- ClusteredHashTable クラス
- CommandCall クラス
- ConnectionPool クラス
- DataArea クラス
- データ変換およびデータ記述クラス
  - 数値変換クラス
  - テキスト (文字) 変換クラス
  - 複合 (数値およびテキスト) 変換クラス
  - FieldDescription クラス
  - RecordFormat クラス
  - Record クラス
  - LineDataRecordWriter クラス
- DataQueue クラス
  - 順次データ待ち行列
  - キー順データ待ち行列
- ディジタル証明書クラス
- EnvironmentVariable クラス
- 例外
- FTP クラス
- 統合ファイル・システム・クラス
  - IFSFile クラス
  - IFSJavaFile クラス
  - IFSFileInputStream クラス
  - IFSTextFileInputStream クラス
  - IFSFileOutputStream クラス
  - IFSTextFileOutputStream クラス
  - IFSRandomAccessFile クラス
  - IFSFileDialog クラス
  - IFSKey クラス
  - 共用モード
- JavaApplicationCall クラス

- JDBC クラス
  - JDBC の拡張
  - JDBC プロパティー
  - Blob クラス
  - CallableStatement クラス
  - Clob クラス
  - Connection クラス
  - ConnectionPool クラス
  - DatabaseMetaData クラス
  - DataSource クラス
  - Driver クラス
  - ParameterMetaData クラス
  - PreparedStatement クラス
  - ResultSet および ResultSetMetaData クラス
  - RowSet クラス
  - Savepoint クラス
  - Statement クラス
  - XAConnection および XAResource クラス
- ジョブ・クラス
  - Job クラス
  - JobList クラス
  - JobLog クラス
- AS400Message クラス
- NetServer クラス
- Permission および UserPermission クラス
  - DLOPermission クラス
  - QSYSPermission クラス
  - RootPermission クラス
- 印刷クラス
  - PrintObjectList クラス
  - PrintObject クラス

IBM Toolbox for Java

IBM Toolbox for Java は、 Java プログラムを使用して iSeries および AS/400e サーバー上のデータにアクセスするための、 Java(TM) クラスのセットです。 これらのクラスを使用して、 iSeries 上のデータを扱うクライアント/サーバー・アプリケーション、アプレット、およびサーブレットを作成することができます。 また、iSeries Java 仮想マシン (JVM) 上で、 IBM Toolbox for Java クラスを使用する Java アプリケーションを実行することもできます。

IBM Toolbox for Java は、システムへのアクセス・ポイントとして iSeries ホスト・サーバーを使用します。 Toolbox for Java は Java に組み込まれた通信機能を使用するので、Toolbox for Java を使うために IBM iSeries Access Express for Windows を使用する必要はありません。 各サーバーはサーバー上の別々のジョブで実行され、 各サーバー・ジョブはソケット接続時にデータ・ストリームを送受信します。

メイン・ナビゲーション・バーまたは以下のリンクを使用して、 IBM Toolbox for Java についての理解を深めてください。

V5R2 の新機能
重要な変更点、拡張された機能、その他の注目すべき項目について参照してください。

IBM Toolbox for Java トピックの印刷
Toolbox for Java トピックの PDF を表示またはダウンロードします。 ZIP 圧縮された Toolbox for Java トピックもダウンロードできます。

クラス・ファインダーの使用
簡単かつすみやかに、名前と記述によってクラスを検索したり、パッケージ別にクラスを表示したり、 Java クラス用のすべての Toolbox のリストをアルファベット順に調べることができます。

IBM Toolbox for Java: はじめに
IBM Toolbox for Java のインストールの管理についてがわかります。 ワークステーションおよびサーバーにインストールする方法を理解します。 簡単なプログラミング例を使用して、 アプリケーションにおいて Toolbox for Java クラスの使用を開始する方法を学びます。

IBM Toolbox for Java クラス
iSeries および AS/400e サーバー・データの処理を可能にする、 IBM Toolbox for Java パッケージ内の種々のクラスについて参照してくだ

さい。 この情報には、IBM Toolbox for Java プログラムの作成に役立つ、解説、コード例、 および技術情報が含まれます。

Graphical Toolbox を使用してユーザー独自の GUI パネルを作成する
Graphical Toolbox は、Java アプリケーション、アプレット、 または iSeries ナビゲーターのプラグインに組み込むことのできる、 Java のカスタム・ユーザー・インターフェース・パネルを作成するために使用します。

JavaBeans
Javasoft JavaBean 標準に合わせて構築されている Toolbox for Java のパブリック・クラスを使用して、JavaBeans を作成することについて参照してください。 プログラム内で JavaBeans を使用する方法を示す例を検討します。

JDBC
Toolbox for Java によって提供される JDBC サポートについてがわかります。 JDBC を使用すると、プログラムは構造化照会言語 (SQL) ステートメントをサーバー上のデータベースに発行し、 そのデータベースからの結果を処理できます。

PCML を使用して iSeries プログラムを呼び出す
プログラム呼び出しマークアップ言語 (PCML) を使用すると、 より少ない量の Java コードしか使用しないで、iSeries プログラムを呼び出すのに役立ちます。 PCML は Java アプリケーションが呼び出す iSeries プログラムの入出力パラメーターを完全に記述する、 XML に基づくタグ構文です。

Proxy サポート
データを暗号化する Sockets Layer (SSL) プロトコルの使用を含む、IBM Toolbox for Java Proxy サポートの使用法です。

» RFML を使用してデータ形式を定義して管理する
レコード様式マークアップ言語 (RFML) を使用して、 データ様式仕様と、Java プログラムのビジネス・ロジックを分離します。 RFML は XML に基づくタグ構文で、PCML と密接な関係があり、 Java アプリケーションが特定の種類のレコード内のフィールドを指定して取り扱えるようにします。 «

» セキュリティー
Java Secure Socket Extension (JSSE) および Toolbox for Java で提供される セキュアなデータ通信を使用して、TCP/IP よりも上位層で実行されるアプリケーションをクライアント・サーバー間で 行う場合に参考になります。 «

» iSeries システム・デバッガー
iSeries システム・デバッガー・グラフィカル・ユーザー・インターフェース (GUI) を使用して、 iSeries サーバー上で実行するプログラムをデバッグしテストします。 «

システム・プロパティー
たとえば、Proxy サーバー、またはトレースのレベルを定義する時など、 IBM Toolbox for Java のさまざまな局面を構成するためにシステム・プロパティーを使用する方法を学びます。 システム・プロパティーを使用して、コードを再コンパイルせずに、便利な実行時構成を実行できます。

» IBM Toolbox for Java 2 Micro Edition
この新しい Toolbox for Java コンポーネントを使用して、 さまざまなワイヤレス・デバイス用の Java プログラムを作成できます。 ToolboxME for iSeries を使用して、 ワイヤレス・デバイスは iSeries サーバーのデータおよびリソースに直接アクセスできます。 «

Javadocs for IBM Toolbox for Java
IBM Toolbox for Java クラスに関する javadoc 参照情報を表示します。

» よく尋ねられる質問 (FAQ)
IBM Toolbox for Java のパフォーマンスの最適化、トラブルシューティングの実行、 JDBC の使用などに関する質問への回答がわかります。 «

以下の追加情報が含まれています。

- Toolbox for Java プログラミング例のリスト
- Toolbox for Java の使用に役立つプログラミングに関するヒント
- Java、サーブレット、XML などについての詳細な情報へのリンクを含む、関連情報

注: 法律上の重要な情報に関しては、 コードの特記事項情報をお読みください。

V5R2 の新機能

IBM Toolbox for Java は、以下の形式で入手できます。

- IBM Toolbox for Java のライセンス・プログラム 5722-JC1、 バージョン 5 リリース 2 (V5R2) は、OS/400 V4R5 およびそれ以降にインストールします。 クライアントから接続する場合、IBM Toolbox for Java は OS/400 V4R5 以降に接続できます。
- また、OS/400 には、IBM Toolbox for Java の非グラフィカル・クラスも組み込まれています。 これは、IBM Toolbox for Java クラスを iSeries Java 仮想マシン (JVM) 上で実行する際に使用するために最適化されています。 したがって、ライセンス・プログラムのグラフィック機能 (グラフィカル・クラス) を必要としない場合でも、 IBM Toolbox for Java をそのまま使用することができます。 詳細については、JAR ファイルを参照してください。
- また、IBM Toolbox for Java は現在オープン・ソースになっています。 JTOpen Web サイトからコードをダウンロードし、 情報を入手することができます。

