks V5
*
* IP or hostname of proxy/socks server (if PRXYTYPE != PRXYNONE)
*
PRXYHOST DC   CL256''
*
* Port number of proxy/socks
*
PRXYPORT DC   F'80'
*
* Socks User ID (for PRXYTYPE = PRXYSKS4 or PRXYSKS5)
*
PRXYUSER DC  CL128''
*
* Socks Password (for PRXYTYPE = PRXYSKS5)
*
PRXYPASS DC  CL128''
*
*
* ********************************************************************
* Support for specifying codepages per TCPIPSERVICE:
* In case different codepages are to be used for different TCPIP-
* SERVCIE definitions, you must supply table SERVTAB with the
* names of the TCPIPSERVICEes and the coresponding codepages.
* Codepage of a service overrule the default codepages specified
* in DFLT_ECP and DFLT_ACP.
* If no mathcing entry is found in the service table, then the
* default codepages from DFLT_ECP and DFLT_ACP are used.
* ********************************************************************
*
* Address of service table (see below)
*
PSERVTAB DC     A(SERVTAB)
```

```
*
OPT_END  EQU    *     End of fixed part
*
* ********************************************************************
* Macro definition used for service table below.
* ********************************************************************
         MACRO
&SERV    SERVICE &NAME=,&ASCIICP=,&EBCDICCP=
&SERV    DC    Y(L&SYSNDX-*)    LENGTH OF THIS ENTRY
         DC    CL8&NAME         NAME OF TCPIPSERVICE
         DC    CL16&EBCDICCP    EBCDIC CODEPAGE (ICONV FORMAT)
         DC    CL16&ASCIICP     ASCII CODEPAGE (ICONV FORMAT)
L&SYSNDX DC    0H               START OF NEXT ENTRY
         MEND
*
* ********************************************************************
* Start of the service table. Entries are specified as follows:
*      SERVICE NAME='service',ASCIICP='asciicp',EBCDICCP='ebcdiccp'
* ********************************************************************
*
SERVTAB  DS     0F              START OF TABLE
*
          SERVICE NAME='CWS-US',                                      X
               ASCIICP='UTF-8',                                       X
               EBCDICCP='IBM-1047'  US ENGLISH
          SERVICE NAME='CWS-DE',                                      X
               ASCIICP='UTF-8',                                       X
               EBCDICCP='IBM-1141'  GERMAN
*
SERVTEND DC     H'0'            END OF TABLE
*
         END
/*
// IF $MRC GT 4 THEN
// GOTO NOLINK
// EXEC LNKEDT,PARM='MSHP'
* ********************************************************************
* DONT FORGET TO NEWCOPY THE IESRESTO IN ORDER TO ACTIVATE IT:       *
*   CEMT SET PROG(IESRESTO) NEWCOPY                                  *
* ********************************************************************
/. NOLINK
/*
/&
* $$ EOJ
```

第 26 章 z/VSE MQ Client Trigger Monitor を非同期プログラム間通信に使用

このトピックでは、WebSphere MQ Server for z/VSE V3.0.0 (5655-U97) を使用しているアプリケーション環境を、WebSphere MQ Client for VSE にマイグレーションする方法について説明します。

IBM 製品「WebSphere MQ for z/VSE V3.0」(5655-U97) は 2014 年 9 月 8 日に市場から撤退し、2015 年 9 月 30 日に実際上のサービスを停止しました。WebSphere MQ for z/VSE V3.0 は、キューイング、トリガー、チャネルのパブリッシュおよびサブスクライブなど、広範囲に及ぶ代表的な MQ 機能をサポートする MQ Server を実装していました。

そのため、WebSphere MQ for z/VSE V3.0 のユーザーには、代わりに VSE 上で MQ Client を使用するという代替オプションが提示されました。このオプションは、http://www.ibm.com/support/docview.wss?rs=171&uid=swg24010051&loc=en_US&cs=utf-8&lang=en からダウンロードする SupportPac MQC5 として入手可能です。

このトピックには以下が含まれます。

- 『z/VSE MQ Client Trigger Monitor の概要』
  代表的なアプリケーション環境 WebSphere MQ Server for z/VSE V3.0.0 (5655-U97) と、代わりに MQ Client を使用した場合の環境の様子について説明する概説。
- 537 ページの『MQ Client for z/VSE に対するアプリケーション変更』
  WebSphere MQ Client for z/VSE のセットアップと、アプリケーションに必要な変更点について説明するトピック。
- 542 ページの『z/VSE MQ Client Trigger Monitor の使用』
  MQ Client アプリケーションでのトリガーの使用を可能にする z/VSE 機能 MQ Client Trigger Monitor について説明するトピック。

z/VSE MQ Client Trigger Monitor の概要

このトピックでは、代表的なアプリケーション環境 WebSphere MQ Server for z/VSE V3.0.0 (5655-U97) について説明します。

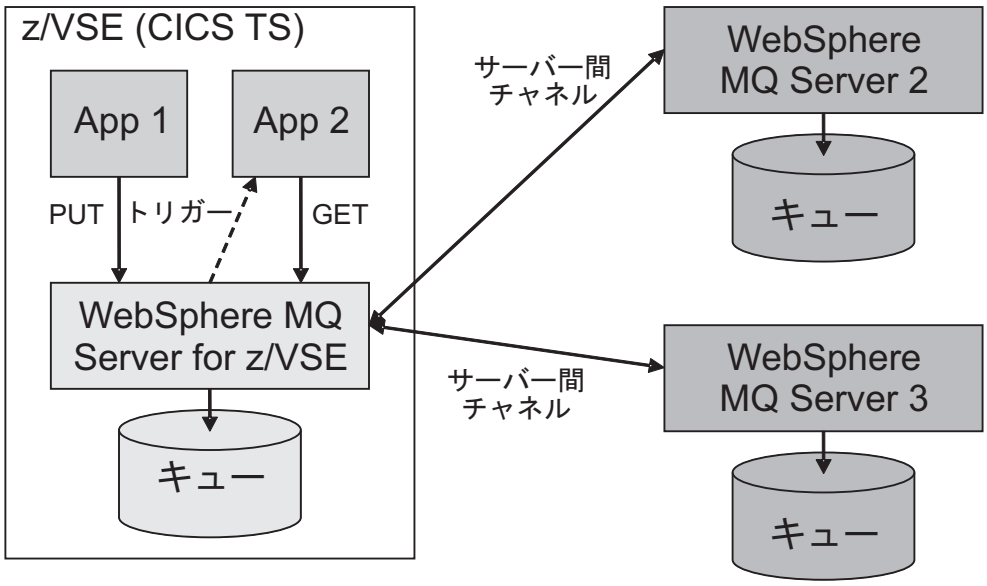


図 *183. アプリケーション環境 WebSphere MQ Server for z/VSE V3.0.0*

WebSphere MQ Server for z/VSE は CICS 領域の内部で稼働します。そのため、CICS またはバッチ・アプライアンスは MQ プログラミング・インターフェース (API) を使用して、キュー・マネージャーに接続し、メッセージ・キューを開き、メッセージをキューに書き込み、メッセージをキューから取得することができます。

特定キューへのメッセージ到着時にアプリケーションを起動 (トリガー) することができます。このことは、キュー (トリガー構成済み) へのメッセージ到着時に、MQ Server は、いわゆるトリガー・プログラムを開始する (呼び出す) ことを意味します。トリガー・プログラム (上の例では App2) は次に、トリガー対象のキューを開き、メッセージをキューから取得し、処理します。

ある MQ Server は、TCP/IP ネットワーク経由でサーバー間チャネルによって他の MQ Server との相互接続が可能です。これにより、MQ Server はメッセージを (送信) キューからもう 1 つの MQ Server 上のキューに渡すことができます。サーバー間チャネルは一般的に、1 つの、または一定数のメッセージが送信キューに入れられるとアクティブになります。そして、すべてのメッセージが (複数回に分けて) 送信され、受信キューに正常に格納された後で、送信キューから削除されます。

アプリケーション・シナリオの例

環境の理解を深めるために、アプリケーション・シナリオの例を詳しく見てみましょう。このシナリオは図 183 を基にしています。

- App 1 は VSE 上の MQ Server のローカル・キューにメッセージを書き込みます。
- このローカル・キューは送信キューとして構成されているため、MQ Server 2 (例えば分散システムとして実行) に対するサーバー間チャネルが開始され、メッセージは MQ Server 2 上のキューに転送されます。

- そのキューにはトリガーが定義されています。これにより、MQ Server 2 上でアプリケーションが開始されます。
- このトリガー・アプリケーションは、キューからメッセージを取得し、処理します。
- 応答メッセージが MQ Server 2 上の別のキューに書き込まれます。
- そのキューも送信キューとして構成されているため、z/VSE 上の MQ Server に対するサーバー間チャネルが開始されます。その結果、応答メッセージが VSE 上の MQ Server のキューに転送し戻されます。
- そのキューにはトリガーが定義されており、z/VSE 上で App 2 が開始されます。
- App 2 は応答メッセージをキューから取得し、処理します。

シナリオは、MQ Server 3 も関与したり、メッセージを変換して他のシステムに渡したりするなど、さらに複雑な場合があります。

MQ Client の使用

z/VSE 上で MQ Server の代わりに MQ Client を使用する場合は、環境を少し変更する必要があります。

MQ Client では現在、z/VSE 上にローカル側のキューは存在しません。代わりにアプリケーションは、別のシステムにある別の MQ Server 上のキューと (MQ Client を通じて) リモートに連携します。

環境を MQ Client を使用する環境にマイグレーションするために、2 つのオプションがあります。これらのオプションを 536 ページの図 184 と 536 ページの図 185 に示します。

オプション **1:**
z/VSE 上の **MQ Server** を **VSE** 上の **MQ Client** に置き換え、**VSE** アプリケーションを **MQ Server 2** および **3** と直接連携させます。

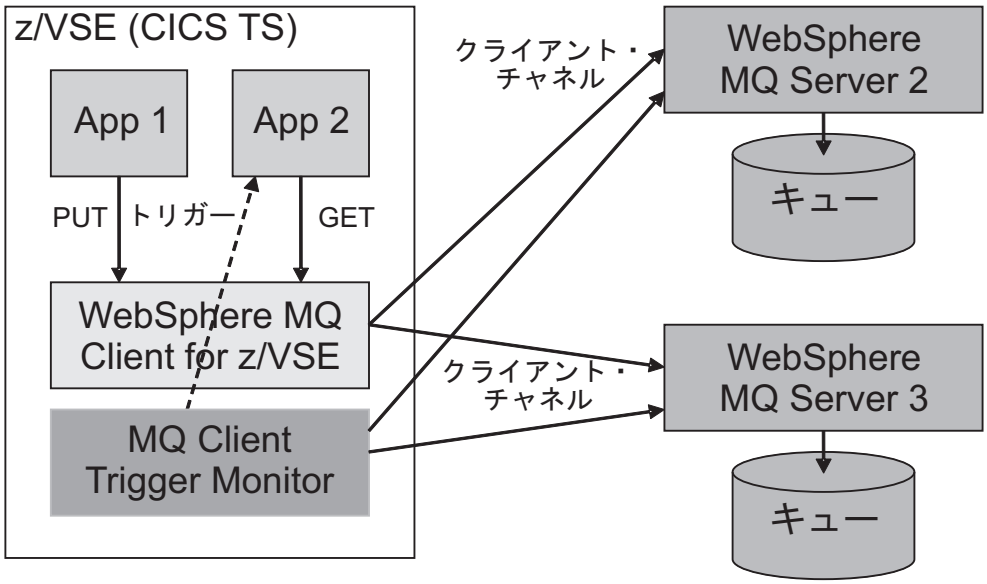


図 *184. VSE* 上で *MQ Client* を使用する (専用 *WebSphere MQ Server* なし)

オプション **2:**
z/VSE 上のアプリケーションが **MQ Client** を通じて連携する、専用 **MQ Server** を追加します。この専用 **MQ Server** は、環境内の他の **MQ Server** へのサーバー間チャネルを持ちます。

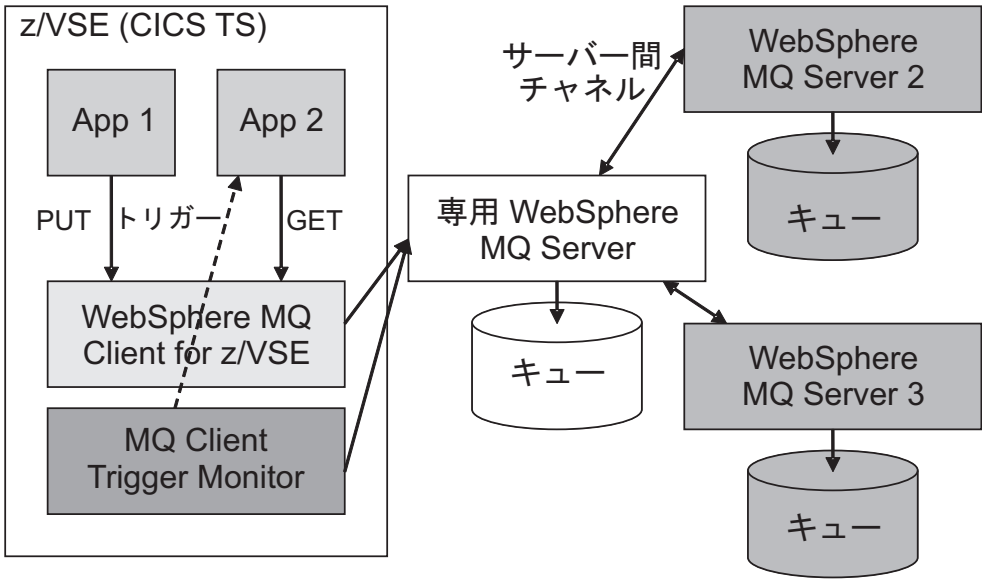


図 *185. VSE* 上で *MQ Client* を使用する (専用 *WebSphere MQ Server* あり)

オプション **1** (図 184 参照) を使用すると、よりシンプルな環境になります。 MQ キューにアクセスする z/VSE アプリケーションには、(特に複数の MQ Server を使用する場合は) さらに変更が必要になることがあります。ここでは、z/VSE アプ

リケーションは、連携する MQ Server を認識/選択することが必要になる場合があります。また、各アプリケーションは、MQ Server 2 および 3 のキューにアクセスするために、異なるキュー名を使用することが必要な場合があります。さらに、クライアント・アプリケーションが z/VSE に対するサーバー間チャネルを接続または削除できるように、MQ Server 2 および 3 で構成変更が必要です。

オプション **2** (536 ページの図 185 参照) は、z/VSE アプリケーションが常に同じ MQ Server と連携できるように、専用 MQ Server (複数の z/VSE システムで共用する場合あり) を追加します。ここでは、MQ Server 2 および 3 の構成はほぼ同じです (チャネルで異なる IP アドレスまたはホスト名が必要になる場合があります)。専用サーバーは、z/VSE 上の MQ Server が使用していたものと同じキューを使用できます。そのため、VSE アプリケーションに対する変更は最小限で済みます。

環境、アプリケーション、およびユース・ケースの複雑さに応じて、オプションを選択 (または両方を混用) してください。

MQ Client for z/VSE に対するアプリケーション変更

このトピックでは、MQ Server の代わりに MQ Client を使用するように既存のアプリケーションをマイグレーションする際に必要な手順について説明します。

WebSphere MQ Client for z/VSE のインストール

WebSphere MQ Client SupportPac は、http://www.ibm.com/support/docview.wss?rs=171&uid=swg24010051&loc=en_US&cs=utf-8&lang=en からダウンロードできます。

カテゴリー 3 の WebSphere MQ SupportPacs は、IBM プログラムのご使用条件 (IPLA) で指定された標準の使用条件のもとに配布されるため、WebSphere MQ のお客様にプログラム問題サービスを提供します。 SupportPac に付属の IPLA ファイルおよび LI ファイルを読み、SupportPac が提供される条件について確実に理解しておいてください。

SupportPac の問題と思われる事象が発生し、(非 z/VSE プラットフォーム上の) WebSphere MQ Server 製品の現行ライセンスを所有している場合は、SupportPac を使用している WebSphere MQ Server 製品に用いるものと同じ問題報告チャネルで問題を報告することにより、プログラム・サービスを要求できます。この SupportPac を使用するためにライセンス証書は不要です。

WebSphere MQ for VSE V3.0 のサポート終了の告知に続き、同じ時間フレーム内でこの SupportPac のサポートを中止する予定は当面ありません。

インストール手順

ファイル **mqc5.zip** を一時ディレクトリーにダウンロードし、解凍します。これにより、必要なライブラリー・ファイルと readme テキスト・ファイルが作成されます。ファイルを z/VSE に転送し、ご使用の環境用に SupportPac をセットアップする方法について詳しくは、readme ファイルの「インストール」セクションにある手順に従ってください。

MQC5.ZIP を解凍すると、以下のファイルが作成または置換されます。

MQICLIB.JCL
MQ Client のオブジェクトおよびフェーズをカタログします。

MQICINC.JCL
コピーブックとインクルード・ファイルをカタログします。

MQICSMP.JCL
サンプル・クライアント・アプリケーション・ソースをカタログします。

MQICBRG.JCL
MQ Client for VSEBridge のコンポーネントをカタログします。

README.TXT
このファイルをよく読んでください。

LICENSES¥*.TXT

.JCL ファイルを z/VSE リーダー (または z/VM) に BINARY および LRECL 80 でアップロードします。これらのファイルは、PHASE、OBJ、インクルード・ファイル、コピーブック、およびサンプルを、選択したライブラリーにカタログします。ジョブには、SETPARM ステートメントによる宛先サブライブラリーの指定を可能にする、PAUSE ステートメントが含まれています。デフォルト・サブライブラリーは PRD2.MQICVSE です。 MQICLIB.JCL を最初に実行する必要があります。宛先サブライブラリーがまだ存在していない場合に作成するためです。

注**:** MQ Client を、MQ Server がインストールされている同じサブライブラリーにインストールしないでください。

LIBDEF に関する考慮事項

MQ Client インストール・サブライブラリーを、アプリケーションと CICS の PHASE LIBDEF チェーンに追加する必要があります。 MQ Client コードは、MQ 呼び出しの実行中に MQ Client 固有のフェーズをロードします。これらの MQ Client フェーズが使用できない場合、MQ Client 機能を呼び出すとアプリケーションは異常終了します。

MQ Client アプリケーションのコンパイル時には、コンパイル・ジョブの SOURCE LIBDEF に MQ Client インストール・サブライブラリーが含まれている必要があります。 MQ Client サブライブラリーのみが含まれるようにし、MQ Server サブライブラリーは含まれないようにします。これにより、MQ Client のインクルード・ファイルとコピーブックが確実に使用されます。 MQ Client のインクルード・ファイルとコピーブック内の宣言は、MQ Server のそれらと互換性がなければなりません。 MQ Client のインクルード・ファイルとコピーブックには、MQ Client にのみ必要な追加の定義 (MQCONNX 呼び出しの構造など) がいくつか含まれています。

CICS 定義

プログラムの自動インストールをアクティブにしていない場合は、MQ Client プログラム「MQICVSEP」、「AMQERROR」、および「UAMQMSET」を CICS に定義する必要があります。