新規パッケージ

IBM Toolbox for Java 2 Micro Edition は、IBM Toolbox for Java の新規パッケージです。 新規の com.ibm.as400.micro パッケージは、 携帯情報端末 (PDA) や携帯電話のようなワイヤレス装置で実行される Java プログラムを書くことを可能にする、 クラスのセットを実装しています。 その micro パッケージは、別個にダウンロードする必要があります。

iSeries システム・デバッガーは、iSeries サーバーで実行される ILE、Java、C、および C++ のプログラムのための新しいグラフィカル・デバッグ環境をワークステーションに提供します。

新規クラス

また、IBM Toolbox for Java V5R2 は既存のパッケージ内に多数の新規クラスを備えています。 新規クラスを使用して、以下を行うことができます。

- ClusteredHashTable クラスを使用して、 クラスター内のノード間で非持続データを共用し、複製する。
- CommandPrompter クラスを使用して、 与えられたコマンドにパラメーターを入力するようにプロンプトを出す。
- 新規 JDBC(TM) クラスについては、 新規 JDBC クラスおよび拡張機能を

参照してください。

- RecordFormatDocument クラスを使用すると、 新規のレコード・フォーマット・マークアップ言語 (RFML) コンポーネントを使用してレコード様式を指定したり、 データ・レコードの作成、読み取り、および書き込みをすることが可能になります。

拡張クラス

IBM Toolbox for Java V5R2 は、既存のクラスに対する拡張も組み込んでいます。 これらの拡張により、次のものが提供されます。

- AS400 オブジェクトへの Kerberos サポートの追加。 これは Toolbox for Java サーバーへの認証に、 Java Generic Security Service (JGSS) フレームワークを使用するようになりました。
- SecureAS400 オブジェクトは、 Java Secure Socket Extention (JSSE) フレームワークを使用して、 クライアントとサーバー間で流れるデータを暗号化できるようになりました。
- プログラム呼び出しマークアップ言語 (PCML) のいくつかのタグおよび機能の変更点:
  - <data> タグの新規属性は、 ユニコードのストリングをサポートし、 ユーザーがストリングからスペースを切り取る方法を指定することが可能になります。
  - <program> タグの新規および変更された属性により、 実行時にパスを指定できるようになり、 さらにサービス・プログラムのエントリー・ポイント名のための CCSID を指定できるようになりました。
  - トレースをする場合には、PCML は PcmlMessageLog クラスの代わりに、 トレース・クラス (com.ibm.as400.access.Trace) を使用するようになりました。

新規 JDBC クラスおよび拡張機能

IBM Toolbox for Java V5R2 の JDBC サポートは、新規クラスおよび拡張機能を備えており、 これには JDBC 3.0 アプリケーション・プログラミング・インターフェース (API) のサポートも含まれています。 目立った変更点には以下のものが含まれます。

- AS400JDBCSavepoint は、 トランザクション・ロールバックに対するよりきめ細かな制御を行う (JDBC 3.0 をサポートする) 新規のクラスです。
- AS400JDBCParameterMetaData は、 PreparedStatement オブジェクトおよび CallableStatement オブジェクト内の パラメーターのタイプおよびプロパ

ティーを取り出すことを可能にする、 (JDBC 3.0 をサポートする) 新規のクラスです。
- 独立補助記憶域プール (IASP) に接続するための機能が追加されました。
- バイナリー・ラージ・オブジェクト (blob) および文字ラージ・オブジェクト (clob) の (JDBC 3.0 をサポートする) 新規のメソッドを使用すると、それらのデータ型の値を更新することが可能になります。
- 新規の JDBC プロパティー (拡張メタデータ) により、 ResultSetMetaData 属性を報告する機能が改善されました。
- JDBC サポートを改善するその他の拡張

新規 XML コンポーネント

V5R2 では、IBM Toolbox for Java に、 PCML に類似した XML 拡張機能である レコード・フォーマット・マークアップ言語 (RFML) が追加されました。 RFML を使用すると、XML を Java プログラムの中で使用して、 データ・バッファーの形式および物理ファイル・レコード形式を指定することが可能になります。 さらに、取り出したバッファーやレコードの内容を調べることも可能になります。

Graphical Toolbox での追加機能

Graphical Toolbox には、以下の新機能が組み込まれています。

- テーブル列のセルをチェック・ボックスとして表示します。
- ダイアログの要素が適切に表示されるように、ダイアログの最小の高さと幅を指定します。
- テーブルの最初の列にダイナミック階層ツリーが含まれるように指定します (各セルはツリーのノードに対応します)。

これらの機能の詳細については、GUI ビルダーのオンライン・ヘルプを参照してください。

互換性

IBM Toolbox for Java は、 Netscape(R) Navigator あるいは Microsoft(R) Internet Explorer のデフォルトの JVM での実行は、 もはやサポートしていません。 Toolbox for Java のクラスを使用してブラウザーで実行されるアプレットのために、 Sun Java 2 Runtime Environment (JRE) 1.3.0 プラグイン のようなプラグインをインストールする必要があります。

Toolbox for Java は、もはや data400.jar を組み込んでいません。 data400.jar に含

まれていたクラスは、現在 jt400.jar にあります。 data400.jar を CLASSPATH ステートメントから取り除いてください。

ResultSet および CallableStatement のための getObject() メソッドは、 SQLType が SMALLINT の時 Integer オブジェクトを戻すようになりました。 以前それらのメソッドは、Short オブジェクトを戻していました。 readObject を使用して SMALLINT 列を読み取る場合、新しいタイプの戻りオブジェクトに対応するために、 Java アプリケーションを変更する必要があります。

データ切り捨てエラーを投じる際のエラー報告が変更され、 結果として警告が出されますがアプリケーションは失敗に終わりません。

IBM Toolbox for Java のこのリリースを使用して、 V5R1 より前のリリースでシリアル化したオブジェクトを非シリアル化することはできません。

Secure Sockets Layer (SSL) を使用して、 クライアントとサーバーの間のデータ・フローを暗号化する場合、以下のいずれかを使用する必要があります。

- Java Secure Socket Extension (JSSE)
- IBM iSeries Client Encryption ライセンス・プログラム 5722-CE2 または 5722-CE3 の V5R1 あるいはそれ以降のバージョンで配布された SSL オブジェクト。 IBM Toolbox for Java のこのリリースは、 V4R5 およびそれ以前のバージョンの iSeries Client Encryption と共には作動しません。

IBM Toolbox for Java は以下のサポートを引き続き提供します。

- Swing 1.1。これは、GUI クラスまたは Graphical Toolbox を使用するために必要です。
- Java 2 Platform, Standard Edition (J2SE)。 Java Development Kit 1.1.8 も引き続きサポートされます。

IBM Toolbox for Java を実行するための OS/400 要件に関連した互換性情報も検討する必要があります。

2002 年 9 月 26 日時点での新機能

IBM Toolbox for Java クラス・ファインダー
簡単かつすみやかに、名前と記述によって新しいクラスを検索したり、パッケージ別にクラスを表示したり、 Java クラス用のすべての IBM Toolbox のリストをアルファベット順に調べることができます。 各クラスの概説部分からは、さらに多くの特定の情報にリンクすることができます。

新規情報や改訂情報を参照する方法

技術的な改訂箇所には、以下のマークを使います (javadoc 以外)。

- » 新規または変更情報の先頭のマークです。
- « 新規または変更情報の終了のマークです。

新機能あるいはこのリリースでの変更内容についてのその他の情報を見つけるには、 プログラム資料説明書 を参照してください。

トピックの印刷

PDF 版をダウンロードし、表示するには、 IBM Toolbox for Java (約 5.08 MB または 869 ページ) を選択します。

注: IBM Toolbox for Java には、 PDF ファイルには含まれていない情報がいくつか含まれています。

PDF ファイルの保存

表示用または印刷用の PDF ファイルをワークステーションに保存するには、次のようにします。

1. ブラウザーで PDF を右マウス・ボタン・クリックする (上記のリンクを右マウス・ボタン・クリックする)。
2. 「リンクを名前を付けて保存」 (Netscape Navigator) または 「対象をファイルに保存」 (Internet Explorer) をクリックする。
3. PDF を保存したいディレクトリーに進む。
4. 「保存」をクリックする。

Adobe Acrobat Reader のダウンロード

これらの PDF を表示または印刷するのに Adobe Acrobat Reader が必要な場合、Adobe Web サイトからダウンロードできます。
(www.adobe.com/products/acrobat/readstep.html) 

表示用または印刷用の PDF ファイルをワークステーションに保存するには、次のようにします。

1. ブラウザーで PDF を開く (上記のリンクをクリックする)。
2. ブラウザーのメニューから「ファイル」をクリックする。
3. 「リンクを名前を付けて保存」 (Netscape Navigator) または 「対象をファイルに保存」 (Internet Explorer) をクリックする。
4. PDF を保存したいディレクトリーに進む。
5. 「保存」をクリックする。

ZIP 圧縮されたパッケージの IBM Toolbox for Java 情報をダウンロードする

javadocs を含む IBM Toolbox for Java トピックの ZIP 圧縮されたパッケージは、IBM Toolbox for Java and JTOpen Web サイト でダウンロードできます。

注: IBM Toolbox for Java は、 ZIP 圧縮されたパッケージに入っていない資料に

リンクしていることがあります。 そのようなリンクは、 ワークステーションにダウンロードしたファイルでは使えません。

IBM Toolbox for Java: はじめに

IBM Toolbox for Java を使用すると、 iSeries のリソース、データ、およびプログラムにアクセスするクライアントの、 Java アプレット、サーブレット、およびアプリケーションを簡単に書くことができます。

以下の情報は、IBM Toolbox for Java をインストールし、使い始める上で役立ちます。

» インストールの管理
Toolbox for Java インストールのインストール方法および管理の仕方の違いが、 どのように管理の利便性やパフォーマンスの向上に影響するかを示します。 «

Toolbox for Java のインストール
Toolbox for Java をクライアント/サーバー環境にインストールするための、 OS/400 要件およびワークステーション要件についてお読みください。 Toolbox for Java をiSeries サーバーおよび iSeries ワークステーションにインストールする方法を学習します。

システム・プロパティー
システム・プロパティーについて読み、 IBM Toolbox for Java のさまざまな局面を構成する上でそれらを使用する方法について学びます。

» 簡単なプログラミング例
Toolbox for Java の使用を開始します。 初めて Toolbox for Java でプログラム開発を検討されている方のために、 簡単なプログラミング例を紹介しています。 この例では、iSeries サーバー上で使用可能なデータおよびサービスを扱うために、 どのように Toolbox for Java を使い始めたらよいかを示しています。 «

IBM Toolbox for Java インストールの管理

IBM Toolbox for Java は、それを使用するクライアント・システム上、 あるいはクライアントがそれにアクセスできるネットワーク上の場所にインストールする必要があります。 クライアントになり得るのは、パーソナル・コンピューター、専用ワークステーション、 あるいは iSeries システムです。 iSeries サーバーあるいは iSeries サーバーの区画をクライアントとして構成できる、 という点を覚えておくことは重要です。 後者のケースでは、Toolbox for Java をサーバーのクライアント区画にインストールする必要があります。

Toolbox for Java をインストールし、管理するために、 以下の方式 (単独であっても組み合わせても) のいずれをも用いることができます。

- 個別管理によるインストールおよび各クライアント上の Toolbox for Java の個別管理
- AS400ToolboxInstaller を使用して各クライアントに Toolbox for Java をインストールし、 管理する、クライアント・インストールのネットワーク管理
- ネットワークを使用してサーバー上の Toolbox for Java の単一共有インストールをインストールし、管理する、 単一インストールのネットワーク管理

以下のセクションでは、各方式がパフォーマンスと管理の容易性の両方に与える影響について短く説明します。 ユーザーが Java アプリケーションを開発する方法および管理する方法をどのように選択するかが、 どの方式 (あるいは複数の方式の組み合わせ) を用いるかの決定要素となります。

個別管理

個別のクライアント上の Toolbox for Java インストールを個別に管理することを選択することができます。 個別のクライアント上に Toolbox for Java をインストールすることの主な利点は、 Toolbox for Java のクラスを使用するアプリケーションをクライアントが開始するために費やす時間を削減できるという点です。

主な欠点は、個別にそれらのインストールを管理するという点です。 ユーザーあるいは作成したアプリケーションのいずれかが、 個々のワークステーションにどのバージョンの Toolbox for Java をインストールしてあるかを追跡し、 管理する必要があります。

クライアント・インストールのネットワーク管理

Toolbox for Java のクライアント・インストールを管理するために、 ネットワークおよび AS400ToolboxInstaller の使用を選択することができます。 各クライアントには Toolbox for Java のコピーがあるため、この方式によるインストールには、 クライアントが Toolbox for Java アプリケーションを開始するために費やす時間を削減する点で、同様の利点があります。 この方式ではさらに、Toolbox for Java のすべての個々のインストールを自動的に更新することが可能になります。

この主な欠点は、個々のインストールを管理するために、AS400ToolboxInstaller を使用するプロセスを作成し、保守するという点です。

単一インストールのネットワーク管理

さらに、ネットワークを使用して、 すべてのクライアントがアクセスできるサーバー上に Toolbox for Java の単一コピーをインストールし、 管理することもできます。 この方式によるネットワーク・インストールには、以下の利点があります。

- すべてのクライアントが、同じバージョンの IBM Toolbox for Java を使用する
- 単一の Toolbox for Java のインストールを更新することにより、 すべてのクライアントが更新版を利用できる
- 個別のクライアントには、同一の初期 CLASSPATH を設定すること以外には、保守の必要はない

この方式によるインストールには、 クライアントが Toolbox for Java アプリケーションを開始するために費やす時間が増大するという欠点があります。 さらに、クライアントの CLASSPATH を使用可能にして、サーバーを指すようにする必要もあります。 OS/400 に組み込まれた iSeries ネットサーバー、 あるいは iSeries Access for Windows のように、 iSeries ネットサーバー上のファイルへのアクセスを可能にするさまざまな方式を用いることができます。

IBM Toolbox for Java のインストール

IBM Toolbox for Java をインストールする方法は、 インストールの管理をどのように行いたいかに依存しています。 インストールの管理の方法を決定した後、環境が以下の要件を満たしているかを確認してください。

- OS/400 の要件
- ワークステーション要件

Toolbox for Java のインストール

インストールの管理の方法を決定し、IBM Toolbox for Java を実行するための要件を確認した後、 Toolbox for Java をインストールすることができます。

- Toolbox for Java をサーバーにインストールする
- Toolbox for Java をワークステーションにインストールする

IBM Toolbox for Java のための OS/400 の要件

インストールの管理の方法を決定した後、 OS/400 の環境が以下の要件を満たしているかを確認してください。

- 必要な OS/400 オプション
- 他のライセンス・プログラムとの依存関係
- さまざまなレベルの OS/400 との互換性
- OS/400 JVM における実行時のネイティブ最適化
- » ToolboxME for iSeries アプリケーションを実行するための要件 «

注: Toolbox for Java を使用する前に、 必ずユーザーの環境に関係するワークステーション要件を確認してください。

必要な OS/400 オプション

IBM Toolbox for Java をクライアント/サーバー環境で実行するには、 QUSER ユーザー・プロファイルを使用可能にし、ホスト・サーバーを開始して、 TCP/IP サーバーを開始しておく必要があります。