```
DEFINE PROGRAM(MQICVSEP) GROUP(MQM) LANGUAGE(C)
       EXECKEY(CICS) DATALOCATION(ANY)
DEFINE PROGRAM(AMQERROR) GROUP(MQM) LANGUAGE(ASM)
       EXECKEY(USER) DATALOCATION(ANY)
DEFINE PROGRAM(UAMQMSET) GROUP(MQM) LANGUAGE(ASM)
       EXECKEY(USER) DATALOCATION(ANY)
```

MQ Client を使用する CICS プログラムもすべて、**EXECKEY(CICS)** を指定して CICS に定義する必要があります。 MQ Client は分岐リンク・インターフェースを使用して呼び出されますが、CICS はキー切り替えを実行できないためです。したがって、呼び出し側プログラムは事前に CICS キーで実行されていなければなりません。

MQ Client を VSEBridge に使用する予定の場合は、メンバー MQCICSD.Z によって CICS CSD ファイル更新を実行することも必要です。

MQ Client 固有のエラーを処理するためのアプリケーション変更

MQ Client アプリケーションはリモート MQ Server を使用するため、追加のエラー状態が発生する可能性があります。各 MQ Client MQI 呼び出しは、TCP/IP を介した MQ Client チャネル経由でのネットワーク通信を必要とします。ネットワーク接続は、さまざまな理由 (ファイアウォールのアイドル・タイムアウトなど) で切断または中断されることがあります。

MQ Client アプリケーションは、そのようなエラー状態に対応できなければなりません。 MQI 呼び出しからの追加エラー・コードがある場合があります。また、より複雑なリカバリー・コードをアプリケーションに実装することが必要になる場合もあります。

データ変換 (ASCII/EBCDIC) に関するアプリケーション変更

z/VSE アプリケーションが z/VSE 上の MQ Server と連携する場合、MQ メッセージに含まれるデータは、通常はホスト・フォーマット (テキスト・データの場合は EBCDIC など) です。メッセージを z/VSE 上の MQ Server から z/VSE 以外の MQ Server に渡す場合、メッセージは、サーバー間チャネル経由での送信時に自動変換することができます。

MQ Client を使用している場合、アプリケーションは、z/VSE の外部 (多くの場合は Linux、Windows、Unix などの ASCII ベースのプラットフォーム) にある MQ Server と直接連携します。そのため、MQ Client から他のシステムに渡されるメッセージのデータ・フォーマットには注意する必要があります。メッセージは自動的に変換されませんが、MQ Client アプリケーションは、オプション MQGMO-CONVERT を使用して、MQGET 呼び出しによってメッセージの変換を要求できます。この機能は、既知のメッセージ・フォーマットに関するメッセージ変換を行います。

MQ Client アプリケーションで使用する文字セットを指定するには、環境変数 MQCCSID を使用します。環境変数の設定は、LE/C を使用している場合は setenv() 呼び出しによって行うことができます。 COBOL および PL/I のアプリケーションでは、MQ Client for VSE で提供されている MQSETENV API 呼び出し

を使用する必要があります。環境変数はまた、LE ランタイム・デフォルト・オプションでも設定可能です (CEEUOPT および CEECOPT を参照)。

注: MQ Client を使用している場合は、コード・ページ変換 (ASCII/EBCDIC) が MQ Server によって実行されます。

MQ Client ライブラリーに対するアプリケーションの再リンク

MQ Client ライブラリー (MQICVSE.OBJ) に対してアプリケーションを再リンクする必要があります。

MQ Server ライブラリーに対してアプリケーションをリンクすると、各 MQI 呼び出しによって適切なオブジェクト・デックが自動的に組み込まれます。例えば、MQGET 呼び出しは MQGET.OBJ をオートリンクによって組み込みます。

MQ Client を使用する場合は、MQ Client オブジェクト・デック MQICVSE.OBJ を明示的にアプリケーションに組み込む必要があります。 MQICVSE には、すべての MQI 呼び出し (MQGET など) の項目が含まれています。 MQ Client サブライブラリーがコンパイルおよびリンク・ジョブの LIBDEF にあることを確認してください。

さらに、MQ Client を使用するアプリケーションは、LE プリリンカーを使用してプリリンクする必要があります。プリリンクは通常、LE/C プログラミング言語で書かれたアプリケーションにのみ必要です。ただし、MQ Client を使用している場合は、アプリケーションが COBOL または PL/I に実装されていても、すべてのアプリケーションをプリリンクする必要があります。

```
// OPTION CATAL
   PHASE YOURPROG,*
   INCLUDE YOURPROG
   INCLUDE MQICVSE
/*
// EXEC EDCPRLK,SIZE=EDCPRLK,PARM='UPCASE MAP'
/*
// EXEC LNKEDT,SIZE=256K
/*
```

MQ Client でサポートされない MQ Server 機能

WebSphere MQ Server for z/VSE V3.0.0 (5655-U97) でサポートされている以下の機能は、MQ Client for z/VSE ではサポートされていないか、使用できません。

パブリッシュおよびサブスクライブ **(Pub/Sub)** はサポートされていません

WebSphere MQ Server for VSE では、APAR PM85374 および PM86869 以降、パブリッシュとサブスクライブがサポートされています。これらによって、以下の MQI 呼び出しが MQ Server on z/VSE に追加されました。

- MQSUB レジスター・サブスクリプション。
- MQSUBRQ サブスクリプション要求。

これらの MQI 呼び出しは、MQ Client on z/VSE では使用できません。

MQSUB 呼び出しと MQSUBRQ 呼び出しがないため、MQ Client では z/VSE アプリケーションからパブリッシュ機能とサブスクライブ機能を使用できません。これらの機能を使用する既存の VSE MQ Server アプリケーションは、代わりに MQ Client が使用されると失敗し、おそらく再設計が必要になります。

メッセージ・プロパティー **API** 呼び出しはサポートされていません

WebSphere MQ Server for VSE では、APAR PM85374 および PM86869 以降、メッセージ・プロパティー API 呼び出しがサポートされています。これらによって、以下の MQI 呼び出しが MQ Server on z/VSE に追加されました。

- MQBUFMH: バッファーをメッセージ・ハンドルに変換します。
- MQCRTMH: メッセージ・ハンドルを作成します。
- MQDLTMH: メッセージ・ハンドルを削除します。
- MQDLTMP: メッセージ・プロパティーを削除します。
- MQINQMP: メッセージ・プロパティーを照会します。
- MQMHBUF: メッセージ・ハンドルをバッファーに変換します。
- MQSETMP: メッセージ・プロパティーを設定します。

これらの MQI 呼び出しは、MQ Client on z/VSE では使用できません。

メッセージ・プロパティー呼び出しがないため、MQ Client では z/VSE アプリケーションからメッセージ・プロパティー機能を使用できません。これらの機能を使用する既存の VSE MQ Server アプリケーションは、代わりに MQ Client が使用されると失敗し、おそらく再設計が必要になります。

WebSphere MQ Client for VSE では **SSL** 有効化チャネルはサポートされていません

MQ Server for VSE では SSL 有効化チャネルがサポートされていますが、Client では SSL はサポートされていません。そのため、データはすべて、サーバー・チャネルを通じて MQ Server へ「平文で」転送されます。

VSE の一部のお客様は、VSE システムから送信されるすべてのデータを暗号化する要件をお持ちです。この要件は、VSE で MQ Client を使用している場合は実現できません。 VSE アプリケーションが、同じボックス上の Linux on System z で稼働している MQ Server を使用している場合でも、平文のまま送信されるデータは漏えいの危険があります。

DB2 Q Capture には **z/VSE** 上の **MQ Server** 環境が必要です

VSE および VM 上の Db2 DataPropagator Q Capture Supplement for Db2 は、Q Capture プログラムを実行するために、z/VSE 上の MQ Server 環境を必要とします。

そのため、Db2 DataPropagator Q Capture 機能は、MQ Client のみの環境では使用できません。

MQ CICS Bridge は **MQ Client** では使用できません

WebSphere MQ for VSE Server 機能である MQ CICS Bridge は、MQ Client では使用できません。

そのため、MQ CICS Bridge 機能は、MQ Client のみの環境では使用できません。

MQ Client では各種 **MQ** 構造の上位バージョンはサポートされていません

MQ プログラミング・インターフェースで使用される構造の多くには、バージョン・フィールドが含まれています。これにより、追加フィールドを構造の末尾に追加できます。 MQ Client では各種 MQ 構造の上位バージョン番号はサポートされていないため、新しいバージョンの構造に関連する各種機能はサポートされません。大半の構造では、MQCD (バージョン 3 までサポート) を除き、バージョン 1 (MQOD、MQPMO、MQMD、MQGMO、MQCFH) のみがサポートされます。

z/VSE MQ Client Trigger Monitor の使用

非同期メッセージ・パッシングの使用時には、トリガーは必須機能です。 1 つ以上のメッセージがキューに到着したときに、この機能によってアプリケーションを起動 (トリガー) することができます。キューごとに、トリガーするトリガー・プログラムを定義できます。呼び出される (トリガーされる) と、トリガー・プログラムは、呼び出される原因となったトリガー・イベントに関する情報を取得します。ほかにも、トリガー・イベントには、トリガーを起こしたキューに関する情報が含まれています。トリガーされたプログラムは次に、MQGET 操作を実行し、キューからメッセージを取得して処理します。

Linux/Unix/Windows や z/OS でも使用可能な MQ Client パッケージとは異なり、z/VSE MQ Client パッケージには、メッセージのキュー到着時にアプリケーションをトリガーできるクライアント・トリガー・モニターは含まれていません。

配布される MQ Client インストールでは、MQ Client Trigger Monitor プログラム RUNMQTMC を実行して、このプログラムを MQ Server に接続し、開始キューをモニターします。 z/OS では、トリガー・モニター・トランザクション CKTI を使用できます。

RUNMQTMC も CKTI も、z/VSE MQ Client では使用できません。

z/VSE MQ Client Trigger Monitor

z/VSE V6.1 以降、APAR PIPI42612 / PTF UI28408 (z/VSE 5.1)、または APAR PI42615 / PTFUI28409 (z/VSE 5.2) で提供される z/VSE MQ Client Trigger Monitor 機能は、z/VSE 上の MQ Client で使用するトリガー・モニターを実装します。 z/VSE 上の MQ Server と同様に、z/VSE MQ Client Trigger Monitor は、CICS 下で実行され、メッセージがキューに到着したときに CICS アプリケーションをトリガーできます。

z/VSE MQ Client Trigger Monitor で実行されるトリガーは、WebSphere MQ for z/VSE V3.0 と同じ方法で動作します。そのため、トリガーされるアプリケーションに対しては、z/VSE MQ Client Trigger Monitor を使用するための変更は不要です。ただし、トリガーされるアプリケーションは、 537 ページの『MQ Client

for z/VSE に対するアプリケーション変更』のトピックで説明されているように、MQ Server の代わりに MQ Client を使用するようにマイグレーションされていなければなりません。

z/VSE MQ Client Trigger Monitor は、他のプラットフォーム上のトリガー・モニターと同じ方法で動作します。したがって、トリガーのセットアップと構成は、他のトリガー・モニターの場合と同じです。

MQ Server でのトリガーの動作の仕方

トリガーがどのように動作するかについての良い説明が「Triggering for beginners」というブログ (https://www.ibm.com/developerworks/community/blogs/aimsupport/entry/triggering_for_beginners?lang=en) にあります。

以下のセクションでは、トリガーを有効化および制御するための構成設定について説明しています。

トリガーの構成**:**

トリガーは、MQ Server 上のローカル・キュー用に構成できます。トリガーを有効にするには、キューでの以下のトリガー設定を構成します。

- トリガー・タイプ**:** 以下のいずれかを指定できます。
  - **FIRST:** トリガー対象キューの現在の深さが 0 から 1 に変わると、トリガー・イベントが発生します。このタイプのトリガーは、サービス提供プログラムがキュー上のすべてのメッセージを処理する場合に使用します。
  - **EVERY:** トリガー対象キューにメッセージが到着するたびに、トリガー・イベントが発生します。このタイプのトリガーは、サービス提供プログラムが一度に 1 つのメッセージのみを処理する場合に使用します。
  - **DEPTH:** トリガー対象キュー上のメッセージ数が「トリガーの深さ」属性の値に達すると、トリガー・イベントが発生します。このタイプのトリガーは、サービス提供プログラムが固定数のメッセージを処理するように設計されている場合に使用します。
- トリガーの深さ**:** トリガー・タイプ DEPTH に使用、上記参照。
- トリガー・データ**:** トリガー・イベントによってトリガー対象アプリケーションに渡されるデータ。 VSE 上の MQ Server および z/VSE MQ Client Trigger Monitor の場合、これは以下を含む 13 文字のフィールドです。
  - 4 文字の CICS トランザクション ID、あるいは 4 つのブランクまたは下線。
  - 8 文字の CICS プログラム名、あるいは 8 つのブランクまたは下線。
  - 1 文字のトリガー・イベント・フラグ。
    - 「F」: トリガータイプは FIRST です。
    - 「E」: トリガー・タイプは EVERY です。
    - アプリケーションは、トリガー・モニターを停止するためにこれを「S」に設定できます。

- 開始キュー: 開始キューの名前。トリガー・イベントの基準が満たされると、キュー・マネージャーが開始キューにトリガー・メッセージを書き込みます。専用のローカル開始キューを作成するか、SYSTEM.DEFAULT.INITIATION.QUEUE を使用してください。
- プロセス名: WebSphere MQ プロセス定義の名前。 z/VSE MQ Client Trigger Monitor の場合、このフィールドは使用されませんが、MQ Server は有効なプロセス名の指定を必要とします。指定しないと、トリガーはアクティブになりません。以下の属性を使用してダミー・プロセスを (例えば CICS.PROGRAM のような名前で) 定義できます。
  - プロセス名: CICS.PROGRAM。
  - アプリケーション・タイプ: CICS VSE (ここでの選択は実際には無関係です)。
  - アプリケーション **ID:** CICSPROG (ここでの選択は実際には無関係です)。
  - 環境データ: トリガー・イベントによってトリガー対象プログラムに渡されるフリー・フォームのテキスト・フィールド。
  - ユーザー・データ: トリガー・イベントによってトリガー対象プログラムに渡されるフリー・フォームのテキスト・フィールド。

トリガー処理:

トリガーが有効になっているキューにメッセージが到着すると、以下の処理が MQ Server によって実行されます。

- トリガー基準が (トリガー・タイプに応じて) 満たされると、MQ Server はトリガー・イベント・メッセージを作成し、開始キューに書き込みます。
  - トリガー・イベント・メッセージには、トリガー・イベントの作成対象キューに関する情報、プロセス名 (CICS.PROGRAM など)、トリガー・データ・フィールド (VSE 上の MQ Server と z/VSE MQ Client Trigger Monitor で使用される 13 文字)、関連するプロセス定義からの情報 (アプリケーションのタイプと ID、環境データとユーザー・データなど) が含まれています。
  - 詳細については、http://www-01.ibm.com/software/integration/wmq/library/ で構造 MQTM を参照してください。

トリガー・モニター:

- トリガー・モニター (通常は別個のプロセスとして開始) は、開始キューをモニターし、開始キューからトリガー・イベント・メッセージを受け取ります。
  - トリガー・モニターは、MQ Server (サーバー・トリガー・モニター) と同じシステムで実行される場合と、MQ Client (クライアント・トリガー・モニター) を使用して MQ Server に接続し、異なるシステムで実行される場合があることに注意してください。
- トリガー・イベント・メッセージの情報に基づいて、トリガー・モニターはトリガー・プログラム (またはプロセス) を開始します。
  - z/VSE では、トリガー・プログラムは、トリガー・データ・フィールド (上記参照) での指定に基づいて、トリガー・トランザクションの EXEC CICS LINK または EXEC CICS START によって開始されます。トリガーされた

プログラムは、COMMAREA (LINK の場合) または EXEC CICS RETRIEVE (START の場合) によって、完全なトリガー・イベント・メッセージ (MQTM 構造) を取得できます。

- トリガーされたプログラムは、キュー・マネージャーに (例えば MQ Client 経由で) 接続し、トリガーを起こしたキュー (MQTM 構造内のフィールド QName で指定) を開き、MQGET 操作を実行して、トリガーを開始したアプリケーション・メッセージを取得します。
  - トリガーされたプログラムは、MQTM 構造の他のフィールド (ユーザー・データ・フィールドや環境データ・フィールドなど) からの情報も使用できます。
- トリガーされたアプリケーションは一般的に、取得可能なメッセージがキュー内になくなるまで、MQGET を (短い待ち時間で) ループで実行します。これにより、トリガー・イベントが作成され、アプリケーションがメッセージの処理を開始した時間フレーム内でキューに到着したアプリケーション・メッセージも確実に処理されます。

z/VSE MQ Client Trigger Monitor のインストール

z/VSE MQ Client Trigger Monitor は、z/VSE 6.1 以降にプリインストールされているか、APAR PI42612 / PTF UI28408 (z/VSE 5.1) または APAR PI42615 / PTF UI28409 (z/VSE 5.2) の適用によってインストールされます。

z/VSE MQ Client Trigger Monitor は、IESMQCTM という 1 つの CICS プログラムのみで構成されています。

プログラム (IESMQCTM.PHASE) を PRD1.BASE にカタログするほかに、以下の CICS 定義を実行する必要があります。

- プログラム **IESMQCTM** を **CICS** に定義:
  - Program: **IESMQCTM**
  - Group: **VSESPG**
  - Language: **C**
  - ExecKey: **CICS**
  - DataLocation: **Any**
- トランザクション **MQTM** を **CICS** に定義:
  - Transaction: **MQTM**
  - Group: **VSESPG**
  - Program: **IESMQCTM**
  - TaskDataKey: **User**
  - TaskdataLoc: **Any**
  - Profile: **IESXACT1**

トランザクション MQTM をセキュリティー・マネージャーに定義することも必要です。 z/VSE 基本セキュリティー・マネージャー (BSM) の場合、以下のコマンドを BSTADMIN を使用して実行してください。