- ホスト・サーバーを開始するために QUSER ユーザー・プロファイルを使用可能にする必要があります。
- ホスト・サーバーは、クライアントからの接続要求を listen したり、受諾したりします。 OS/400 ホスト・サーバー・オプション (ライセンス製品 5722SS1) は、 OS/400 の基本オプションとして組み込まれています。 詳細については、ホスト・サーバーの管理を参照してください。
- OS/400 に組み込まれている TCP/IP サポートを使用すると、 サーバーをネットワークに接続することが可能になります。 詳細については、TCP/IP を参照してください。

必要な OS/400 オプションの開始

iSeries コマンド行から、以下の手順に従って、必要な OS/400 オプションを開始してください。

1. QUSER プロファイルが使用可能になっていることを確認する。
2. OS/400 ホスト・サーバーを開始するために、CL の「ホスト・サーバーの開始」コマンドを使用する。 STRHOSTSVR *ALL と入力し、ENTER を押す。
3. TCP/IP 分散データ管理 (DDM) サーバーを開始するために、 CL の「TCP/IP サーバーの開始」コマンドを使用する。 STRTCPSVR SERVER(*DDM) と入力し、ENTER を押す。

IBM Toolbox for Java がサーバーにインストール済みかどうかを判別する

多くの iSeries サーバーは、 IBM Toolbox for Java のライセンス製品がすでにインストールされた状態で出荷されています。

Toolbox for Java がすでにインストール済みかどうかを調べるための手順は以下のとおりです。

- iSeries ナビゲーターで、使用したいシステムを選択し、サインオンします。
- 「機能ツリー」(左側のペイン) でシステムを展開した後、 「構成およびサービス」を展開します。
- 「ソフトウェア」を展開した後、 「インストール済みプロダクト」を展開します。
- 「詳細」ペイン (右側のペイン) で、 「プロダクト」欄の「5722jc1」を探します。 このプロダクトが見つかった場合、 IBM Toolbox for Java のライセンス・プログラム は選択したサーバーにインストール済みです。

  注: Toolbox for Java がインストール済みかどうかを調べるために、 CL の「メニュー (Go To Menu)」コマンド (GO MENU(LICPGM)) のオプション 11 を使用することもできます。

Toolbox for Java がインストール済みでない場合は、 IBM Toolbox for Java ライセンス製品をインストールすることができます。

以前のバージョンの Toolbox for Java がインストール済みの場合は、 まず現在インストールされているバージョンを削除し、 その後 IBM Toolbox for Java ライセンス製品をインストールします。 起こりうる問題を避けるために、インストールされた現行バージョンの Toolbox for Java を削除する前に、 そのバックアップを取ることを考慮してください。

QUSER プロファイルの検査

OS/400 ホスト・サーバーは QUSER ユーザー・プロファイルの下で開始するため、 まず QUSER プロファイルが使用可能になっていることを確認する必要があります。

QUSER プロファイルを検査する

コマンド行を使用して QUSER プロファイルを検査するには、以下のステップをすべて行ってください。

1. iSeries コマンド行で、DSPUSRPRF USRPRF(QUSER) と入力し、Enter を押します。
2. 「状況」 が *ENABLED になっていることを確認します。 プロファイル状況が *ENABLED になっていない場合は、QUSER プロファイルを変更します。

QUSER ユーザー・プロファイルの変更

QUSER プロファイルが *ENABLED ではない場合、 それを使用可能にして OS/400 ホスト・サーバーを始動することが必要です。 また、QUSER プロファイル・パスワードは *NONE にはできません。 *NONE になっているなら、リセットする必要があります。

コマンド行を使用して QUSER プロファイルを使用可能にするには、以下のステップを完了します。

1. CHGUSRPRF USRPRF(QUSER) と入力する。
2. 「状況」フィールドを *ENABLED に変更し、 「ENTER」 を押す。

これで QUSER ユーザー・プロファイルは、OS/400 ホスト・サーバーを始動する準備が整いました。

他のライセンス・プログラムとの依存関係

IBM Toolbox for Java をどのように使用したいかにより、 他のライセンス・プログラムをインストールする必要が生じるかもしれません。 以下の情報はそれらの依存関係を説明しています。

スプール・ファイル・ビューアー

IBM Toolbox for Java のスプール・ファイル・ビューアー機能 (SpooledFileViewer クラス) を使用する場合は、 サーバーにホスト・オプション 8 (AFP 互換フォント) がインストールされていることを確認してください。

注: SpooledFileViewer、PrintObjectPageInputStream、 および PrintObjectTransformedInputStream クラスは、 V4R4 以降のシステムに接続している場合にのみ機能します。

Secure Sockets Layer

Secure Sockets Layer (SSL) を使用する場合は、 次のものがインストールされていることを確認してください。

- IBM HTTP Server for iSeries ライセンス・プログラム (5722-DG1)
- OS/400 オプション 34 (ディジタル証明書マネージャー)
- IBM Cryptographic Access Provider 128-bit for iSeries (5722-AC3)
- iSeries Client Encryption (128-bit) (5722-CE3)

IBM Toolbox for Java の V5R2 バージョンを使用する場合、 » iSeries Client Encryption の V5R1 あるいは V5R2 バージョンのいずれか « を使用しなければなりません。

SSL についての詳細は、Secure Sockets Layer および Java Secure Socket Extension を参照してください。

アプレット、サーブレット、SSL、 あるいは AS400ToolboxInstaller を使用するための HTTP サーバー

iSeries システムでアプレット、サーブレット、SSL、または AS400ToolboxInstaller クラスを使用する場合は、 iSeries システムで HTTP サーバーをセットアップし、クラス・ファイルをインストールしなければなりません。 IBM HTTP Server についての詳細については、 以下の URL にある IBM HTTP Server for AS/400 Webmaster's Guide (GC41-5434) を参照してください。 http://www.ibm.com/eserver/iseries/products/http/docs/doc.htm 。 Webmaster's

Guide は、HTML と PDF の両方の形式で入手可能です。

ディジタル証明書マネージャーについて、 および IBM HTTP Server を使用してディジタル証明書を作成および処理する方法については、ディジタル証明書管理を参照してください。

さまざまなレベルの OS/400 との互換性

IBM Toolbox for Java は、サーバーとクライアントの両方で実行できるため、互換性の問題は、サーバーでの実行と、クライアントからサーバーへの接続の両方に影響します。

IBM Toolbox for Java、バージョン 5 リリース 2 をサーバーで実行する場合

IBM Toolbox for Java (ライセンス・プログラム 5722-JC1 V5R2M0) を iSeries システムにインストールするには、 サーバーで次のいずれかが実行されていなければなりません。

- OS/400 バージョン 5 リリース 2
- OS/400 バージョン 5 リリース 1
- OS/400 バージョン 4 リリース 5

システムでインストールできるのは、IBM Toolbox for Java ライセンス・プログラムの 1 つのバージョンだけです。 別のバージョンをインストールするには、まず既存の IBM Toolbox for Java ライセンス・プログラムを除去してください。

IBM Toolbox for Java を使用してクライアントからサーバーに接続する場合

クライアントと、接続先となるサーバーで、IBM Toolbox for Java の別々のバージョンを使用することができます。 IBM Toolbox for Java のバージョン 5 リリース 2 を使用して、 iSeries システム上のデータおよびリソースにアクセスするには、 接続先となるサーバーで次のいずれかを実行しておかなければなりません。

- OS/400 バージョン 5 リリース 2
- OS/400 バージョン 5 リリース 1
- OS/400 バージョン 4 リリース 5

以下の表では、 OS/400 の各バージョンへの IBM Toolbox for Java のインストールおよび接続に関する互換性の要件を示しています。

Toolbox のモディフィケーション	OS/400 と一緒に出荷	LPP	OS/400 上にインストール	OS/400 への接続
Mod 0	V4R2	5763-JC1 V3R2M0	V3R2 以上	V3R2 以上

Mod 1	V4R3	5763-JC1 V3R2M1	V3R2 以上	V3R2 以上
Mod 2	V4R4	5769-JC1 V4R2M0	V4R2 以上	V4R2 以上
Mod 3	V4R5	5769-JC1 V4R5M0	V4R3 以上	V4R2 以上
Mod 4	V5R1	5722-JC1 V5R1M0	V4R4 以上	V4R3 以上
»Mod 5	V5R2	5722-JC1 V5R2M0	V4R5 以上	V4R5 以上«

iSeries JVM における実行時のネイティブ最適化

ネイティブ最適化は、IBM Toolbox for Java クラスが、 OS/400 での実行時にユーザーが予期したとおりに働くようにするための機能のセットです。 最適化は、iSeries JVM 上で実行される時にのみ、IBM Toolbox for Java の操作に影響します。

Java プログラムでネイティブ最適化が使用されるのは、 サーバー上の OS/400 のバージョンに適合する IBM Toolbox for Java のバージョンを使用する場合だけです。 最適化は次のとおりです。

- サインオン: AS400 オブジェクトでユーザー ID またはパスワードが指定されていない場合、 現行ジョブのユーザー ID およびパスワードが使用されます。
- OS/400 API の直接呼び出し (ホスト・サーバーへのソケット呼び出しの代わり):
  - セキュリティー要件が満たされた場合、レコード・レベルのデータベース・アクセス、 データ待ち行列、およびユーザー・スペース。
  - セキュリティー要件およびスレッド・セーフ要件が満たされた場合、 プログラム呼び出しおよびコマンド呼び出し。

»注: 最高のパフォーマンスを得られるようにするため、 Java プログラムおよびデータベース・ファイルが同じ iSeries システム上にある時にネイティブ・ドライバーを使用するように JDBC ドライバーのプロパティーを設定してください。«

最適化のために Java アプリケーションに変更を加える必要はありません。 IBM Toolbox for Java は、必要と判断された場合のみ、最適化を自動的に使用可能にします。

ネイティブ最適化の互換性のための要件

以下の表では、 ネイティブ最適化を使用するために実行しなければならない IBM Toolbox for Java と OS/400 のバージョンを示します。 この表では、ネイティブ最適化に影響を与える互換性だけを示します。 一般的な互換性の問題に関しては、さまざまなレベルの OS/400 との互換性を参照してください。

OS/400 のレベル	ネイティブ最適化を使用するために必要な Toolbox モディフィケーション
V4R2	Mod0 拡張機能は使用できません。
V4R3	Mod1 Mod2

V4R4	Mod1	Mod2			
V4R5			Mod3		
V5R1				Mod4	
»V5R2					Mod5«

パフォーマンスを向上させるためには、 OS/400 のネイティブ最適化を含む JAR ファイルを使用するようにしなければなりません。 詳細については、JAR ファイルの注 1 を参照してください。

IBM Toolbox for Java と OS/400 のバージョンが適合しない場合、 ネイティブ最適化は使用できません。 しかしそのような場合でも、IBM Toolbox for Java は、クライアント上で実行されているかのように機能します。

ToolboxME for iSeries の要件

ワークステーション、ワイヤレス・デバイス、およびサーバーは、 ToolboxME for iSeries アプリケーションの開発および実行に関する特定の要件 (以下にリストされている) を満たしている必要があります。 IBM Toolbox for Java 2 Micro Edition は IBM Toolbox for Java の一部と見なされますが、 ライセンス交付を受けた製品には含まれません。

ToolboxME for iSeries (jt400Micro.jar) は、JTOpen と呼ばれる、 Toolbox for Java のオープン・ソース・バージョンに含まれています。 別個に、JTOpen に組み込まれている ToolboxME for iSeries をダウンロードしてセットアップする必要があります。

要件

ToolboxME for iSeries を使用するには、ワークステーション、 Tier0 ワイヤレス・デバイス、 およびサーバーは以下の要件を満たしている必要があります。

ワークステーション要件

ToolboxME for iSeries アプリケーションを開発するためのワークステーション要件は以下のとおりです。

- Java 2 Platform、Standard Edition、バージョン 1.3 以降
- ワイヤレス・デバイス用の Java 仮想マシン
- ワイヤレス・デバイス・シミュレーターまたはエミュレーター

ワイヤレス・デバイス要件

Tier0 デバイスで ToolboxME for iSeries アプリケーションを実行するための唯一の要件は、 ワイヤレス・デバイス用の Java 仮想マシンを使用することです。

サーバー要件

ToolboxME for iSeries アプリケーションを使用するためのサーバー要件は以下のとおりです。

- MEServer。 IBM Toolbox for Java または最新バージョンの JTOpen に含まれています。
- IBM Toolbox for Java のための OS/400 の要件

IBM Toolbox for Java のワークステーション要件

インストールの管理の方法を決定した後、 ワークステーションが以下の要件を満たしているかを確認してください。

- Java アプリケーションを実行するための要件
- Java アプレットを実行するための要件
- ToolboxME for iSeries アプリケーションを開発するための要件
- Swing の要件

注: Toolbox for Java を使用する前に、 必ずユーザーの環境に関係する OS/400 の要件を確認してください。

Toolbox for Java アプリケーションを実行するためのワークステーション要件

Toolbox for Java アプリケーションを開発し、実行するために、 ワークステーションが以下の要件を満たしているかを確認してください。

- Java 2 Platform、Standard Edition (J2SE(TM)) または Java 2 Platform、Enterprise Edition (J2EE(TM)) のバージョン 1.3.x 以上をサポートする Java 仮想マシン (JVM) を推奨します。 Toolbox for Java の多くの新しい機能は、このバージョンの JVM を使用することが必須となっています。 しかし、JDK 1.1.8 またはそれ以降のいずれかの JDK (Java 2 Platform を含む) を完全にサポートする JVM もまだ使用できます。
- プログラムで Graphical Toolbox、または vaccess パッケージ内のクラスを使用する場合は、 Swing 1.1 も必要です。 以下の環境がテスト済みです。
  - »Windows(R)2000«
  - »Windows(R)XP«
  - AIX バージョン 4.3.3.1
  - Sun Solaris(TM) バージョン 5.7
  - » OS/400 バージョン 4 リリース 5 あるいはそれ以降«
  - »Linux (Red Hat 7.0)«
- TCP/IP がインストールされ、構成されていること

IBM Toolbox for Java アプレットを実行するためのワークステーション要件

Toolbox for Java アプリケーションを開発し、実行するために、 ワークステーションが以下の要件を満たしているかを確認してください。

- 互換性のある Java 仮想マシン (JVM) を持つブラウザー。 以下の環境がテスト済みです。
  - » Netscape Communicator 4.7 (Java 1.3 あるいはそれ以降のプラグインを使用するもの)

  注: IBM Toolbox for Java は、 Netscape Navigator あるいは Microsoft Internet Explorer のデフォルトの JVM での実行は、 もはやサポートしていません。 Toolbox for Java のクラスを使用してブラウザーで実行されるアプレットのために、 Sun Java 2 Runtime Environment (JRE) 1.3.0 プラグイン のようなプラグインをインストールする必要があります。 «
- TCP/IP がインストールされ、構成されていること
- » OS/400 V4R5 以上のサーバーにワークステーションが接続されること «

IBM Toolbox for Java のワークステーション Swing 要件

IBM Toolbox for Java は、V4R5 で、Swing 1.1 をサポートするようになり、 このリリースにおいても、そのサポートは継続されます。 Swing への切り替えには、IBM Toolbox for Java クラスでのプログラミング変更が必要となっていました。 このため、プログラムで V4R5 より前のリリースからの Graphical Toolbox または Vaccess クラスを使用している場合には、 プログラムを変更することも必要です。

プログラミング変更に加えて、プログラムの実行時には、 Swing クラスを CLASSPATH に入れておかなければなりません。 Swing クラスは、Java 2 Platform の一部です。 Java 2 Platform をお持ちでない場合は、 Sun Microsystems, Inc から Swing 1.1 クラスをダウンロードすることができます。

IBM Toolbox for Java を iSeries サーバーにインストールする

iSeries サーバーあるいはそのサーバーの区画をクライアントとして構成してある時にのみ、 IBM Toolbox for Java をそのサーバー上にインストールする必要があります。