```
ADD TCICSTRN 'MQTM' UACC(NONE) DATA('IBM SUPPLIED')
PERMIT TCICSTRN 'MQTM' ID(GROUP01) ACCESS(READ)
PERFORM DATASPACE REFRESH
```

上記の定義を実行するジョブが、インストール・パッケージで提供されています (ftp://public.dhe.ibm.com/eserver/zseries/zos/vse/download/mqtmdefs.job)。

z/VSE MQ Client Trigger Monitor は、新しいメッセージをいくつか出します (接頭部 IESC60 が付いています)。これらの新しいメッセージのメッセージ説明を、提供されている OME 更新ジョブ (ftp://public.dhe.ibm.com/eserver/zseries/zos/vse/download/mqtmome.job) を使用して z/VSE オンライン・メッセージ説明 (OME) ファイルにロードすることができます。

CICS 定義と OME メッセージはいずれも、デフォルトですでに z/VSE 6.1 以降で提供されています。

z/VSE MQ Client Trigger Monitor の構成

z/VSE MQ Client Trigger Monitor をインストールしたら、それを構成する必要があります。

z/VSE MQ Client Trigger Monitor の複数のインスタンスを同時に開始することができます。 1 つのトリガー・モニター・インスタンスは、MQ Server 上の 1 つの開始キューをモニターします。同一の、または異なる MQ Server 上の異なる開始キューをモニターするには、z/VSE MQ Client Trigger Monitor の複数のインスタンス (モニターする開始キューごとに 1 つ) を開始する必要があります。

z/VSE MQ Client Trigger Monitor のユーザー・インターフェースが MQTM トランザクションによって提供されています。このインターフェースは、端末から、または z/VSE コンソールから (MSG F2,DATA=MQTM ... を指定) 使用できます。

MQTM トランザクションで使用される一般的な構文は次のとおりです。

```
MQTM command parameter1=val1,parameter2=val2,...
```

パラメーターは、コマンドによってはオプションの場合があります。コマンドと最初のパラメーターの間には少なくとも 1 つのブランクが必要です。複数のパラメーターはコンマで区切ります。コンマの前後にブランクを使用できますが、それらのブランクはパラメーター名にも値にも属しません。パラメーターは通常、「key=value」の形式です。

以下のセクションでは、使用可能なコマンドについて詳細に説明します。

MQTM トランザクションは通常、ユーザーが対話式に入力します。ただし、MQTM で特定のアクションを自動化するために、プログラムから MQTM トランザクションを使用することもできます。そのためには、EXEC CICS START を使用して、MQTM トランザクションを開始し、コマンド・ストリング (コマンド名とパラメーター) をデータとして渡すことができます。あるいは、プログラム IESMQCTM に対して EXEC CICS LINK を実行し、COMMAREA を使用してコマンド・ストリングを渡すこともできます。

トリガー・モニター・インスタンスを開始する:

トリガー・モニター・インスタンスを開始するには、次の START コマンドを使用します。

```
MQTM START MQSERVER=server-channel-name/TCP/ip-or-hostname
 [,QMGR=queue-manager-name]
  [,INITQ=initiation-queue-name]
  [,RECONTIME=n]
  [,RECONTRIES=n]
  [,WAITTIME=n]
```

MQSERVER パラメーターは必須です。接続する MQ Server を指定します。形式は 「server-channel-name/TCP/ip-or-hostname」 です (MQ Client アプリケーションで使用される MQSERVER 環境変数の場合に似ています)。

QMGR パラメーターはオプションです。MQ Server 上のキュー・マネージャーの名前を指定します。省略すると、MQ Server 上のデフォルト・キュー・マネージャーが使用されます。

INITQ パラメーターはオプションです。トリガーに使用する開始キューの名前を指定します。省略すると、システムのデフォルト開始キュー SYSTEM.DEFAULT.INITIATION.QUEUE が使用されます。

RECONTIME パラメーターはオプションです。MQ Server に対する既存の接続の中断後にトリガー・モニターが再接続を試行するまでの時間 (秒) を指定します。RECONTIME を省略すると、デフォルトの 30 秒が使用されます。 RECONTIME に 0 (ゼロ) を指定すると、トリガー・モニターは再接続を試行せず、接続が中断されると終了します。

RECONTRIES パラメーターはオプションです。MQ Server に対する既存の接続の中断後にトリガー・モニターが再接続を試行する回数を指定します。RECONTRIES を省略するか、RECONTRIES に 0 (ゼロ) を指定すると、無制限に試行を行います。

WAITTIME パラメーターはオプションです。トリガー・モニターが開始キューに対する MQGET 呼び出しで使用するタイムアウト (秒) を指定します。WAITTIME を省略すると、デフォルトの 60 秒が使用されます。 WAITTIME に 0 (ゼロ) を指定すると、MQGET は無制限に待機します。アイドリング接続はファイアウォールによって中断されることがあるため、MQGET でタイムアウトを使用すると、クライアント接続を開いたままにしておく場合に役立ちます。

トリガー・モニター・インスタンスを停止する:

アクティブなトリガー・モニター・インスタンスを停止するには、次の STOP コマンドを使用して、停止するインスタンスを特定するために MQ Server、キュー・マネージャー、および開始キューの名前を指定します。

```
MQTM STOP MQSERVER=server-channel-name/TCP/ip-or-hostname
  [,QMGR=queue-manager-name]
  [,INITQ=initiation-queue-name]
```

MQSERVER パラメーターは必須です。接続する MQ Server を指定します。形式は 「server-channel-name/TCP/ip-or-hostname」 です (MQ Client アプリケーションで使用される MQSERVER 環境変数の場合に似ています)。

QMGR パラメーターはオプションです。MQ Server 上のキュー・マネージャーの名前を指定します。省略すると、MQ Server 上のデフォルト・キュー・マネージャーが使用されます。

INITQ パラメーターはオプションです。トリガーに使用する開始キューと同じものを指定します。省略すると、システムのデフォルト開始キュー SYSTEM.DEFAULT.INITIATION.QUEUE が使用されます。

すべてのアクティブなトリガー・モニターのリストを表示する:

すべてのアクティブなトリガー・モニター・インスタンスのリストを表示するには、SHOW コマンドを使用します。コマンドをコンソールから発行すると (MSG F2,DATA=MQTM SHOW など)、リストがコンソールに出力されます。コマンドを端末から発行すると、3270 画面が端末に表示されます。 1 画面に収まらない数のインスタンスがアクティブである場合は、画面をスクロールできます。 ENTER キーを押すと表示を更新できます。

```
MQTM SHOW
```

すべての事前構成トリガー・モニター・インスタンスを開始する:

複数のトリガー・モニター・インスタンスがある環境では、通常はそれらを (MQTM START によって) 手動では開始しません。代わりに、開始するトリガー・モニターのリストを指定して、それらを一度に開始します。

STARTALL コマンドは、アクティブな LIBDEF SOURCE チェーン内で検索される、開始リスト・ライブラリー・メンバー IESMQCTL.Z を読み取ります。IESMQCTL.Z を PRD2.CONFIG に入れるのが良い方法です。

```
MQTM STARTALL
```

メンバー IESMQCTL.Z はプレーン・テキストであり、開始するトリガー・モニター・インスタンスの START コマンドを含んでいます。 START コマンドは「START」キーワードで始まり、MQSERVER、QMGR、INITQ、RECONTIME などのパラメーターが続きます。コマンドを次の行に続けるには、行の終わりに「-」文字を置きます。これは、コマンドが次の行に続いていることを示します。アスタリスク (*) で始まる行はコメント行として扱われ、無視されます。空の行 (すべてブランク) も無視されます。

例:

```
*
START MQSERVER=server-channel-name/TCP/ip-or-hostname -
[,QMGR=queue-manager-name] -
[,INITQ=initiation-queue-name] -
[,RECONTIME=n] -
[,RECONTRIES=n] -
[,WAITTIME=n]
*
```

すべてのアクティブなトリガー・モニター・インスタンスを停止する:

STOPALL コマンドは、現在アクティブなすべてのトリガー・モニター・インスタンスを停止します。

```
MQTM STOPALL
```

CICS 始動中にトリガー・モニターを自動開始する**:**

事前構成されたトリガー・モニター・インスタンスを CICS の始動時に開始したい場合は、プログラム IESMQCTM を CICS の DFHPLTPI リストに組み込むことができます。これにより、STARTALL コマンドと同じアクションが実行され、また開始リスト・ライブラリー・メンバー IESMQCTL.Z が読み取られます。

```
DFHPLT TYPE=ENTRY,PROGRAM=IESMQCTM
```

CICS シャットダウン中にトリガー・モニターを自動シャットダウンする**:**

CICS の終了時にアクティブなトリガー・モニター・インスタンスをすべてシャットダウンしたい場合は、プログラム IESMQCTM を CICS の DFHPLTSD リストに組み込むことができます。これにより、STOPALL コマンドと同じアクションが実行されます。

```
DFHPLT TYPE=ENTRY,PROGRAM=IESMQCTM
```

問題判別とトレース

トリガー・モニターが期待どおりに機能しない場合は、問題の原因を判別するために以下が役立ちます。

1. z/VSE MQ Client Trigger Monitor から出されるメッセージには、MQ の戻りコードと理由コードが含まれるものがあります。 MQ Client で提供されている MQ 資料とコピーブックを使用して、これらの戻りコードと理由コードの意味を判別してください。
2. z/VSE MQ Client Trigger Monitor のトレースをオンにすることができます。いくつかのトレース・メッセージが SYSLOG と SYSLST のいずれかまたは両方に出されます。トレースをオンにするには、TRACE=nnn パラメーターを MQTM コマンドに追加してください。 TRACE パラメーターには以下の値を指定できます。
   - TRACE=SYSLST
   - TRACE=SYSLOG
   - TRACE=BOTH
   - TRACE=OFF (デフォルト)

トリガー・モニター・インスタンスのアクティビティーをトレースするには、TRACE パラメーターが START コマンドまたは STARTALL コマンドに指定されている必要があります。すでにアクティブなインスタンスのトレースを動的にオンにすることはできません。インスタンスを停止して、この場合に指定する TRACE パラメーターを使用して再始動することが必要です。

第 27 章 非 Java アクセスのための VSE スクリプト・コネクターの使用

このトピックでは、*VSE* スクリプト・コネクター (これは、VSE スクリプト・クライアントと VSE スクリプト・サーバーから成る) を使用して、非 *Java* プラットフォームから z/VSE ホスト・データまたは物理/論理中間層データにアクセスする方法について説明します。

VSE スクリプト・コネクター を使用する主な利点は、非 Java プラットフォームからアクセスできることです。 しかし、VSE スクリプト・コネクターを使用して *Java* プラットフォーム からz/VSE ホスト・データまたは物理/論理中間層データにアクセスすることもできます。

ユーザーが作成するそれぞれの VSE スクリプト・クライアントごとに、特定の要件を満たすためには、大抵、(VSE スクリプト言語を使用して) 対応する VSE スクリプトを作成する必要があります。 しかし、新規 VSE スクリプト・クライアントは、既存の VSE スクリプトを使用して z/VSE ホストのデータ、または物理/論理中間層データにアクセスできます。

このトピックに含まれるのは次のとおりです。

- 『VSE スクリプト・コネクターの使用方法』
- 553 ページの『クライアントとサーバーとの間で使用されるプロトコルの概要』
- 554 ページの『VSE スクリプト言語を使用した VSE スクリプトの作成』
- 557 ページの『VSE スクリプト・クライアントを作成するために使用できるサンプル・ファイル』
- 557 ページの『VSE スクリプト・クライアント (およびその VSE スクリプト) を作成する例』
- 568 ページの『VSE スクリプト・クライアントを使用してデータを中間層から取得』

関連トピック:

- 45 ページの『VSE スクリプト・コネクター の概要』
- 45 ページの『第 7 章 VSE スクリプト・コネクターのインストール』

VSE スクリプト・コネクターの使用方法

VSE スクリプト・コネクターを使用して、以下からデータを取得できます。

- z/VSE ホスト.
- 物理/論理中間層

552 ページの図 186 には、VSE スクリプト・クライアントが VSE スクリプトを呼び出す方法を示します。

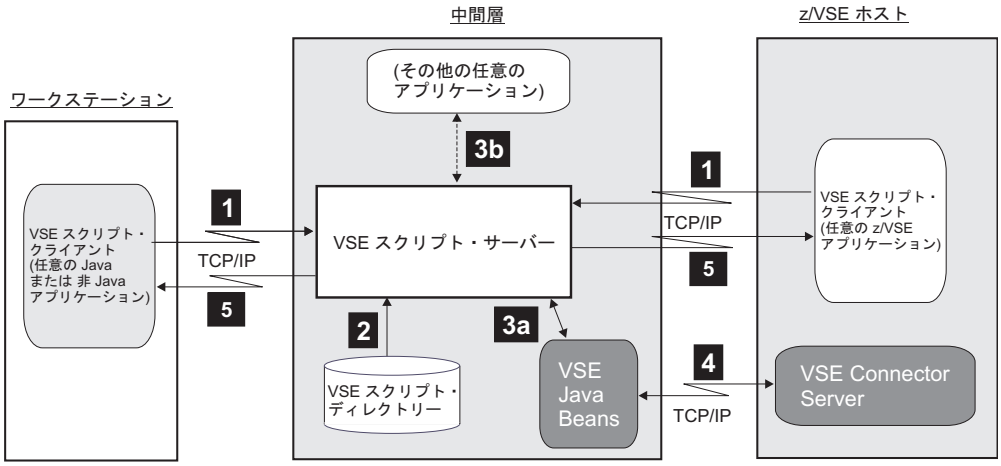


図 *186. VSE* スクリプト・コネクターを使用したホストまたは中間層からのデータの取得

1 VSE スクリプト・クライアントが、物理/論理中間層上の VSE スクリプト・サーバーへの TCP/IP 接続を確立します。 次に、VSE スクリプト・クライアントは、
- VSE スクリプトのファイル名
- VSE スクリプトに付随するパラメーター

を、物理/論理中間層および Java 使用可能プラットフォーム上で実行されている VSE スクリプト・サーバーに送信することによって、VSE スクリプトを呼び出します。

2 VSE スクリプト・サーバーは、VSE スクリプト・ディレクトリーにある VSE スクリプト・ファイルを読み取り、VSE スクリプト・ファイルのステートメントの解釈と変換を始めます。 各 VSE スクリプト・ファイルは、*VSE* スクリプト言語 を使用して作成されています (554 ページの『VSE スクリプト言語を使用した VSE スクリプトの作成』に説明があります)。

3a VSE スクリプト・ファイルに、z/VSE ホストからのデータが必要な場合、VSE Java Beans 要求は、VSE Java Beans によって実行されます。

3b VSE スクリプト・ファイルは、アプリケーションを呼び出して、物理/論理中間層で実行中の他のすべてのアプリケーションからデータを取得することもできます。 これは、VSE スクリプト・クライアントを z/VSE ホストで実行しようとしている場合、特に役立ちます。

4 VSE Java Beans は、 z/VSE ホスト (これは、関数とデータに対する要求を実行します) 上で実行されている VSE コネクター・サーバー と通信します。 (このステップは、ステップ **3a** に従う時にのみ適用されます)。

5 次に、取得されたデータは VSE スクリプト・クライアントが使用できるフォーマットに変換され、VSE スクリプト・クライアントに戻されます。

VSE スクリプト・コネクターを使用して、リモート・アプリケーション (REXEC など) を実行できます。 例えば、次のようになります。

- ステップ **3a** に従うと、VSE スクリプト・クライアントによって、z/VSE ホストでジョブを実行できます。
- ステップ **3b** に従うと、VSE バッチ・ジョブまたはプログラムによって物理/論理中間層のアプリケーションやシェル・スクリプトを実行できます。

クライアントとサーバーとの間で使用されるプロトコルの概要

このトピックでは、物理/論理中間層上にインストールされた VSE スクリプト・クライアントと VSE スクリプト・サーバーとの間のデータ・フローに使用されるプロトコルについて説明します。アクションの一般的なフローは、以下のとおりです。

1. VSE スクリプト・クライアントが、VSE スクリプト・サーバーへの接続をオープンします。
2. コード・ページを送信します (オプション)。
   a. 最初の行 (すなわち、スクリプト名が送信される前の行) (例: `**CODEPAGE=Cp1252**`) に、`CR LF (X'0D' X'0A')` が続きます。 コード・ページ行は常に UTF-8 で送信されます。
   b. 続く行は、指定したコード・ページ、または UTF-8 (コード・ページが指定されていない場合) のいずれでも送信されます。
3. 次に、VSE スクリプト・クライアントが、以下を VSE スクリプト・サーバーに送信します。
   a. VSE スクリプトの名前。この後に `CR LF (X'0D' X'0A')` が続きます。
   b. それぞれのパラメーター値。この後に `CR LF` が続きます。
4. 最後のパラメーターが送信された後に、空の行 (`CR LF` のみ) が送信されます。 これは、すべてのパラメーターが伝送されたことを示します。
5. VSE スクリプト・サーバーが、VSE スクリプトの実行を開始します。
6. VSE スクリプト・サーバーが、VSE スクリプトからの出力を、1 行ずつ VSE スクリプト・クライアントに戻します。 それぞれの行は、`CR LF` で終了します。
7. VSE スクリプト・サーバーは、最後の出力行を VSE スクリプト・クライアントに送信すると、接続をクローズします。

ここで説明したプロトコルは、どの種類のプログラム言語にも、簡単にインプリメントできます。 プログラム言語で必要なのは、TCP/IP ソケット関数への呼び出しをサポートすることだけです。

VSE スクリプト・コネクターは、Telnet アプリケーションを使用して簡単にテストできます。 このためには、以下を実行する必要があります。

1. Telnet クライアントと VSE スクリプト・サーバーとの間の接続を作成する。
2. ご使用の VSE スクリプトの名前を入力し、Enter を押して `CR LF` との回線を終了する。
3. 各パラメーターの名前を入力し、Enter を押して `CR LF` との回線を終了する。

   注**:** VSE スクリプト・サーバーは、受け取ったデータをエコー出力しないので、ユーザーが入力したものは表示されません。
4. 最後のパラメーターを入力した後で、Enter を押して、VSE スクリプトの実行を開始する。 これによって、VSE スクリプトからの出力が、Telnet クライアントによって表示されます。

以下の例は、パラメーターが 3 つ含まれている `test.src` という VSE スクリプトが実行されるシーケンスを示しています。 パラメーターは、`Hello`、`583`、および

get です。 VSE スクリプト・サーバーは、VSE スクリプトにある出力 (Script test.src has been executed) を VSE スクリプト・クライアントに送信して戻します。

1. VSE スクリプト・クライアントから VSE スクリプト・サーバーに:

```
test.src<CR><LF>
Hello<CR><LF>
583<CR><LF>
get<CR><LF>
<CR><LF>
```

2. VSE スクリプト・サーバーから VSE スクリプト・クライアントに:

```
Script test.src has been executed<CR><LF>
```

SSL 暗号化接続の使用

z/VSE 4.2 以降の VSE スクリプト・サーバーでは、VSE スクリプト・クライアントから VSE スクリプト・サーバーへの接続で *SSL/TLS* 暗号化接続 をサポートします。SSL/TLS を使用すると、プロトコルは同じままですが、暗号化された SSL 接続を通じて送信されます。SSL/TLS を使用するには、以下の作業を実行済みである必要があります。

- TCP/IP での SSL/TLS を有効にする。
- 必要な鍵および証明書を以下に作成する。
  - VSE スクリプト・サーバー・サイド
  - z/VSE.

VSE スクリプト言語を使用した VSE スクリプトの作成

VSE スクリプト・コネクターには、*VSE* スクリプト言語 という特殊なプログラム言語が組み込まれており、ユーザーは、これを使用して、独自の VSE スクリプトを作成できます。ご使用の VSE スクリプト・クライアントから VSE 関数およびデータにアクセスするには、これらの VSE スクリプトが必要です。

その他のプログラム言語の場合と同様に、VSE スクリプト言語は、以下のものから成っています。

- ステートメント (if、while、for、break、continue、sub、gosub、return)
- 演算子 (論理、連結、算術計算)
- 変数
- 組み込み関数

VSE スクリプト言語は、言語解説書 のオンライン・ドキュメンテーションに詳しい説明があります。これは、VSE スクリプト・サーバーのインストールの際に、ご使用のワークステーションにインストールされています (45 ページの『第 7 章 VSE スクリプト・コネクターのインストール』に説明があります)。

ここでは、以下のサブトピックで、VSE スクリプト言語の概要を説明します。

- 555 ページの『VSE スクリプト言語に適用される一般規則』
- 556 ページの『VSE スクリプト言語の組み込み関数』
- 556 ページの『VSE スクリプト言語トレース・サポート』

VSE スクリプト言語に適用される一般規則

ステートメント:

1. ステートメントは、以下のいずれかです。
   - キーワード (IF、WHILE、...)
   - 関数呼び出し
   - 変数宣言
   - 変数割り当て
2. 各ステートメントはセミコロン (;) で終了しなければならず、1 つの行には任意の数のステートメントを入れられます。
3. 1 つのコマンドは 2 行にまたがって分割できません (do ステートメントを持っている、1 つの行上のステートメントが 1 つのコマンドと見なされます)。
4. ブランク行は許容されます。
5. VSE スクリプト言語は、大文字と小文字を区別しません。
6. VSE スクリプトの解釈中に起こったリカバリー不能エラーは、*ScriptErrors* と呼ばれます。 ScriptError は、以下の情報を生成します。
   - エラーが起こった行番号とステートメント番号。
   - (固有の) エラー番号。
   - エラー番号に対する記述テキスト。

コメント:

1. コメントは、2 つのスラッシュ「//」で始まり、行の終わりで終わります。
2. コメントは、**コマンド**の後でも、あるいは、行の先頭でも入れられます。

変数:

1. 宣言する変数の数に制限はありません。
2. 各変数は、1 つの VSE スクリプトの中で 1 回だけ宣言できます。
3. 変数は、宣言されたあとでは、VSE スクリプトの中で、すべてグローバルになります。
4. 変数は、宣言しなければ使用できませんが、いつでも、どこでも定義できます。
5. 変数名は、英字で始まらなければなりません。 その英字の後には、すべての英字、数字、および、下線「_」が使用できます。
6. データ・タイプには、整数、ストリング、およびブールの 3 つのタイプがあります。
7. それぞれの変数は、値の配列としてアクセスできます。
8. `var[0]` は、変数 `var` そのものと同じで、したがって、`var[0]=1` は `var=1` と同じです。
9. 変数 ARGV は必ず VSE スクリプトの中で定義され、この名前が付いた変数は宣言することはできません。
10. 変数は STRING というタイプであり、変数の値は、スタートアップ時に VSE スクリプトに受け渡されるパラメーターです。

ライブラリアン・ファイル名:

1. ライブラリアン・ファイル名 は、「/」、「¥」、または「.」で区切られたライブラリー、サブライブラリー、メンバー、およびタイプが含まれるストリングです。
2. このフォーマットは以下のいずれかです。
   - *library.sublib¥member.type*
   - *library¥sublib¥member.type*
3. ライブラリアン関数では、以下の項目が処理対象として期待されます。
   - ライブラリー のみ。
   - ライブラリー およびサブライブラリー のみ。
   - 上記の (2.) にリストされているすべての項目。

VSE スクリプト言語の組み込み関数

VSE スクリプト言語の組み込み関数については、VSE Script Server の一部としてインストールされているオンライン HTML 文書に説明があります。

VSE スクリプト言語トレース・サポート

VSE スクリプト・トレース・サポートを使用すると、ご使用の VSE スクリプトのトレースおよびデバッグが可能です。

トレース域には INSTRUCTIONS、CONDITIONS、および PARAMETERS の 3 つがあります。

*** INSTRUCTION**
このトレースは、実行前の各ステートメントを示します。トレースの前に「*-*」が付きます。

*** CONDITION**
このトレースは、IF、WHEN、および FOR の各ステートメントでの条件の評価結果を示します。トレース出力の前に「===」が付きます。

*** PARAMETER**
このトレースは、関数呼び出しへのすべての入力パラメーターと、関数呼び出し後のすべての出力変数を示します。入力値の前には「>>>」が、出力変数の前には「<<<」が付きます。

これらの 3 つのトレース域は、すべて指定することも、1 つまたは 2 つのみ指定することもできます。どの場合も、複数の領域はコンマで区切る必要があります。

領域は Script Server プロパティー・ファイルでグローバルに有効化でき、以下のコマンドを使用して Script Server コンソール上で操作できます。

- traceon
- traceoff
- status

さらに、各スクリプトで新しいスクリプト組み込み関数

- traceOn()
- traceOff()

を使用して、コード自体の中でトレース域を動的に有効化または無効化できます。スクリプトがこれらの関数の 1 つを呼び出した場合、新しい設定が、スクリプトのランタイムのグローバル・トレース域設定を永続的にオーバーライドします。

VSE スクリプト・クライアントを作成するために使用できるサンプル・ファイル

表 10 にリストされているファイルは、VSE スクリプト・サーバーをインストールしたディレクトリーにコピーされます (詳しくは 48 ページの『ステップ 1.2: VSE スクリプト・サーバーのインストールを実行する』を参照)。

- VSEScriptServlet.java
- VSEScriptWebService.java

ユーザーは、表 10 にあるファイルを使用して、独自の VSE スクリプト・クライアントを作成できます。

表 10. VSE スクリプト・クライアントを作成するために提供されているファイル

ファイル	説明
VSEScriptClient.dll	例えば MS Office および Lotus 製品で使用する Windows DLL
VSEScriptClient.h	VSEScriptClient.dll で使用する C ヘッダー
VSEScriptClient.lib	VSEScriptClient.dll で使用する C ライブラリー・ファイル
VSEScriptClient.123	VSEScriptClient.dll を使用する Lotus 1-2-3 のサンプル
VSEScriptClient.lss	Visual Basic スクリプトの Lotus Script ソース
VSEScriptClient.xls	VSEScriptClient.dll を使用する Microsoft Excel のサンプル
VSEScriptClient.bas	Visual Basic スクリプトの Visual Basic ソース
VSEScriptCgi.c	VSEScriptClient.dll を使用する CGI のサンプル C ソース
VSEScriptCgi.exe	コンパイル済み CGI
com¥ibm¥vse¥script¥client¥VSEScriptServlet.java	サンプル・サーブレットの Java ソース
com¥ibm¥vse¥script¥client¥VSEScriptWebService.java	サンプル WebService の Java ソース

VSE スクリプト・クライアント (およびその VSE スクリプト) を作成する例

このトピックでは、VSE スクリプト・クライアントおよびその対応する VSE スクリプトを作成し、VSAM データを、Lotus 1-2-3 または Lotus Wordpro などのオフィス・アプリケーションに挿入する方法の例を提供します。

この例では、任意の Windows プログラム (例えば Lotus スプレッドシート) で使用可能な Windows DLL ファイル **VSEScriptClient.dll** を使用します。 ただし、この DLL ファイルは、 563 ページの図 190 に示した Lotus 1-2-3 のステートメ

ントのようなステートメントを使用して、宣言する必要があります。これらのステートメントもオンライン・ドキュメンテーションに入っていますので、必要に応じて、簡単にコピーと貼り付けができます。

このトピックには、以下のサブトピックがあります。
- 『ステップ 1: VSE スクリプト・サーバーのプロパティー・ファイルのセットアップ』
- 560 ページの『ステップ 2: 接続プロパティー・ファイルのセットアップ』
- 560 ページの『ステップ 3: サンプル VSAM データの定義』
- 560 ページの『ステップ 4: サンプル VSE スクリプトの変更』
- 562 ページの『ステップ 5: z/VSE ホストでの VSE コネクター・サーバー の開始』
- 562 ページの『ステップ 6: ローカルでの VSE スクリプト・サーバーの開始』
- 562 ページの『ステップ 7(a): サンプルの Lotus 1-2-3 スプレッドシート・ファイルのオープン』
- 565 ページの『ステップ 7(b): サンプル MS Office スプレッドシートのオープン』
- 568 ページの『ステップ 7(c): コマンド行からのサンプル VSE スクリプトの開始』

その他のクライアント・プラットフォーム用の VSE スクリプト・クライアントのプログラム作成方法の例は、VSE スクリプト・コネクターに付随のオンライン・ドキュメンテーションの中に提供されています。

ステップ 1: VSE スクリプト・サーバーのプロパティー・ファイルのセットアップ

VSE スクリプト・サーバーのパラメーターは、ファイル **VSEScriptServer.properties** に定義されており、このファイルは、VSE スクリプト・サーバーがインストールされているディレクトリーに入っています (詳しくは 48 ページの『ステップ 1.2: VSE スクリプト・サーバーのインストールを実行する』を参照)。 **VSEScriptServer.properties** の例を次に示します。

注: プロパティー値はすべて 1 行で指定しなければなりません。下の例では、本書のページ幅の都合上、いくつかの行は折り返されています。

```
#VSEScriptServer

#print messages (on) or do not print (off)
messages=on

#server mode: either 'vsescript' (default) or 'websocket'
#mode=websocket

#port where the server listens
listenport=4711

#Optional. Address to bind the listening socket to
#bindaddr=127.0.0.1

#number of maximum connections allowed
maxconnections=256

#root directory for scripts
scriptdirectory=./scripts
```

```
#name of the connection config file
connectionconfig=Connections.properties

#codepage for script clients.
#If not specified the current default codepage is used
#codepage=Cp1252

# Uncomment to log all script input parameters. (messages must be on)
#logscriptinputparams=true

# Uncomment to log all script output. (messages must be on)
#logscriptoutput=true

# Enable specified trace area(s) during server startup: INSTRUCTION,
PARAMETER or CONDITION
#traceon INSTRUCTION, PARAMETER, CONDITION

#SSL specific settings (uncomment to activate):

#sslversion can be either SSL or TLS
#sslversion=SSL

#If client authentication is true, the server requests the client to send
its certificate and verifies it.
#If client authentication is false or not specified, no client
authentication is done.
#clientauthentication=true

#keyringfile specifies the file name of the keyring file
#keyringfile=keyring.pfx

#keyringpwd specifies the password for opening the keyring file
#keyringpwd=ssltest

#ciphersuites specifies a comma separated list of cipher suites that are
accepted by the server.
# If this property is not specified, all supported cipher suites are used.
#ciphersuites=TLS_RSA_WITH_AES_256_CBC_SHA,TLS_RSA_WITH_AES_128_CBC_SHA
# For use with OpenSSL:
#ciphersuites=TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA,TLS_DHE_RSA_WITH_AES_256_CBC_SHA256,
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256,TLS_DHE_RSA_WITH_AES_256_CBC_SHA,
TLS_DHE_RSA_WITH_AES_128_CBC_SHA,SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA,
TLS_RSA_WITH_AES_256_CBC_SHA256,TLS_RSA_WITH_AES_128_CBC_SHA256,
TLS_RSA_WITH_AES_256_CBC_SHA,TLS_RSA_WITH_AES_128_CBC_SHA


#-------------------------------------------------------------------
# Available cipher suites with VSE:
#
# SSL_RSA_WITH_NULL_MD5             - deprecated
# SSL_RSA_WITH_NULL_SHA             - deprecated
# SSL_RSA_EXPORT_WITH_DES40_CBC_SHA - deprecated
# SSL_RSA_WITH_DES_CBC_SHA          - deprecated
# SSL_RSA_WITH_3DES_EDE_CBC_SHA     - deprecated
# TLS_RSA_WITH_AES_128_CBC_SHA
# TLS_RSA_WITH_AES_256_CBC_SHA
#
# The following cipher suites are only available with OpenSSL:
# TLS_RSA_WITH_NULL_SHA256          - deprecated
# TLS_RSA_WITH_AES_128_CBC_SHA256
# TLS_RSA_WITH_AES_256_CBC_SHA256
# SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
# TLS_DHE_RSA_WITH_AES_128_CBC_SHA
# TLS_DHE_RSA_WITH_AES_256_CBC_SHA
# TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
```