注: ネイティブ・バージョンの Toolbox for Java が OS/400 と共に出荷されます。ですから、Toolbox for Java を iSeries サーバー上のみで使用したい場合、 ライセンス製品をインストールする必要はありません。 ネイティブ・バージョンの Toolbox for Java についての詳細については、JAR ファイル: 注 1 を参照してください。

IBM Toolbox for Java をインストール前に、ご使用の OS/400 のバージョンが、Toolbox for Java を実行するための要件を満たしていることを確認する必要があります。 さらにサーバーの中には、Toolbox for Java のインストールがすでに構成された状態で出荷されるものもあります。 サーバーに Toolbox for Java ライセンス製品がすでにインストール済みかどうかを判別するとよいでしょう。

Toolbox for Java のインストール

IBM Toolbox for Java のライセンス・プログラムを、 iSeries ナビゲーターあるいはコマンド行のいずれかによってインストールできます。

iSeries ナビゲーターを使用した Toolbox for Java のインストール

iSeries ナビゲーターを使用して Toolbox for Java をインストールするための手順は以下のとおりです。

1. iSeries ナビゲーターで、使用したいシステムにサインオンします。
2. 「機能ツリー」(左側のペイン) で、「ユーザー接続」を展開します。
3. 「ユーザー接続」の中で、 Toolbox for Java をインストールしたいシステムを右マウス・ボタン・クリックします。
4. 「実行 (Run)」コマンドを選択します。
5. 「ライセンス・プログラム復元 (RSTLICPGM)」ダイアログで、 以下の情報を入力した後、「OK」をクリックします。
   - プロダクト: 5722JC1
   - デバイス: デバイス名あるいは保管ファイル

   注: 詳細については、 「ライセンス・プログラム復元 (RSTLICPGM)」ダイアログにある「ヘルプ」をクリックしてください。

iSeries ナビゲーターを使用して以下の手順を行うことにより、 発生したマネージメント・セントラル・コマンド・タスクの状況を表示することができます。

1. 「マネージメント・セントラル」を展開します。
2. 「タスク活動」を展開します。
3. 「タスク活動」の下にある「コマンド」を選択します。
4. 「詳細」ペインで、当てはまる「コマンドの実行」タスクをクリックします。

コマンド行を使用した Toolbox for Java のインストール

iSeries コマンド行から Toolbox for Java をインストールするための手順は以下のとおりです。

1. iSeries コマンド行で、CL の「メニュー (Go to Menu)」コマンドを使用します。 「GO MENU(LICPGM)」と入力し、ENTER を押します。
2. 11 (ライセンス・プログラムのインストール) を選択します。
3. 「5722-JC1 IBM Toolbox for Java」 を選択します。

ライセンス・プログラムのインストールについての詳細は、ソフトウェアとライセンス・プログラムの管理を参照してください。

IBM Toolbox for Java をワークステーションにインストールする

Toolbox for Java をインストールする前に、 必ずユーザーの環境に関係するワークステーション要件を確認してください。 IBM Toolbox for Java をワークステーションにインストールする方法は、 インストールの管理をどのように行いたいかに依存しています。

- Toolbox for Java を個別のクライアントにインストールするには、 JAR ファイルをワークステーションにコピーし、ワークステーションの CLASSPATH を構成します。
- サーバーにインストールされた Toolbox for Java を使用するには、 ワークステーションの CLASSPATH を構成して、サーバー・インストールを指すようにするだけで十分です。 ワークステーションの CLASSPATH がサーバーを指すようにするには、 サーバーに iSeries ネットサーバーがインストール済みである必要があります。

この資料は、クラス・ファイルをワークステーションにコピーする方法を説明しています。 CLASSPATH をワークステーションに設定することについての詳細は、 ワークステーションのオペレーティング・システム資料、あるいは Sun 社の Java Web サイト で入手できる情報を参照してください。

注: Toolbox for Java のクラスをアプリケーションで使用する時には、 システムが OS/400 の要件を満たしていることが必要です。

Toolbox for Java のクラス・ファイルはいくつかの JAR ファイルにパックされているため、 これらの JAR ファイルの 1 つまたは複数をワークステーションにコピーする必要があります。 どの JAR ファイルが Toolbox for Java の特定の機能に必要となるかについての詳細は、JAR ファイルを参照してください。

例: jt400.jar のコピー

以下の例では、コア IBM Toolbox for Java クラスが入っている jt400.jar をコピーするものとします。

JAR ファイルを手動でコピーするには、以下の手順に従ってください。

1. 次のディレクトリー内で jt400.jar ファイルを見つけます。
   /QIBM/ProdData/HTTP/Public/jt400/lib
2. サーバーからワークステーションに jt400.jar をコピーします。 これを行うための方法はいくつかあります。
   - iSeries Access for Windows を使用して、 ワークステーション上の

ネットワーク・ドライブをサーバーにマップし、ファイルをコピーする。
- ファイル転送プロトコル (FTP) を使用して、ファイルを (バイナリー・モードで) ワークステーションに送信する。

3. ワークステーションの CLASSPATH 環境変数を更新します。
   - たとえば、Windows NT を使用していて、jt400.jar を C:\jt400\lib にコピーした場合、 CLASSPATH の終わりに次のストリングを追加します。

```
;C:/jt400/lib/jt400.jar
```

JTOpen と呼ばれる、Toolbox for Java のオープン・ソース版を使用するというオプションもあります。 JTOpen についての詳細は、 IBM Toolbox for Java および JTOpen の Web サイト を参照してください。

ブラウザーで Javascript が使用可能ではない場合、 注釈のイメージの隣にあるテキスト・リンクをクリックして適切な注釈にリンクします。 以下の表にすべての注釈をリストします。

JAR ファイル

IBM Toolbox for Java は JAR ファイルのセットとしてお手元に届きます。 各 JAR ファイルには、それぞれ別々の機能を備えた Java パッケージが入っています。 必要な機能を備えた JAR ファイルのみを使用すれば、 ストレージ・スペースの所要量を減らすことができます。

JAR ファイルを使用するには、そのエントリーが CLASSPATH に組み込まれていることを確認してください。

以下の図は、 一覧中のパッケージ内のクラスを使用するのにどの JAR ファイルが CLASSPATH 内になければならないかを示しています。

表のエントリーの中には、詳細を提供する注釈があるものもあります。 ブラウザーで Javascript が使用可能な場合、 イメージをクリックし、別のウィンドウの情報を表示します。 使用不可の場合には、テキスト・リンクをクリックして、 以下の表にリストされている同じ情報にリンクできます。

Toolbox for Java パッケージまたは機能	CLASSPATH に置く必要のある JAR ファイル
アクセス・クラス	通常のクライアント/サーバー環境では jt400.jar (クライアント) または jt400Native.jar (サーバー) 注 1、あるいは Proxy 環境では jt400Proxy.jar
»CommandPrompter 注 2	jt400.jar、jui400.jar、util400.jar 注 3、および x4j400.jar «
HTML クラス	jt400.jar 注 1 と jt400Servlet.jar (クライアント)、または jt400Native.jar (サーバー) 注 1
JDBC データ・ソース GUI 注 4	jt400.jar (クライアント) 注 1 および jui400.jar
NLS のシステム・メッセージとエラー・メッセージ	jt400Mri_lang_cntry.jar 注 5
PCML (開発) 注 6	jt400.jar (クライアント) または jt400Native.jar (サーバー) 注 1、 注 7、および x4j400.jar
PCML (ランタイム、逐次化されたもの)	jt400.jar (クライアント) または jt400Native.jar (サーバー) 注 1、 注 7