```
# TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
# TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
# TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
# TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
# TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
#------------------------------------------------------------------
```

このプロパティー・ファイルは、サンプルで使用するように事前にカスタマイズされているので、通常、変更する必要はありません。 変更する場合は、詳しくは、48 ページの『ステップ 2: VSEScriptServer プロパティー・ファイルを構成する』を参照してください。

ステップ 2: 接続プロパティー・ファイルのセットアップ

Connections.properties ファイルで、作業を行う z/VSE システムの設定値を定義します。 **Connections.properties** は、VSE スクリプト・サーバーがインストールされているディレクトリーにあります。このファイルの例を次に示します。

```
connection.1.password=mypassw
connection.1.ip=9.12.34.56
connection.1.name=VSE01
connection.1.port=2893
connection.1.userid=jsch
connection.timeout=100
```

Connections.properties の詳細については、50 ページの『ステップ 3: 接続プロパティー・ファイルを構成する』を参照してください。

このファイルは、任意のテキスト・エディターを使用して編集できます。 ご使用の VSE パスワードを、パラメーター `password` に平文で入力してください。 VSE スクリプト・サーバーが開始され、接続プロパティー・ファイルが読み取られると、パスワードは暗号化されてパラメーター `encpassword` に入り、 `password` はファイルから除去されます。 接続プロパティー・ファイルの詳細については、50 ページの『ステップ 3: 接続プロパティー・ファイルを構成する』を参照してください。

ステップ 3: サンプル VSAM データの定義

VSE/ESA 2.5 以降、サンプル・ジョブ SKVSSAMP が提供され、ICCF ライブラリー 59 に保管されています。 これを使用すると、さまざまな z/VSE e-business コネクターのサンプルで使用できるサンプル・データを定義することができます。

ジョブ SKVSSAMP をサブミットして、VSAM.CONN.SAMPLE.DATA という名前の VSAM クラスターを、VSESP.USER.CATALOG に定義します。 これには、「中古車」を説明する多くのレコードが入ります。 ジョブ SKVSSAMP の詳細については、226 ページの『4. VSAM データ・クラスターの定義』を参照してください。 ジョブ SKVSSAMP は自動的に IDCAMS RECMAP コマンドを使用して、サンプル・データのフィールド名が入る VSAM マップを定義します。 詳細については、119 ページの『RECMAP を使用したマップの定義』を参照してください。

ステップ 4: サンプル VSE スクリプトの変更

561 ページの図 187 に、サンプルの VSE スクリプト **getdata.src** を示します。このスクリプトは、『ステップ 3: サンプル VSAM データの定義』で定義された VSAM ファイルにあるレコードを読み取り、データをスプレッドシートに挿入します。 **getdata.src** は、VSE スクリプト・サーバーがインストールされているディレ

クトリーのサブディレクトリー **/scripts/samples** にあります。 このレコードは、パラメーターで VSE スクリプトに受け渡されるキーによって識別されます。 この VSE スクリプトは、VSE スクリプト・クライアント **VSEScriptClient.123** の Lotus 1-2-3 の例で使用されます。

z/VSE ホストが (次の例の **VSE01** として示されているように定数 **host** を使用して) 正しく定義されていることを確認してください。 このホスト名は、**Connections.properties** で定義されているホスト名に一致しなければなりません。

```
// constants
String host      = "VSE01";
String file      = "VSESP.USER.CATALOG¥¥VSAM.CONN.SAMPLE.DATA¥¥USEDCARS";

// Variables
String keyfields;
String keyvalues;
String fields;
String values;
int rc;

// prepare the fields
keyfields[0] = "ARTICLENO";
keyvalues[0] = argv[0];   // argument is key

fields[0] = "ARTICLENO";
fields[1] = "MANUFACTURER";
fields[2] = "TYPE";
fields[3] = "MODEL";
fields[4] = "HP";
fields[5] = "DISPLACEMENT";
fields[6] = "CYLINDERS";
fields[7] = "COLOUR";
fields[8] = "FEATURES";
fields[9] = "PRICE";

// get the record
getVSAMRecord(host,file,&keyfields,&keyvalues,&fields,&values, &rc);

// print received data

if (rc!=0) do;
  println("Not found");
else do;
  println(values[0]);
  println(values[1]);
  println(values[2]);
  println(values[3]);
  println(values[4]);
  println(values[5]);
  println(values[6]);
  println(values[7]);
  println(values[8]);
  println(values[9]);
endif;
```

図 *187. VSE* スクリプト・コネクターの例で提供されている *VSE* スクリプト

ステップ 5: z/VSE ホストでの VSE コネクター・サーバー の開始

VSE コネクター・サーバー が非 **SSL** モードで開始されていることを確認してください。サーバーは、デフォルトでは、クラス R で実行されます。 読み取り待ち行列にあるジョブ STARTVCS を使用してサーバーを開始してください (詳しくは 39 ページの『VSE コネクター・サーバー の始動』を参照)。

ステップ 6: ローカルでの VSE スクリプト・サーバーの開始

ローカルで VSE スクリプト・サーバーを開始するには、以下のいずれかを使用します。

- Windows では **runserver.bat**
- OS/2 では **runserver.cmd**
- Linux/Unix ワークステーションでは **runserver.sh**

ステップ 7(a): サンプルの Lotus 1-2-3 スプレッドシート・ファイルのオープン

サンプルの Lotus スプレッドシート・ファイル **VSEScriptClient.123** には、VSE スクリプトを実行するのに必要なセットアップがすでに行われています。 **VSEScriptClient.123** は、VSE スクリプト・サーバーがインストールされているディレクトリーにあります。 ファイルをオープンするために Windows Explorer を使用してファイルをダブルクリックすると、図 188 に示すように、「**Execute Script** (スクリプトの実行)」ボタンが表示されます。

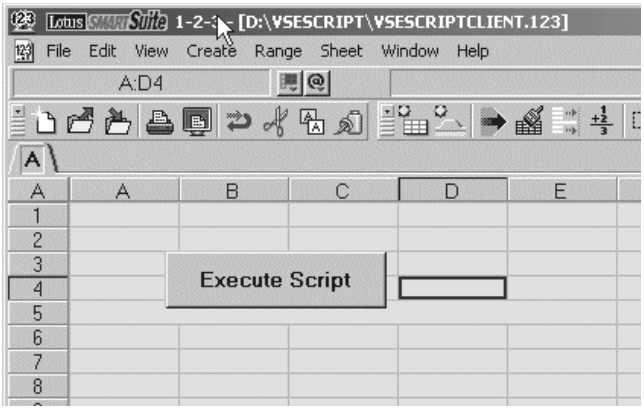


図 *188. VSE* スクリプト・コネクターの例のサンプル *Lotus 1-2-3* スプレッドシート

VSE スクリプト・サーバーが、ご使用のスプレッドシートと同じワークステーションで実行されている場合は、「**Execute Script** (スクリプトの実行)」を押すと、563 ページの図 189 に示すように、データが VSAM クラスターから Lotus スプレッドシートに転送されます。 VSE スクリプト・サーバーが、ご使用のスプレッドシートと同じワークステーションで実行されていない 場合は、まず、例えば 563 ページの『Lotus 1-2-3 で VSE サンプル・スクリプトを定義する方法』 に示されているスクリプト・エディターを使用して Visual Basic スクリプトを変更し、その Visual Basic スクリプトに、VSE スクリプト・サーバーが実行されているワ

ークステーションの IP アドレスまたはホスト名を入れなければなりません。

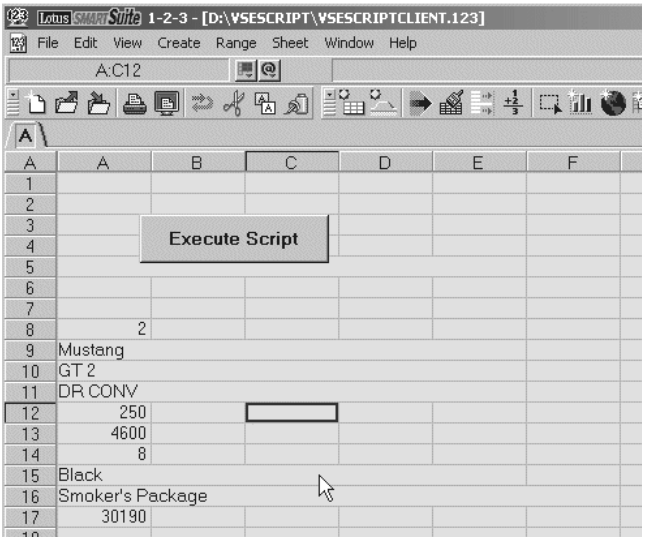


図 *189. VSAM* クラスターから *Lotus 1-2-3* スプレッドシートへのデータの転送

Lotus 1-2-3 で VSE サンプル・スクリプトを定義する方法

Lotus 1-2-3 などのオフィス・アプリケーション内からサンプルの VSE スクリプトを使用できるようにするには、 (図 190 に示すように) 例えば Visual Basic スクリプトを使用して VSE スクリプトを定義しなければなりません。 サンプル・ファイル **VSEScriptClient.123** をオープンし、さらに、スクリプト・エディターをオープンすると、図 190 に示すように、「Globals」セクションに、グローバル関数宣言が表示されます。

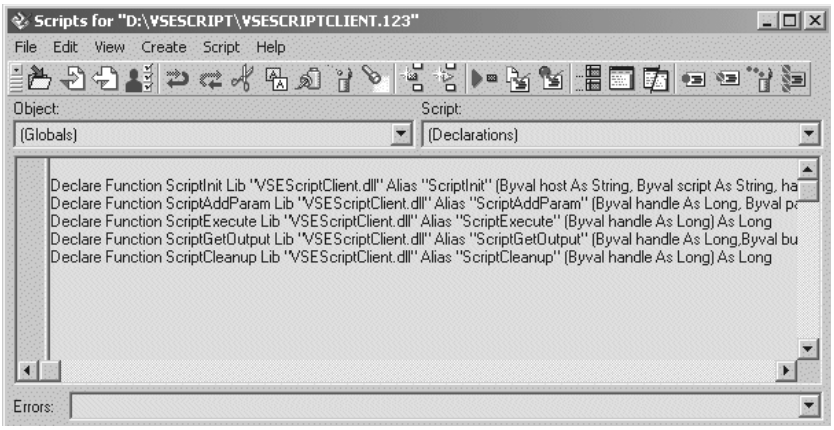


図 *190. Lotus 1-2-3* に定義されたサンプル・スクリプト

また、 564 ページの図 191 に示すように、**Button 1** に関連したセクションに Visual Basic スクリプトが表示されます (図 189 では、**Button 1** には、 **Execute Script** というラベルが付いていました)。

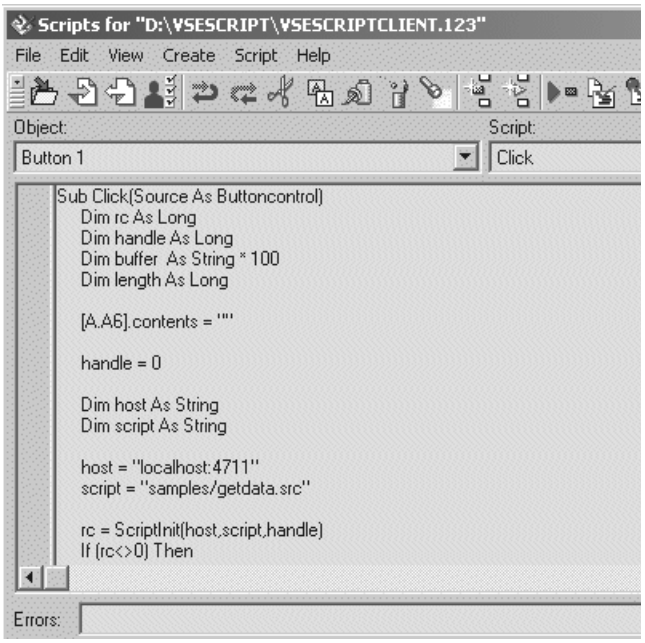


図 *191. Lotus 1-2-3* スプレッドシートの例で使用されている *Visual Basic* スクリプト

注**: VSEScriptClient.dll** は、オフィス・アプリケーションでアクセス可能でなければなりません。 この DLL を Lotus 1-2-3 DLL ディレクトリーにコピーするか、DLL と同じディレクトリーにあるサンプル・スプレッドシート・ファイルをダブルクリックしてください。

次に、VSE スクリプトを Lotus 1-2-3 で実行するために使用される VSE スクリプト・クライアントの完全 Visual Basic コードを示します。 太字で示した部分は、Lotus スプレッドシート環境に固有のものです。

```
Sub Click(Source As Buttoncontrol)
  Dim rc As Long
  Dim handle As Long
  Dim buffer  As String * 100
  Dim length As Long

  [A.A6].contents = ""

  handle = 0

  Dim host As String
  Dim script As String

  host = "localhost:4711"
  script = "samples/getdata.src"

  rc = ScriptInit(host,script,handle)
  If (rc0) Then
     [A.A6].contents = "rc = " &rc
     Goto finish
  End If

  rc = ScriptAddParam(handle,"2")
  If (rc0) Then
     [A.A6].contents = "rc = " &rc
     Goto finish
  End If
```

```
  rc = ScriptExecute(handle)
  If (rc0) Then
     [A.A6].contents = "rc = " &rc
     Goto finish
  End If

  Dim counter As Long
  Dim rows As Range
  Dim cell As Variant

  Set rows = Bind("A8..A65535")

  counter = 0
  Do
     rc = ScriptGetOutput(handle,buffer,100,length)
     If (rc=0) Then
        buffer = Left(buffer,length)

        Set cell = rows.Cell(counter,0,0)
        cell.contents = buffer

        counter = counter + 1
     End If
  Loop While rc=0

  Finish:

  rc = ScriptCleanup(handle)
End Sub
```

ステップ 7(b): サンプル MS Office スプレッドシートのオープン

サンプルの MS Office スプレッドシート・ファイル **VSEScriptClient.xls** には、VSE スクリプトを実行するのに必要なセットアップがすでに行われています。**VSEScriptClient.xls** は、VSE スクリプト・サーバーがインストールされているディレクトリーにあります。 ファイルをオープンするために Windows Explorer を使用してファイルをダブルクリックすると、図 192 に示すように、「**Execute Script** (スクリプトの実行)」ボタンが表示されます。

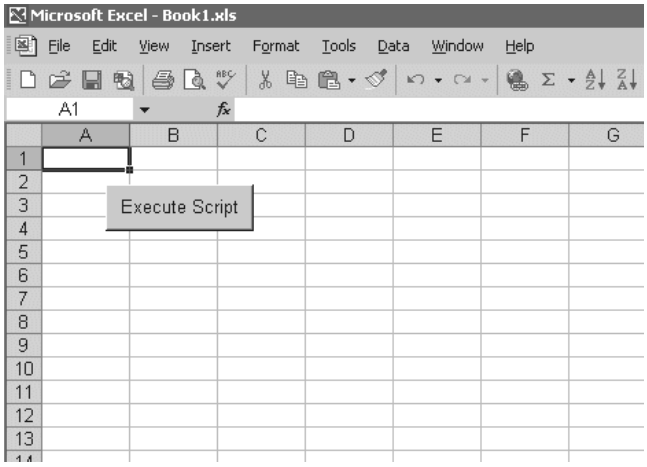


図 *192. MS Office* スプレッドシートの例のサンプル・スプレッドシート

VSE スクリプト・サーバーが、ご使用のスプレッドシートと同じワークステーションで実行されている場合は、「**Execute Script** (スクリプトの実行)」を押すと、566 ページの図 193 に示すように、データが VSAM クラスターから MS Office

スプレッドシートに転送されます。VSE スクリプト・サーバーが、ご使用のスプレッドシートと同じワークステーションで実行されていない 場合は、まず、例えば 567 ページの図 194 に示されているスクリプト・エディターを使用して Visual Basic スクリプトを変更し、その Visual Basic スクリプトに、VSE スクリプト・サーバーが実行されているワークステーションの IP アドレスまたはホスト名を入れなければなりません。

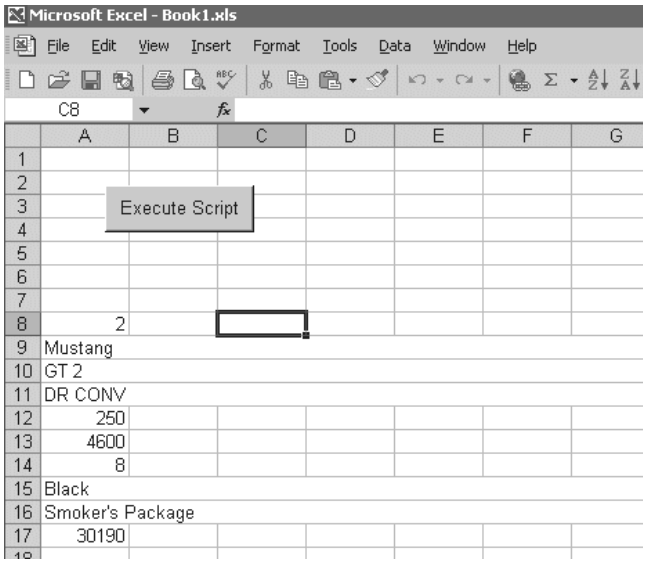


図 *193. VSAM* クラスターから *MS Office* スプレッドシートへのデータの転送

MS Office へのサンプル VSE スクリプトの定義方法

MS Office などのオフィス・アプリケーション内からサンプルの VSE スクリプトを使用できるようにするには、 (567 ページの図 194 に示すように) 例えば Visual Basic スクリプトを使用して VSE スクリプトを定義しなければなりません。 サンプル・ファイル **VSEScriptClient.xls** をオープンし、さらに、Visual Basic エディターをオープンすると、グローバル関数宣言と Visual Basic スクリプトが表示されます。

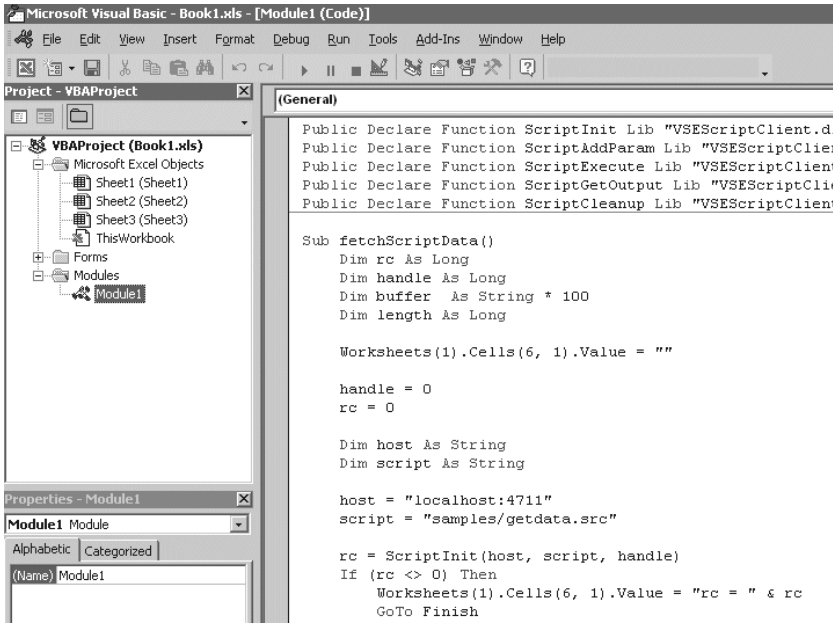


図 *194. MS Office* に定義されているサンプル・スクリプト

注**: VSEScriptClient.dll** は、オフィス・アプリケーションでアクセス可能でなければなりません。 この DLL を MS Office ディレクトリーにコピーするか、DLL と同じディレクトリーにあるサンプル・スプレッドシート・ファイルをダブルクリックしてください。

次に、VSE スクリプトを MS Office で実行するために使用される VSE スクリプト・クライアントの完全 Visual Basic コードを示します。 太字で示した部分は、MS Office スプレッドシート環境に固有のものです。

```
Attribute VB_Name = "Module1"
Public Declare Function ScriptInit Lib "VSEScriptClient.dll" (ByVal host As
   String, ByVal script As String, handle As Long) As Long
Public Declare Function ScriptAddParam Lib "VSEScriptClient.dll" (ByVal handle
   As Long, ByVal parameter As String) As Long
Public Declare Function ScriptExecute Lib "VSEScriptClient.dll" (ByVal handle
   As Long) As Long
Public Declare Function ScriptGetOutput Lib "VSEScriptClient.dll" (ByVal handle
   As Long, ByVal buffer As String, ByVal maxlen As Long, retlen As Long) As Long
Public Declare Function ScriptCleanup Lib "VSEScriptClient.dll" (ByVal handle
   As Long) As Long
Sub fetchScriptData()
    Dim rc As Long
    Dim handle As Long
    Dim buffer  As String * 100
    Dim length As Long

    Worksheets(1).Cells(6, 1).Value = ""

    handle = 0
    rc = 0

    Dim host As String
    Dim script As String

    host = "localhost:4711"
    script = "samples/getdata.src"

    rc = ScriptInit(host, script, handle)
```

```
    If (rc <> 0) Then
        Worksheets(1).Cells(6, 1).Value = "rc = " & rc
        GoTo Finish
    End If

    rc = ScriptAddParam(handle, "2")
    If (rc <> 0) Then
        Worksheets(1).Cells(6, 1).Value = "rc = " & rc
        GoTo Finish
    End If

    rc = ScriptExecute(handle)
    If (rc <> 0) Then
        Worksheets(1).Cells(6, 1).Value = "rc = " & rc
        GoTo Finish
    End If

    Dim counter As Long

    counter = 0
    Do
        rc = ScriptGetOutput(handle, buffer, 100, length)
        If (rc = 0) Then
            buffer = Left(buffer, length)

            Worksheets(1).Cells(counter + 8, 1).Value = buffer

            counter = counter + 1
        End If
    Loop While rc = 0

Finish:

    rc = ScriptCleanup(handle)
End Sub
```

ステップ 7(c): コマンド行からのサンプル VSE スクリプトの開始

コマンド行からサンプル VSE スクリプトを開始するには、VSE スクリプト・サーバーがインストールされているディレクトリーにあるバッチ・ファイル **runscript.bat** を使用できます。

例えば、**break_cont.src** という名前のサンプル・スクリプトを開始するには、コマンド行に次のように入力します。

```
runscript samples¥break_cont.src
```

注: **samples** ディレクトリーには、学習およびテストの目的でユーザーが使用できる、このほかのサンプル・スクリプトが入っています。

VSE スクリプト・クライアントを使用してデータを中間層から取得

IBM では、z/VSE の下で実行される 2 つの VSE スクリプト・クライアントを提供しています。

- バッチ・プログラム として稼働する VSE スクリプト・クライアント。
- *CICS* クライアント として稼働する VSE スクリプト・クライアント。

これらの VSE スクリプト・クライアントを使用して、VSE スクリプト・サーバーのスクリプトを呼び出して物理/論理中間層からデータを取得できます。

z/VSE 4.2 以降の VSE スクリプト・クライアントでは、VSE スクリプト・サーバーへの接続に対する *SSL/TLS* 暗号化接続 をサポートします。SSL/TLS を使用するには、以下の作業を実行済みである必要があります。

- TCP/IP での SSL/TLS を有効にする。
- 必要な鍵および証明書を以下に作成する。
  - VSE スクリプト・サーバー・サイド
  - z/VSE.

両方の VSE スクリプト・クライアントとも、

1. 入力として 80 バイトの固定サイズの行を取得します。
2. 出力として 80 バイトの固定サイズの行を戻します。

- 最初の入力行は、VSE スクリプト・サーバーが稼働しているサーバーのホスト名にしてください。
- SSL=NO の場合は、呼び出されるスクリプトの名前を 2 番目の入力行に入れる必要があります。
- SSL=YES の場合は、以下のようにしてください。
  - 2 番目の入力行に、使用する SSL バージョンを入れる必要があります (例えば、SSL30 または TLS31)。
  - 3 番目の入力行に、SSL メンバーを含んでいる VSE 鍵リング・ライブラリーの名前を入れる必要があります (各 SSL メンバーには鍵および証明書が含まれます)。
  - 4 番目の入力行に、VSE 鍵リング・ライブラリー内の必要な SSL メンバーの名前を入れる必要があります。
  - 5 番目の入力行に、使用する SSL 暗号仕様を入れる必要があります。
  - 6 番目の入力行に、SSL セッション・タイムアウト (秒) を入れる必要があります。
  - 7 番目の入力行に、呼び出されるスクリプトの名前を入れる必要があります。
- これ以降のすべての入力行は、呼び出したスクリプトに指定されたパラメーターで成り立っています (各行が、単一のパラメーター)。
- パラメーターが 1 行よりも長くなる場合は、行連結 を使用する必要があります。

(*SSL* パラメーター および行連結 についてはこのセクションで後述します)。

以下は、IBM 提供の両方の VSE スクリプト・クライアントで使用されるパラメーターです。

- CODEPAGE
- SHOWERROR
- NOCONT
- SSL
- ALLOWCLIENTAUTH
- SYMBOLS

CODEPAGE パラメーターは、z/VSE 入力データと出力データのコード・ページを指定します。

- VSE スクリプト・サーバーは、ASCII ベースなので、すべての入出力データは、EBCDIC から ASCII へ、またはその逆に変換する必要があります。
- 変換は、指定したコード・ページを使用して VSE スクリプト・サーバー内で行われます。コード・ページが指定されていない場合は、デフォルトのコード・ページ IBM-1047 が使用されます。

SHOWERROR パラメーターは YES か NO に設定できます。

- NO に設定した場合、スクリプト・エラーを含む行 (これはスクリプト出力の最後にあります) が除去されます。
- YES (デフォルト) に設定した場合、スクリプト・エラーを含む行 (これはスクリプト出力の最後にあります) が保存されます。

NOCONT パラメーターは、行連結を入力行に使用するべきか、そうでないかを指定します。NOCONT パラメーターは YES か NO に設定できます。

- NO (デフォルト) に設定した場合、行連結が使用されます。 1 行より長い長さを持つパラメーターを使用する場合は、NOCONT を NO に設定してください。
  - 「-」で終わるそれぞれの行には、それに追加される入力データが次の行にあります (「-」は除去されます)。
  - 「-」で終わるパラメーター行を使用する場合、2 つのダッシュ (すなわち「--」) で行を終わらせてください。 次の入力行を空にする必要もあります。
- YES に設定した場合、入力行は書き込まれたとおりに使用されます。

SSL パラメーターは、VSE スクリプト・サーバーへの接続に SSL/TLS を使用するかどうかを指定します。

- SSL=NO (デフォルト) の場合は、VSE スクリプト・サーバーへの接続で SSL/TLS を使用せず、接続は暗号化されません。
- SSL=YES の場合は、VSE スクリプト・サーバーへの接続が SSL/TLS を使用して暗号化されます。

ALLOWCLIENTAUTH パラメーターは、VSE スクリプト・サーバーへの接続に SSL クライアント認証を使用するかどうかを指定します。このパラメーターは、SSL=YES の場合にのみ有効です。

- ALLOWCLIENTAUTH=NO (デフォルト) の場合は、VSE スクリプト・サーバーへの SSL 接続で SSL クライアント認証を使用しません。必要な唯一の VSE 鍵リング・ライブラリー・メンバーは .CERT です。
- ALLOWCLIENTAUTH=YES の場合は、VSE スクリプト・サーバーへの接続で SSL クライアント認証を使用できます。VSE 鍵リング・ライブラリーは、メンバー .CERT、 .ROOT、および .PRVK を含む必要があります。

VSE スクリプト・サーバーが、クライアント認証を実行するかどうかを決定します。サーバーでクライアント認証を要求としているときに、VSE スクリプト・クライアントでクライアント認証を許可していない場合は、接続が失敗します。

SYMBOLS パラメーターは、スクリプト入力テキスト内のシンボルを解決するかどうかを指定します。

- SYMBOLS=NO (デフォルト) の場合は、スクリプト入力テキスト内のシンボルは解決されません。
- SYMBOLS=YES の場合は、スクリプト入力テキスト内のシンボルは、そのシンボルによって定義されている値で置き換えられます。

それぞれの VSE スクリプト・ファイルは、常にスクリプト関数 `exit(<returncode>)` の呼び出しで終了させてください。

- `exit()` 関数に提供される `returncode` は、VSE ジョブの戻りコードとして使用されます。
- スクリプトがなんらかの理由で失敗した場合、スクリプト・エラー・コードが、ジョブの戻りコードとして使用されます。
- スクリプトが `exit()` を呼び出さず、失敗しない場合は、`returncode` はゼロになります。

バッチで稼働する VSE スクリプト・クライアントの使用

バッチ用の *VSE* スクリプト・クライアント はジョブによって起動されます。プログラムは IESSCBAT.phase です。

1. 起動後、VSE スクリプト・クライアントが、VSE スクリプト・サーバーに接続します。
2. VSE スクリプト・クライアントは、指定したスクリプトを呼び出し、すべての入力データ行をスクリプトに送ります。
3. VSE スクリプト・クライアントは、スクリプトからすべての出力行を受け取り、ジョブの SYSLST に行を書き込みます。
4. 呼び出したスクリプトの戻りコードは、ジョブの戻りコードと同じです。

注**:** VSE スクリプト・クライアントで使用する一部の戻りコードは、ローカル・エラーを示します。 ローカル・エラーをリモート・スクリプト・エラーと確実に区別できるようにするには、呼び出したスクリプトで、これらの戻りコードを使用しない ようにする必要があります。

バッチ用の VSE スクリプト・クライアントは、ジョブの PARM ステートメント内に含まれる 3 つのパラメーターの値を使用します。 パラメーターが PARM に指定されていない場合は、デフォルト値が使用されます。 パラメーターは、以下のように指定します。

```
CODEPAGE=xxx
SHOWERROR=YES|NO
NOCONT=YES|NO
SSL=YES|NO
ALLOWCLIENTAUTH=YES|NO
SYMBOLS=YES|NO
```

ジョブの SYSIPT 入力には以下が含まれます。

- 接続先の z/VSE ホスト名 (最初の行に)。
- SSL=YES の場合は、すべて の SSL パラメーター (1 行に 1 つ)。
- 呼び出すスクリプト名 (1 行で入力)。
- スクリプトにパスされるパラメーター (残りの行すべて)。

以下は、バッチ用の VSE スクリプト・クライアントを呼び出すジョブの例です。

```
* $$ JOB JNM=START,DISP=L,CLASS=A
// JOB START
// LIBDEF *,SEARCH=(PRD2.TCPIPC,PRD1.BASE,PRD2.SCEEBASE,PRD2.DBASE)
// EXEC IESSCBAT,PARM='CODEPAGE=Cp1047 SHOWERROR=yes'
10.31.0.1:4711
testscript.src
First parameter.
This is a longer parameter that takes two input-lines on SYSIPT. It -
is provided as a single parameter to the invoked script.
This third parameter line ends with a --
/*
/&
* $$ EOJ
```

以下は、バッチ用の VSE スクリプト・クライアントを呼び出し、SSL=YES を使用するジョブの例です。

```
* $$ JOB JNM=START,DISP=L,CLASS=A
// JOB START
// LIBDEF *,SEARCH=(PRD2.TCPIPC,PRD1.BASE,PRD2.SCEEBASE,PRD2.DBASE)
// EXEC IESSCBAT,PARM='CODEPAGE=Cp1047 SHOWERROR=yes SSL=yes'
10.31.0.1:4711
SSL30
CRYPTO.KEYRING
SCRPTKEY
010208090A62
86400
testscript.src
First parameter.
This is a longer parameter that takes two input lines on SYSIPT. It -
is provided as a single parameter to the invoked script.
This third parameter line ends with a --
/*
/&
* $$ EOJ
```

z/VSE 5.1 以降では、バッチで使用する VSE スクリプト・クライアントの SYSIPT 入力でシンボリック・パラメーター を使用できます。

- このシンボリック・パラメーターは実行時に解決され、パラメーターがシンボリック・パラメーターの値で置き換えられます。
- シンボリック・パラメーターが JCL で使用されるときと同じ規則が、シンボリック・パラメーターが SYSIPT 入力で使用されるときにも適用されます。そのため、詳細については、資料「*z/VSE System Control Statements*」の章『Job Control and Attention Routine』にあるシンボリック・パラメーターの説明を参照してください。
- シンボル処理を有効にするには、パラメーター SYMBOLS を YES に設定する必要があります。デフォルトは NO です。

バッチ用に VSE スクリプト・クライアントを呼び出して SYSIPT 入力でシンボリック・パラメーター を使用するジョブを以下に例示します。

```
* $$ JOB JNM=START,DISP=L,CLASS=A
// JOB START
// LIBDEF *,SEARCH=(PRD2.TCPIPC,PRD1.BASE,PRD2.SCEEBASE,PRD2.DBASE)
// SETPARM IP='10.31.0.1'
// SETPARM PORT='4711'
// EXEC IESSCBAT,PARM='SYMBOLS=YES CODEPAGE=Cp1047 SHOWERROR=yes'
&IP:&PORT
testscript.src
First parameter.
This is a longer parameter that takes two input lines on SYSIPT. It -
```