PDML (開発) 注 2	uitools.jar、jui400.jar、util400.jar 注 3、および x4j400.jar
PDML (ランタイム、構文解析済みのもの) 注 2	jui400.jar、util400.jar 注 3、および x4j400.jar
PDML (ランタイム、逐次化されたもの) 注 2	jui400.jar、および util400.jar 注 3
ReportWriter クラス	jt400.jar (クライアント) または jt400Native.jar (サーバー) 注 1 および reportwriter jars 注 8
リソース・クラス	jt400.jar (クライアント) または jt400Native.jar (サーバー) 注 1
» RFML	jt400.jar (クライアント) または jt400Native.jar (サーバー) 注 1、および x4j400.jar «
セキュリティー・クラス	通常のクライアント/サーバー環境では jt400.jar (クライアント) または jt400Native.jar (サーバー) 注 1、あるいは Proxy 環境では jt400Proxy.jar
サーブレット・クラス	jt400.jar 注 1 と jt400Servlet.jar (クライアント)、または jt400Native.jar (サーバー) 注 1
» iSeries システム・デバッガー 注 2	jt400.jar (クライアント) 注 1 および tes.jar «
» ToolboxME for iSeries	jt400Micro.jar (クライアント) および jt400.jar (サーバー) または jt400Native.jar (サーバー) 注 9 «
Vaccess クラス	jt400.jar (クライアント) 注 1

注 1: 次のように、 一部の IBM Toolbox for Java クラスは複数の JAR ファイルに入っています。

- » jt400.jar - Access、resource、vaccess、security、PCML、RFML、 JDBC サポートおよび MEServer。 «
- jt400.zip - jt400.zip に代えて jt400.jar を使用してください。 jt400.zip は、旧リリースの IBM Toolbox for Java との互換性を保つために用意されています。
- jt400Access.zip - Access、resource、security、PCML。 つまり、jt400.jar に入っているものと同じクラスから vaccess クラスを除いたもの。 jt400Access.zip は、旧リリースの IBM Toolbox for Java との互換性を保つ

ために用意されています。 jt400Access.zip に代えて jt400.jar または jt400Native.jar を使用してください。

- » jt400Native.jar - Access、resource、security、PCML、HTML、RFML、MEServer、およびネイティブ最適化。 « ネイティブ最適化は、iSeries JVM での実行時に iSeries 機能を利用する一連のクラス (20 個未満) です。 jt400Native.jar にはネイティブ最適化が備えられているため、 iSeries JVM での実行時には jt400.jar ではなく jt400Native.jar を使用します。 jt400Native.jar はご購入の OS/400 に添付されており、ディレクトリー /QIBM/ProdData/OS400/jt400/lib に置かれています。
- jt400Native11x.jar - ネイティブ最適化のみ。 iSeries JVM での実行中に jt400.jar を使用したい場合、 jt400Native.jar ではなく jt400Native11x.jar を CLASSPATH に入れます。 jt400Native11x.jar はご購入の OS/400 に添付されており、ディレクトリー /QIBM/ProdData/OS400/jt400/lib に置かれています。

» 注 2: CommandPrompter、PDML、または iSeries システム・デバッガーを使用するには、 Toolbox for Java の一部ではない追加の JAR ファイル、jhall.jar を必要とします。 jhall.jar のダウンロードの詳細は、 Sun JavaHelp (TM) の Web サイトを参照してください。 «

注 3: util400.jar には、 入力を形式設定するためとコマンド行 (CL) プロンプターを使うための iSeries 固有のクラスが含まれています。 CommandPrompter クラスを使用するには、util400.jar が必要です。 PDML を使用するのに util400.jar は必須ではありませんが、あるほうが便利です。

注 4: jui400.jar には、 JDBC DataSource GUI インターフェースを使用するのに必要なクラスが入っています。 jt400.jar (注 1) には、他のすべて JDBC 機能に必要なクラスが入っています。

注 5: jt400Mri_xx_yy.jar には変換済みのメッセージが入っています。 それには、例外メッセージ、ダイアログ、およびその他の通常処理からの出力に含まれるストリングもあります。 jt400Mri_lang_cntry.jar では lang は ISO 言語コードであり、cntry は、 その中に入っているテキストの変換に使われる ISO 国および地域別コードです。 場合によっては ISO 国および地域別コードは使われません。 IBM Toolbox for Java ライセンス・プログラムの特定の各国語バージョンを iSeries にインストールすると、 該当する jt400Mri_lang_cntry.jar ファイルがインストールされます。 サポートされていない言語をインストールした場合のデフォルトは英語バージョンですが、 それは IBM Toolbox for Java の JAR ファイルに入っています。

- たとえば、ライセンス・プログラム 5722-JC1 のドイツ語バージョンをインストールすると、 ドイツ語の JAR ファイル jt400Mri_de.jar がインストールされます。

このような JAR ファイルを複数個 CLASSPATH に追加すれば、他の言語のサポートを追加することができます。 Java では、現在のロケールを基に正しいストリングがロードされます。

注 6: 開発時に PCML ファイルをシリアル化すると、 次のような利点があります。

1. 実行時ではなく開発時にのみ PCML ファイルを構文解析すれば済む。
2. ユーザーがアプリケーションを実行するための CLASSPATH 内の JAR ファイルが少なくて済む。

開発時に PCML ファイルを構文解析するには、data.jar または jt400.jar 内の PCML ランタイムと、 x4j400.jar 内の PCML パーサーの両方が必要です。 ユーザーが逐次化されたアプリケーションを実行するためには、jt400.jar だけが必要です。 詳細については、PCML を使用して iSeries プログラム呼び出しを作成するを参照してください。

注 7: data400.jar に代えて、 jt400.jar および jt400Native.jar を使用してください。 data400.jar には PCML ランタイム・クラスも入っていますが、これは現在、jt400.jar および jt400Native.jar にも入っています (注 1)。 data400.jar は、旧リリースの IBM Toolbox for Java との互換性を保つために用意されています。

注 8: ReportWriter クラスのコピーは複数の JAR ファイルに入っています。

- composer.jar
- outputwriter.jar
- reportwriters.jar
- xslparser.jar
- x4j400.jar

アプリケーションが PCL データを iSeries スプール・ファイルにストリームする場合、 適切な JAR ファイルを使用してアクセス・クラスを使用可能にする必要があります (注 1)。 スプール・ファイルを作成し PCL データを保持するには、AS400、 OutputQueue、PrintParameterList、および SpooledFileOutputStream クラスが必要です。 詳細については、ReportWriter クラスを参照してください。

» 注 9: jt400Micro.jar には、 jt400.jar および jt400Native.jar の両方に常駐する MEServer を実行するのに必要なクラスは含まれません (注 1)。 jt400Micro.jar は IBM Toolbox for Java and JTOpen Web サイトからのみ入手可能です。 «

システム・プロパティー

システム・プロパティーを指定して、IBM Toolbox for Java のさまざまな局面を構成することができます。 たとえば、システム・プロパティーを使って、Proxy サーバー、またはトレースのレベルを定義できます。 システム・プロパティーは、コードを再コンパイルしなくても、都合に合わせた実行時の構成を行うのに役立ちます。 システム・プロパティーは、実行時にシステム・プロパティーを変更する場合の環境変数のように作動し、変更は通常、次回アプリケーションを実行するまで反映されません。

システム・プロパティーの設定方法は、以下のとおりです。

- java.lang.System.setProperties() メソッドを使用する

  java.lang.System.setProperties() メソッドを使用すると、方針に基づいてシステム・プロパティーを設定できます。

  たとえば、次のコードは com.ibm.as400.access.AS400.proxyServer プロパティーを、hqoffice に設定します。

  ```
  Properties systemProperties = System.getProperties();
     systemProperties.put ("com.ibm.as400.access.AS400.proxyServer", "hqoffice");
     System.setProperties (systemProperties);
  ```

- java コマンドの -D オプションを使用する

  多くの環境では、アプリケーションをコマンド行から実行しているときに、java コマンドの -D オプションを使用して、システム・プロパティーを設定することができます。

  たとえば次のプログラムは、com.ibm.as400.access.AS400.proxyServer プロパティーを hqoffice に設定した、Inventory というアプリケーションを実行します。

  ```
  java -Dcom.ibm.as400.access.AS400.proxyServer=hqoffice Inventory
  ```