```
is provided as a single parameter to the invoked script.
This third parameter line ends with a --
/*
/&
* $$ EOJ
```

下記にリストされたものは、ローカル・エラーが VSE スクリプト・クライアントで発生した場合に生成される戻りコードです (10 進数)。

- 以下は、一般戻りコード です。
  - **1** ERR_NULL_PTR
  - **2** ERR_INVALID_HANDLE
  - **3** ERR_INVALID_STATUS
  - **4** ERR_UNKNOWN_HOST
  - **5** ERR_CONNECT_FAILED
  - **6** ERR_CONNECTION_BROKEN
  - **7** ERR_NO_MORE_DATA
  - **8** ERR_EMPTY_PARAM
  - **9** ERR_ICONV_OPEN_FAILED
  - **10** ERR_ICONV_FAILED
  - **11** ERR_SSL_NOT_SUPPORTED
  - **12** ERR_SSL_INIT_FAILED
  - **13** ERR_SSL_UNKNOWN_KEYNAME
  - **14** ERR_SSL_HANDSHAKE_FAILED
- 以下は、バッチ・クライアント特有の戻りコード です。
  - **16** ERR_OUT_OF_MEMORY
  - **17** ERR_NO_SYSIPT_INPUT
  - **18** ERR_UNEXPECTED_END_OF_SYSIPT
  - **19** ERR_INVALID_PARM
  - **20** ERR_SYMBLIB_ERROR

CICS で実行される VSE スクリプト・クライアントの使用

CICS 用の *VSE* スクリプト・クライアントは、バッチ用の VSE スクリプト・クライアント とよく似た方法で作動します。 571 ページの『バッチで稼働する VSE スクリプト・クライアントの使用』 に示す情報は、以下の方法を除いて CICS 用の VSE スクリプト・クライアントにも適用されます。

- CICS クライアントが呼び出されます。
- 入出力データが供給されます。

CICS クライアントは、IESSCCIC という名前を持つプログラムです。 これは、各 CICS プログラムの EXEC CICS LINK ステートメントを使用して呼び出すことができます。

以下は、INPUT COMMAREA のレイアウトです。

```
char[12] codepage;          // codepage of input/output data
unsigned short flags;       // 0x0001=SHOWERROR
                            // 0x0002=NOCONT
                            // 0x0004=SSL
                            // 0x0008=ALLOWCLIENTAUTH
                            // 0x0010=SYMBOLS
```

```
                                // 0x1000=PRINTERRORSTOSYSLST
unsigned short int lineCount; // number of input-lines,
...input-lines...             // each 80 bytes long
```

以下は、OUTPUT COMMAREA のレイアウトです。

```
int clientRc;                 // return code of client, see below
int scriptRc;                 // return code of script
unsigned short int lineCount; // number of output lines,
...output lines...            // each 80 bytes long
```

下記にリストされたものは、ローカル・エラーが VSE スクリプト・クライアントで発生した場合に生成される戻りコードです (10 進数)。

- 以下は、一般戻りコード です。
  - **1** ERR_NULL_PTR
  - **2** ERR_INVALID_HANDLE
  - **3** ERR_INVALID_STATUS
  - **4** ERR_UNKNOWN_HOST
  - **5** ERR_CONNECT_FAILED
  - **6** ERR_CONNECTION_BROKEN
  - **7** ERR_NO_MORE_DATA
  - **8** ERR_EMPTY_PARAM
  - **9** ERR_ICONV_OPEN_FAILED
  - **10** ERR_ICONV_FAILED
  - **11** ERR_SSL_NOT_SUPPORTED
  - **12** ERR_SSL_INIT_FAILED
  - **13** ERR_SSL_UNKNOWN_KEYNAME
  - **14** ERR_SSL_HANDSHAKE_FAILED
- 以下は、*CICS* クライアント特有の戻りコード です。
  - **16** ERR_OUT_OF_MEMORY
  - **17** ERR_UNEXPECTED_END_OF_INPUT
  - **18** ERR_INVALID_COMMAREA
  - **19** ERR_INVALID_LINECOUNT
  - **20** ERR_SYMBLIB_ERROR
  - **32** WARN_OUTPUT_LINES_TRUNCATED

フィールド clientRc 戻りコードが以下のいずれかの場合、フィールド scriptRc は、呼び出した VSE スクリプト・ファイルの戻りコードで設定されます。

- ゼロ (エラーなし)。
- 32 (出力行が切り捨てられたことを示す警告)。

その他のすべての clientRc 戻りコードの場合、フィールド scriptRc は無効です。

CICS クライアントは、EXEC CICS LINK が呼び出された後に戻りコードを RESP2 フィールドに戻します。 この戻りコードは、出力 COMMAREA フィールド clientRc で設定されたものと同じです (1 つの例外あり)。

- 提供された COMMAREA が 8 バイトより小さいと、エラー・コードは COMMAREA に合いません。 COMMAREA は変更されないまま、戻りコード ERR_INVALID_COMMAREA が RESP2 フィールドに戻されます。

- INPUT COMMAREA には、CODEPAGE パラメーターが含まれています。これは入出力データのコード・ページを指定します。 コード・ページ・パラメーターが 2 進ゼロまたは空 (ブランクのみ) の場合、デフォルトのコード・ページが使用されます。
- INPUT COMMAREA の FLAGS パラメーターを使用すれば、SHOWERROR、NOCONT、SSL、ALLOWCLIENTAUTH、SYMBOLS、および PRINTERRORSTOSYSLST パラメーターを設定できます。
  - パラメーターのフラグが設定されている場合、これは YES を示します。
  - パラメーターのフラグが設定されていない場合、これは NO を示します。
  - 例えば、SHOWERROR と NOCONT を YES に設定する場合、X'0003' に対して FLAGS パラメーターを設定する必要があります。
- INPUT COMMAREA の LINECOUNT パラメーターは、COMMAREA に含まれる入力行数を指定します。
- 入力行は、LINECOUNT パラメーターの後から開始します。

OUTPUT COMMAREA には、以下が含まれます。

- スクリプト・クライアントの戻りコード。
- 呼び出されたスクリプトの戻りコード。
- 出力行数。
- 出力行は、LINECOUNT パラメーターの後から開始します。
  - 呼び出されたスクリプトからの出力行が、指定された COMMAREA に合わない場合、それらは切り捨てられます。
  - 出力行が切り捨てられた場合、クライアント戻りコードは、WARN_OUTPUT_LINES_TRUNCATED に設定されます。

CICS 用 **VSE** スクリプト・クライアントのセットアップ

- プログラム IESSCCIC は、デフォルトで DBDCCICS に定義されます。
- このプログラムを別の CICS システムで定義する必要がある場合は、CEDA DEFINE PROGRAM コマンドの以下のオプションを使用してください。

```
CEDA  DEFine PROGram( IESSCCIC )
 PROGram        : IESSCCIC
 Group          : VSESPG
 DEscription  ==>
 Language     ==> C
 RELoad       ==> Yes
 RESident     ==> No
 USAge        ==> Normal
 USEsvacopy   ==> No
 Status       ==> Enabled
 RSl            : 00
 Cedf         ==> Yes
 DAtalocation ==> Any
 EXECKey      ==> Cics
REMOTE ATTRIBUTES
 REMOTESystem ==>
 REMOTEName   ==>
 Transid      ==>
 EXECUtionset ==> Fullapi
```

第 5 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。本書は他の言語で IBM から提供されている場合があります。ただし、これを入手するには、本製品または当該言語版製品を所有している必要がある場合があります。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。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。

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 の将来の方向性および指針に関するすべての記述は、予告なく変更または撤回される場合があります。これらは目標および目的を提示するものにすぎません。

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、類似する人物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。

プログラミング・インターフェース情報

本書には、プログラムを作成するユーザーが z/VSE のサービスを使用するためのプログラミング・インターフェースが記述されています。

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、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。

Java およびすべての Java 関連の商標およびロゴは Oracle やその関連会社の米国およびその他の国における商標または登録商標です。

Linux は、Linus Torvalds の米国およびその他の国における商標です。

Microsoft、Windows、Windows NT および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

UNIX は The Open Group の米国およびその他の国における登録商標です。

製品資料のご使用条件

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

IBM Web サイトの「ご利用条件」に加えて、以下のご使用条件が適用されます。

個人使用

これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、非商業的な個人による使用目的に限り複製することができます。ただし、IBM の明示的な承諾をえずに、これらの資料またはその一部について、二次的著作物を作成したり、配布 (頒布、送信を含む) または表示 (上映を含む) することはできません。

商業的使用

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。

アクセシビリティー

アクセシビリティー機能は、運動障害または視覚障害など身体に障害を持つユーザーがソフトウェア製品を快適に使用できるようにサポートします。 z/VSE のアクセシビリティーの主要機能により、ユーザーは以下のことができるようになります。

- 画面読み上げ機能および画面拡大機能などの支援機能の使用
- キーボードのみを使用して、特定の機能または画面を使用したのと同等の機能を操作
- 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

支援機能の使用

画面読み上げ機能などの支援機能は、z/VSE のユーザー・インターフェースを使用して機能します。 この支援機能を使用して z/VSE インターフェースにアクセスする場合、その特定情報については支援機能の資料を参照してください。

資料の形式

本製品の資料は、Adobe Portable Document Format (PDF) で提供され、アクセシビリティー標準に準拠しています。 PDF ファイルの使用に問題があり、Web ベース形式の資料を必要とする場合は、s390id@de.ibm.com 宛てに E メールを送信するか、または下記の宛先まで書面でご請求ください。

```
IBM Deutschland Research & Development GmbH
Department 3282
Schoenaicher Strasse 220
D-71032 Boeblingen
Federal Republic of Germany
```

この請求には必ず、資料番号および表題を付記してください。

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。

用語集

この用語集は、「z/VSE e-business コネクター ユーザーズ・ガイド」で使用される技術用語および省略語を定義しています。 お探しの用語が見つからない場合には、*IBM Dictionary of Computing* (**www.ibm.com/ibm/terminology**) をご覧ください。

この用語集には、記号 * 付きの定義があります。これは、「IBM コンピューティング辞典」から転載した定義です。

アプレット **(* applet)**
: Java プログラム言語で作成されたアプリケーション・プログラムの 1 つで、Web サーバーからリトリーブされ、Web ブラウザーによって実行できる。 アプレットを指すリファレンスは、グラフィックス・ファイルを指すリファレンスが表示される場合と同じ方法で Web ページのマークアップに表示される。ブラウザーは、グラフィックス・ファイルをリトリーブするのと同じ方法でアプレットをリトリーブする。セキュリティー上の理由で、アプレットのアクセス権限は、以下の 2 つ、すなわち、アプレットは、そのアプレットが実行されているクライアントのファイル・システムにアクセスできない、および、ネットワークを介したアプレットの通信は、アプレットがダウンロードされた元のサーバーにだけ許される、という方法に限られる。サーブレット *(servlet)* と対比。

非対称暗号方式 **(Asymmetric cryptography)**
: 公開鍵暗号方式 *(public key cryptography)* と同義語。

認証 **(* authentication)**
: (1) コンピューター・セキュリティーにおいて、ユーザーの識別の検査、または、オブジェクトに対するユーザーのアクセス資格の検査。 (2) コンピューター・セキュリティーにおいて、メッセージが変更されていない、または破壊されていないことの検査。 (3) コンピューター・セキュリティーにおいて、情報システムまたは保護リソースのユーザーを検査するために使用するプロセス。

バイトコード **(bytecode)**
: *Java* バイトコード *(Java bytecode)* を参照。

CA
: 認証局 *(Certificate Authority)* を参照。

CICS ECI
: CICS 外部呼び出しインターフェース (CICS External Call Interface) (CICS ECI) は、CICS Transaction Server for z/VSE によって提供される *CICS* ビジネス・ロジック・インターフェース の 1 つの可能なリクエスター・タイプ。 これは CICS クライアントの一部であり、これを使用すると、ワークステーション・プログラムが z/VSE ホスト上の CICS 関数を呼び出せる。

CICS EPI
: CICS 外部表示インターフェース (CICS EPI) は CICS クライアントの一部。これによって非 CICS クライアント・アプリケーションが論理 3270 端末として機能し、CICS 3270 アプリケーションを制御できるようになる。

CICS EXCI
: 外部 CICS インターフェース (EXternal CICS Interface) (EXCI) は、CICS Transaction Server for z/VSE によって提供される *CICS* ビジネス・ロジック・インターフェース の 1 つの可能なリクエスター・タイプ。 すべての VSE バッチ・アプリケーションが CICS 関数を呼び出せるようにする。

共通コネクター・フレームワーク **(Common Connector Framework (CCF))**
: IBM の *Visual Age for Java* の一部で、リモート・ホストへの接続の作成と維持を可能にする。 CCF クラスは VSEConnector.jar ファイルにあり、VSE Java Beans によって内部的に使用され

る。 CCF は、例えば、サーブレットが中間層プラットフォームで実行される、複数層アーキテクチャーで重要。 CCF では、オープン接続がプールで保持できるので、サーブレットが呼び出されるたびに必要な、リモート z/VSE ホストへの TCP/IP 接続のオープンおよびクローズにかかる時間を避けられる。

Common Object Request Broker Architecture (CORBA) (* Common Object Request Broker Architecture (CORBA))
Object Management Group (OMG) によって作成された仕様で、さまざまなタイプのオブジェクト要求ブローカー (object request broker) (クライアント常駐 ORB、サーバー・ベースの ORB、システム・ベースの ORB、および、ライブラリー・ベースの ORB) で使用する規格を提供する。 CORBA 規格のインプリメンテーションにより、さまざまなソフトウェア・ベンダーのオブジェクト要求ブローカーが共同操作できるようになる。

ConnectionManager クラス **(ConnectionManager class)**
CCF の一部で、リモート z/VSE ホストへの接続を識別する。中間層とリモート z/VSE サーバーとの間の接続を保持する。 サーブレットはプールにある接続を予約し、処理し、後で戻すことができる。この動作は、VSE Java Beans を使用して内部的に実行される。

コネクター **(connector)**
z/VSE のコンテキストにおいて、コネクターは、2 つのプラットフォーム (Web クライアントと z/VSE ホスト、中間層と z/VSE ホスト、または Web クライアントと中間層) を接続するミドルウェアを提供する。 これらのコネクターは、本書で説明されている。

コンテナー **(container)**
IBM WebSphere Application Server などのアプリケーション・サーバーの JVM の一部で、リソース管理およびトランザクション管理のリソースを提供することにより、サーブレット、EJB、および JSP のインプリメンテーションを容易にする。 例えば、EJB 開発者は、アプリケーション・サーバーの JVM に対してコーディングしてはならないが、コンテナーによって提供されるインターフェースに対してはコーディングできる。 コンテナーの主な役割は、EJB とクライアントの間の仲介として機能し、また、複数の EJB インスタンスを管理することにある。 作成された EJB は、アプリケーション・サーバーにあるコンテナーに保管する必要がある。 次に、コンテナーは、すべてのスレッド化、および、EJB とのクライアント相互作用を管理し、さらに、接続とインスタンス・プーリングを調整する。

暗号トークン **(cryptographic token)**
通常は単にトークン と呼ばれ、デジタル署名の生成、またはデータの暗号化などの暗号関数を実行するためのインターフェースを提供する装置。

暗号方式 **(* cryptography)**
(1) 意味を隠すためのデータのトランスフォーメーション。 (2) コンピューター・セキュリティーにおいて、「プレーン・テキスト」を暗号化し、「暗号文」を復号するための原則、手段、メソッド。

Db2 ベース・コネクター
VSE/ESA 2.5 で導入されたフィーチャー。Db2 ストアード・プロシージャーを使用して、Db2、VSAM、および DL/I データにアクセスを提供する、VSAM および DL/I 機能性を備えたカスタマイズ済み Db2 バージョンが組み込まれている。

Db2 ストアード・プロシージャー **(Db2 Stored Procedure)**
z/VSE のコンテキストにおいて、Db2 ストアード・プロシージャーは、Db2 データにアクセスする言語処理環境 (LE) プログラム。ただし、VSE/ESA 2.5 以降、ユーザーは、Db2 ストアード・プロシージャーを使用して VSAM データおよび DL/I データにもアクセスできる。したがって、VSAM と Db2 との間でデータの交換が可能である。

Data Encryption Standard (DES)
コンピューター・セキュリティーにおいて、米国政府によって連邦情報処理標準 (FIPS) 資料 46 として採用された、米国連

邦情報・技術局 (NIST) のデータ暗号化規格。データ暗号化アルゴリズムのハードウェア・インプリメンテーションのみを扱う。

復号 **(* Decryption)**
コンピューター・セキュリティーにおいて、エンコードされたテキストすなわち暗号文をプレーン・テキストにトランスフォームするプロセス。

DES *Data Encryption Standard* を参照。

デジタル署名 **(* digital signature)**
コンピューター・セキュリティーにおいて、受信側が送信側の識別を確認できることを可能にする、メッセージまたはメッセージの一部に付加された暗号化されたデータ。

デジタル署名アルゴリズム **(Digital Signature Algorithm (DSA))**
デジタル署名アルゴリズムは、米国政府によって定義されたデジタル署名用の規格。DSA デジタル署名は、規則のセット (すなわち DSA)、および、署名者の識別とデータの保全性が検証できるパラメーターのセットを使用して計算された大きな数のペアである。 DSA は、シグニチャーを生成し、検証する機能を提供する。

DSA デジタル署名アルゴリズム *(Digital Signature Algorithm)* を参照。

ECI *CICS ECI* を参照。

暗号化 **(* Encryption)**
コンピューター・セキュリティーにおいて、オリジナル・データを取得できないようにする、あるいは、復号プロセスを使用することによってのみ取得できるように、データを理解しにくいフォームにトランスフォームするプロセス。

Enterprise Java Bean (EJB)
EJB は、分散 Java Bean である。 「分散」とは、EJB の 1 つの部分が Web アプリケーション・サーバーの JVM の内側で実行され、ほかの部分が、Web ブラウザーの JVM の内側で実行されることを意味する。 EJB は、データベース内の 1 つのデータ行 (Entity Bean)、あるいは、リモート・データベースへの 1 つの接続 (Session Bean) のどちらかを表す。 通常、両方のタイプの EJB が一緒に作動する。 これによって、リレーショナル・データと非リレーショナル・データが混在する異種環境で、標準化された方法でデータを表しアクセスすることが可能になる。 *Java Bean* も参照。

EPI *CICS EPI* を参照。

EXCI *CICS EXCI* を参照。

FCP (* FCP)
ファイバー・チャネル・プロトコル を参照。

ファイバー・チャネル接続 **(FICON®) (*fibre-channel connection (FICON))**
IBM メインフレーム・コンピューターおよび周辺装置のために設計されたファイバー・チャネル通信プロトコル。

ファイバー・チャネル・プロトコル **(FCP) (* Fibre Channel Protocol (FCP))**
ファイバー・チャネル・ネットワークで使用されるシリアル SCSI コマンド・プロトコル。

FICON (* FICON)
ファイバー・チャネル接続 を参照。

ファイアウォール **(* firewall)**
通信において、1 つのネットワークからほかのネットワークへの接続を保護し制御する機能装置。 ファイアウォールは、(a) 望まない、あるいは無許可の通信トラフィックが、保護ネットワークに入らないようにする、および、(b) 選択された通信トラフィックだけが保護ネットワークを終了できるようにする。

ハッシュ関数 **(hash function)**
ハッシュ関数は、可変サイズの入力データを受けて、ハッシュ値 と呼ばれる固定サイズのストリングを戻すトランスフォーメーションである。 暗号方式において、ハッシュ関数には、以下のプロパティーがある。

- ハッシュ関数は計算が簡単。
- ハッシュ関数は片方向。つまり、「逆」関数を計算することは不可能。

- ハッシュ関数には衝突がない。つまり、異なる入力値が同じハッシュ値になることは不可能。

ハッシュ値 **(hash value)**
テキストにハッシュ関数 を適用した結果得られる固定サイズのストリング。

ホーム・インターフェース **(home interface)**
新規 EJB オブジェクトをインスタンス化し、EJB をイントロスペクトし、さらに、EJB のインスタンス化を除去するメソッドを提供する。 リモート・インターフェースの場合は、デプロイメント・ツールがインプリメンテーション・クラスを生成するので、インターフェースだけが必要である。 それぞれの Session Bean のホーム・インターフェースは、少なくとも 1 つの *create()* メソッドを提供する必要がある。

ホップ **(*hop)**
経路指定されたネットワークで、隣接するノードとの間の伝送パスの 1 セグメント。

HTTP セッション **(HTTP Session)**
z/VSE のコンテキストにおいて、サーブレットを呼び出す Web ブラウザー・クライアントを識別する (すなわち、クライアントと中間層プラットフォームとの間の接続を識別する)。

インターネット **(* internet)**
産業、教育、政府、およびリサーチの分野で、数多くの異種ネットワークを接続する広域ネットワーク。 インターネット・ネットワークは、情報を伝送する規格として TCP/IP (Transmission Control Protocol/Internet Protocol) を使用する。

interface definition language (IDL) (* interface definition language (IDL))
CORBA において、オブジェクト・インプリメンテーションに関係せずに、オブジェクト・インターフェースを記述するために使用する宣言言語。

JAR プラットフォームとは独立したファイル・フォーマットで、多くのファイルを 1 つのファイルに集合する。 複数のアプレットおよびその必要コンポーネント (.class ファイル、イメージ、および音) が JAR ファイルにバンドルされ、次に、単一 HTTP トランザクションを使用して Web ブラウザーにダウンロードされる (ダウンロード速度が大幅に改善される)。 また、JAR フォーマットは圧縮をサポートし、ファイル・サイズが削減される (さらにダウンロード速度が改善される)。 使用される圧縮アルゴリズムは、ZIP アルゴリズムと完全に互換性がある。 アプレットの所有者は、JAR ファイル内の個々の項目にデジタルに署名し、その発信元を認証できる。

Java アプレット **(Java applet)**
アプレット を参照。

Java アプリケーション **(Java application)**
Web ブラウザーの JVM の内側で実行される Java プログラム。 プログラムのコードは、ローカル・ハード・ディスク上、または LAN 上にある。 Java アプリケーションは、グラフィカル・インターフェースを使用した大規模プログラムの場合がある。 Java アプリケーションは、ローカル・リソースに無制限にアクセスできる。

JavaBeans (* JavaBeans)
「Bean」という再使用可能な Java コンポーネントを構築するための、プラットフォームに依存しない、ソフトウェア・コンポーネント・テクノロジー。 これらの Bean は、構築されると、ほかのソフトウェア・エンジニアが使用できるようになるか、または、Java アプリケーションで使用できるようになる。 また、JavaBeans を使用すると、ソフトウェア・エンジニアは、グラフィカルなドラッグ・アンド・ドロップ開発環境で、Bean を操作しアセンブルできる。

Java バイトコード **(Java bytecode)**
バイトコードは、Java ソース言語ステートメントが入っているファイルがコンパイルされるときに作成される。 コンパイル済み Java コードすなわち「バイトコード」は、実行される (命令がコンピューター内で一時に 1 つずつ実行される) 準備ができたプログラム・モジュールまたはファイルに似る。 ただし、バイトコード内の命令は、実際に、*Java* 仮想マシン に対する命令である。 命令を一時に 1 つずつ

解釈する代わりに、バイトコードは、ジャストインタイム (JIT) コンパイラーを使用して、それぞれのオペレーティング・システムのプラットフォームごとに再コンパイルされる。 通常、これにより、Java プログラムは、より速く実行される。 バイトコードは、接尾部 **.CLASS** を持つバイナリー・ファイルに入る。

Java Database Connectivity (JDBC) (* Java Database Connectivity (JDBC))
Open Database Connectivity (ODBC) と同じ特性を持つが、特に Java データベース・アプリケーション用に設計されたアプリケーション・プログラミング・インターフェース (API)。 また、JDBC ドライバーがないデータベースに対して、JDBC は JDBC から ODBC へのブリッジ (JDBC から ODBC に変換するメカニズム) を組みこむ。 JDBC は JDBC API を Java データベース・アプリケーションに提示し、これを ODBC に変換する。 JDBC は、Sun Microsystems, Inc. およびさまざまなパートナーとベンダーによって開発された。

Java Development Kit (JDK) (* Java Development Kit (JDK))
Java アプレットおよびアプリケーションを作成、コンパイル、デバッグ、および実行するために使用できるソフトウェア・パッケージ。

Java Runtime Environment (JRE) (* Java Runtime Environment (JRE))
Java Development Kit (JDK) のサブセットで、標準的な Java プラットフォームを構成する核となる実行可能モジュールとファイルが入っている。 JRE には、Java 仮想マシン、コア・クラス、および、それをサポートするファイルが組み込まれている。

JavaScript (* JavaScript)
Java に似たスクリプト言語で、Netscape ブラウザーで使用するために Netscape によって開発された。

Java Server Page (JSP)
HTML ページに似た Web ページ。 JSP の部分は、Web ページを要求ブラウザーに送信している間に、 Web サーバーの JSP エンジンによってコンパイルされてサーブレットの中に入れられる。 JSP には、サーブレットが変更されるたびに、開発者がサーブレットを再コンパイルする必要がないという利点がある。 変更は常時、JSP ファイルに対して行われる。 また、JSP は、Web ページを動的に作成する。

Java サーブレット (Java servlet)
サーブレット を参照。

Java 仮想マシン (* Java Virtual Machine (JVM))
コンパイル済み Java コード (アプレットおよびアプリケーション) を実行する、中央演算処理装置 (CPU) のソフトウェア・インプリメンテーション。

Lotus Domino Server
各ユーザーの複合データベース を保管するために使用される Lotus Notes サーバー。

Multipurpose Internet Mail Extensions (MIME)
インターネットを介して転送されるオブジェクトのタイプを識別するためのインターネット規格。 MIME タイプは、オーディオ、グラフィックス、およびビデオのいくつかの変形を含む。

nonce (ランダム・ストリング) (*nonce)
(1) ランダムで固有のテキスト・ストリングで、データと一緒に暗号化され、次に、この暗号化されたデータを送信するシステムに対する攻撃を検出するために使用されます。nonce は、特に認証のために使用され、データを暗号化するたびに暗号化されたデータが異なるようになります。 (2) リプレイ・アタックの検出に役立てるためにメッセージに埋め込まれる、固有の暗号番号。

パーシスタンス (persistence)
アプリケーションが終了する際に破棄されるのではなく、長期にわたり存続が可能な、データベース内のオブジェクトのストレージの記述に使用される用語。
WebSphere のようなエンタープライズ Java Bean コンテナーは、コンテナー内に

デプロイされている EJB に対してパーシスタンス・サービスを提供する。

永続ストレージ **(persistent storage)**
パーシスタンス *(Persistence)* を参照。

PKCS (Public Key Cryptography Standards)
PKCS は、RSA Data Security, Inc. によって発行された、公開鍵暗号方式のインプリメンテーションのための規格のセット。

PKI 公開鍵インフラストラクチャー *(Public key infrastructure)* を参照。

ポート **(* port)**
ディスプレイ装置およびプリンターなどのほかの装置のケーブルが接続される装置上のコネクター。

秘密鍵 **(* private key)**
コンピューター・セキュリティーにおいて、所有者のみが知っている鍵。 公開鍵暗号方式 *(Public key cryptography)* を参照。

プロキシー・サーバー **(* proxy server)**
別のサーバーに向けた要求を受け取り、さらに、要求されたサービスを取得するためにクライアントに代わって (クライアントのプロキシーとして) 行動するサーバー。 プロキシー・サーバーは、多くの場合、クライアントとサーバーが直接接続するには非互換である場合 (例えば、クライアントは、サーバーのセキュリティー認証要件を満たさないが、あるサービスを受けることは許される場合) に使用されます。

公開鍵 **(* public key)**
コンピューター・セキュリティーにおいて、情報を暗号化したいすべてのユーザーに使用可能になっている鍵。 公開鍵暗号方式 *(Public key cryptography)* を参照。

公開鍵暗号方式 **(* public key cryptography)**
コンピューター・セキュリティーにおいて、公開鍵が暗号化のために使用され、秘密鍵が復号のために使用される暗号方式。 非対称暗号方式 *(Asymmetric cryptography)* と同義語。

リモート・メソッド呼び出し **(RMI) (remote method invocation (RMI))**
CORBA の Java だけのバージョン。 RMI は Java 固有であるので、CORBA より使いやすい。オブジェクトを記述するために IDL を作成する代わりに、 **rmic** というプログラムを Java クラス・ファイルで実行でき、これが、スタブ・クラスとスケルトン・クラスをクラス・ファイルから直接作成する。 RMI は、例えば、EJB によって、クライアント・スタブと EJB のサーバー部分との間の通信に使用される。

リモート・インターフェース **(remote interface)**
z/VSE のコンテキストにおいて、リモート・インターフェースを使用すると、クライアントが EJB (EJB がリモート z/VSE ホストにある場合でも) にメソッド呼び出しを行うことができる。 コンテナーはリモート・インターフェースを使用して、クライアント・サイド・スタブとサーバー・サイド・プロキシー・オブジェクトを作成し、クライアントから EJB への着呼メソッド呼び出しを処理する。

リモート・プロシージャー・コール **(RPC) (* remote procedure call (RPC))**
(1) サーバーからプロシージャー呼び出しの実行を要求するためにクライアントが使用する機能。この機能には、プロシージャーのライブラリーと外部データ表現が組み込まれる。 (2) 別のノードにあるサービス・プロバイダーへのクライアントの要求。

RSA アルゴリズム **(RSA algorithm)**
アルゴリズムの発明者 Rivest、Shamir、および Adleman の名前をとって付けられた公開鍵のアルゴリズム。

秘密鍵 **(secret key)**
秘密鍵 *(private key)* と同義語。

Secure Electronic Transaction (SET)
インターネットなどのオープン・ネットワークを使用した支払いカード・トランザクションを保護するための公開された仕様。 SET は、Visa および MasterCard、さらに、 GTE、 IBM、 Microsoft、 Netscape、 SAIC、 Terisa Systems、および VeriSign などのテクノロジー組織の参加を得て開発された。 SET は、RSA Data Security からの暗号化テクノロジーを基にする。

Secure Sockets Layer (SSL)
クライアントがサーバーを認証し、すべてのデータおよび要求が暗号化されるセキュリティー・プロトコル。 SSL は、Netscape Communications Corp. および RSA Data Security, Inc. によって開発された。 SSL バージョン 3.0 の後継は、1999 年に Transport Layer Security (TLS) 1.0 と名付けられました。

サーブレット **(* servlet)**
Java プログラム言語で作成され、Web サーバーで実行されるアプリケーション・プログラム。 サーブレットを指すリファレンスは、グラフィックス・ファイルを指すリファレンスが表示される場合と同じ方法で Web ページのマークアップに表示される。 Web サーバーはサーブレットを実行し、実行の結果 (結果がある場合) を Web ブラウザーに送信する。 アプレット *(applet)* と対比。

SET *Secure Electronic Transaction* を参照。

スケルトン **(skeleton)**
CORBA のコンテキストにおいて、スタブの等価であるが、サーバー (z/VSE ホスト) 上で使用される。 スケルトンは、ネットワークから着信するデータを再アセンブルして認識可能なフォーマットにし、サーバー・サイド・メソッドを呼び出し、返信をクライアントに戻す。

socks enabled (* socks enabled)
Socks プロトコル を認識する TCP/IP ソフトウェア、または、特定の TCP/IP アプリケーションに関係する用語。「Socksified」は「socks-enabled」を意味するスラング用語。

socksified (* socksified)
Socks 対応 *(socks enabled)* を参照。

Socks プロトコル **(* socks protocol)**
機密保護機能のあるネットワーク内のアプリケーションが、*Socks* サーバー *(socks server)* を使用し、ファイアウォールを通って通信できるようにするプロトコル。

Socks サーバー **(* socks server)**
ファイアウォールを通って、非セキュアなネットワーク内のサーバー・アプリケーションに、セキュアな片方向接続を提供する回線レベル・ゲートウェイ。

SSL *Secure Sockets Layer (SSL)* を参照。

スレッド **(* thread)**
プロセスを制御しているコンピューター命令のストリーム。 マルチスレッド・プロセスは、命令の 1 つのストリーム (1 つのスレッド) で始まり、タスクを実行するために、後で、ほかの命令ストリームを作成する場合がある。

TLS 「*Transport Layer Security*」を参照。

Transport Layer Security (TLS)
最新の SSL 暗号プロトコル。プライバシーおよびデータ保全性に対する強度が向上した。

Uniform Resource Locator (URL)
(1) コンピューター、またはインターネットなどのネットワーク上の、情報リソースを表す文字のシーケンス。 文字のシーケンスには、 (a) 情報リソースにアクセスするために使用されるプロトコルの省略名、および、(b) 情報リソースを見つけるためにプロトコルによって使用される情報、が含まれる。 例えば、インターネットのコンテキストにおいて、さまざまな情報にアクセスするために使用されるプロトコルの省略名には、http、ftp、gopher、telnet、および news があり、また、IBM のホーム・ページの URL は、`http://www.ibm.com` である。 (2) ワールド・ワイド・ウェブ上の項目のアドレス。これには、プロトコルと、それに続く完全修飾ドメイン・ネーム (ホスト名と呼ばれる場合もある)、および要求が組み込まれる。 Web サーバーは、通常、URL の要求部分をパスとファイル名にマップする。例えば、URL が `http://www.networking.ibm.com/nsg/nsgmain.htm` である場合、プロトコルは `http`、完全修飾ドメイン名は `www.networking.ibm.com`、要求は `/nsg/nsgmain.htm` である。

URL *Uniform Resource Locator (URL)* を参照。

VSE コネクター・サーバー **(VSE Connector Server)**
VSE Java Beans のホスト部分であり、

z/VSE のインストール時に VSE/POWER の読み取り待ち行列に入れられるジョブ STARTVCS を使用して開始される。 デフォルトで、動的クラス R で実行される。

VSE Java Beans
すべての VSE ベースのファイル・システム (VSE/VSAM、ライブラリアン、VSE/POWER、および VSE/ICCF) にアクセスを許可し、ジョブをサブミットし、さらに、z/VSE オペレーター・コンソールにアクセスする Java Bean。 クラス・ライブラリーは、*VSEConnector.jar* アーカイブに入っている。 *JavaBeans* も参照。

WebSphere Application Server
サーブレット、JSP、および EJB を処理するために、IBM HTTP Server と一緒に使用される。 また、特定の JDK および Db2 が必要。

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]

[ラ行]

[数字]

A

B

C

D

E

F

G

H

I

J

L

M

O

P

R

S

T

U

V

W

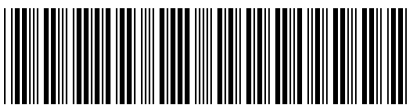
X

Z

プログラム番号: 5686-VS6

Printed in Japan

SC43-4402-00



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