- jt400.properties ファイルを使用する

  環境によっては、すべてのユーザーに独自のシステム・プロパティーを設定するように指示するのが不都合な場合もあります。 代替方法として、com.ibm.as400.access パッケージの一部であるかのように検索される jt400.properties というファイルで、IBM Toolbox for Java システム・プロパティーを指定できます。 つまり、クラスパスが示す com/ibm/as400/access ディレクトリーに jt400.properties ファイルを入れるということです。

  たとえば、次の行を jt400.properties ファイルに挿入することによって、com.ibm.as400.access.AS400.proxyServer プロパティーを hqoffice に設定します。

  ```
  com.ibm.as400.access.AS400.proxyServer=hqoffice
  ```

  円記号 (\) は、プロパティー・ファイルでエスケープ文字として機能します。 文字通りの円記号を指定するには、2 つの円記号 (\\) を使用します。

  jt400,properties ファイルのこのサンプルを、ご使用の環境に合わせて変更してください。

- Properties クラスを使用する

  ブラウザーによっては、セキュリティー設定を明示的に変更しないと、プロパティー・ファイルをロードしません。 しかし、ほとんどのブラウザーは .class ファイルにあるプロパティーを許可するので、 IBM Toolbox for Java システム・プロパティーも

java.util.Properties を拡張した com.ibm.as400.access.Properties というクラスによって指定できます。

たとえば、com.ibm.as400.access.AS400.proxyServer プロパティーを hqoffice に設定するには、次の Java コードを使用します。

```
package com.ibm.as400.access;

public class Properties
extends java.util.Properties
{
public Properties ()
   {
      put ("com.ibm.as400.access.AS400.proxyServer", "hqoffice");
   }
}
```

Properties.java ソース・ファイルのこのサンプルを、ご使用の環境に合わせて変更およびコンパイルしてください。

IBM Toolbox for Java システム・プロパティーが上記で解説された複数のメカニズムを使用して設定される場合、 優先順位は (優先度の高いものから順に) 次のとおりです。

1. java.lang.System.setProperties() を使用した、方針に基づいて設定されるシステム・プロパティー
2. java コマンドの -D オプションを使用して設定されるシステム・プロパティー
3. Properties クラスを使用して設定されるシステム・プロパティー
4. jt400.properties ファイルを使用して設定されるシステム・プロパティー

IBM Toolbox for Java は、次のシステム・プロパティーをサポートします。

- Proxy サーバー・プロパティー
- トレース・プロパティー
- CommandCall/ProgramCall プロパティー

Proxy サーバー・プロパティー

Proxy サーバー・プロパティー	説明
com.ibm.as400.access.AS400.proxyServer	次の形式を使用して、Proxy サーバーのホスト名とポート番号を指定します。 hostName:portNumber ポート番号はオプションです。

com.ibm.as400.access.SecureAS400.proxyEncryptionMode	SSL を使用して暗号化される Proxy のデータ・フローの部分を指定します。 有効な値は次のとおりです。 ● 1 = Proxy クライアントから Proxy サーバー ● 2 = Proxy サーバーから iSeries ● 3 = Proxy クライアントから Proxy サーバー、および Proxy サーバーから iSeries
com.ibm.as400.access.TunnelProxyServer.clientCleanupInterval	Proxy サーバーがアイドル接続を検索する頻度を、秒単位で指定します。 Proxy サーバーはスレッドを開始して、もう通信を行っていないクライアントを探します。 このプロパティーを使用して、スレッドがアイドル接続を検索する頻度を設定します。
com.ibm.as400.access.TunnelProxyServer.clientLifetime	JVM がガーベッジ・コレクションを実行できるように、 Proxy サーバーがオブジェクトへの参照を除去する前にクライアントがアイドル状態になる時間を秒単位で指定します。 Proxy サーバーはスレッドを開始して、もう通信を行っていないクライアントを探します。 ガーベッジ・コレクションを実行する前にクライアントがアイドル状態になる時間を設定するには、このプロパティーを使用します。

トレース・プロパティー

トレース・プロパティー	説明
com.ibm.as400.access.Trace.category	使用可能にするトレース・カテゴリーを指定します。 これは、トレース・カテゴリーの任意の組み合わせを含む、コンマで区切ったリストです。 トレース・カテゴリーの完全なリストは、Trace クラスで定義されています。
com.ibm.as400.access.Trace.file	トレース出力が書き込まれるファイルを指定します。 デフォルトでは、トレース出力は System.out に書き込まれます。

»com.ibm.as400.access.ServerTrace.JDBC	JDBC サーバー・ジョブ上で開始するトレース・カテゴリーを指定します。 サポートされる値の詳細については、JDBC サーバー・トレースのプロパティーを参照してください。 «

CommandCall/ProgramCall プロパティー

CommandCall/ProgramCall プロパティー	説明
com.ibm.as400.access.CommandCall.threadSafe	CommandCalls がスレッド・セーフであることを想定するかどうかを指定します。 true の場合、すべての CommandCalls がスレッド・セーフであると想定されます。 false の場合、すべての CommandCalls がスレッド・セーフでないと想定されます。 オブジェクトに対して CommandCall.setThreadSafe(true/false) または AS400.setMustUseSockets(true) のいずれかが実行されている 場合、このプロパティーは、指定されている CommandCall オブジェクトについては無視されます。
com.ibm.as400.access.ProgramCall.threadSafe	ProgramCalls がスレッド・セーフであることを想定するかどうかを指定します。 true の場合、すべての ProgramCalls がスレッド・セーフであると想定されます。 false の場合、すべての ProgramCalls がスレッド・セーフでないと想定されます。 オブジェクトに対して ProgramCall.setThreadSafe(true/false) または AS400.setMustUseSockets(true) のいずれかが実行されている場合、 このプロパティーは、指定されている ProgramCall オブジェクトについては無視されます。

簡単なプログラミング例

以下の例は、IBM Toolbox for Java クラスを使って独自の Java プログラムをコーディングするいくつかの方法を示しています。 Toolbox for Java のクラスを初めて使用するプログラマーを対象としているため、 これらの例には、コード内のかぎとなる行についての詳細説明が含まれています。

以下の方法で、詳細説明を表示することができます。

- 詳細説明を表示するには、ポップアップ・ウィンドウの  イメージをクリックします。 (ポップアップ・ウィンドウ内の注記を参照するためには、ブラウザーが JavaScript をサポートしている必要があります。)
- テキスト・リンクをクリックして、例の最後にある詳細説明を参照します。

初めて開始する上で助けが必要な場合には、初めて Toolbox for Java プログラムを書くを参照してください。

Toolbox for Java 情報の中にある、その他多くの例へのリンクについては、 コード例を参照してください。

簡単なプログラミング例を表示するには、以下のリストを使用してください。

- コマンドを呼び出す
- メッセージ待ち行列を使用する
- レコード・レベルのアクセスを使用する
- JDBC クラスを使用して、テーブルの作成と記入を行う
- サーバー・ジョブのリストを GUI で表示する

以下の特記事項は、IBM Toolbox for Java のすべての例に適用されます。

ここに含まれるすべてのプログラムは、現存するままの状態で提供され、いかなる保証条件も適用されません。 第三者の権利の不侵害、商品性、特定目的適合性に関する黙示の保証の適用も一切ありません。

IBM Toolbox for Java クラス

IBM Toolbox for Java クラスは、(すべての Java クラスのように) パッケージに分類されます。 各パッケージは、特定の機能を提供します。 便宜上、この資料では通常、各パッケージを短縮名で参照します。 たとえば、com.ibm.as400.access パッケージは access パッケージと呼ばれます。

以下のリストのリンクを使用して、 さまざまな Toolbox for Java パッケージのクラスに関する情報を検索します。

- アクセス・クラスは、 iSeries 上のリソースをアクセスおよび管理するために使用します。
- HTML クラスは、HTML フォームおよびテーブルを迅速に作成するのに使用します。
- » Micro クラスは、 ワイヤレス装置に iSeries サーバー・データおよびサービスに直接アクセスを提供する Java プログラムの作成を可能にします。 «
- ReportWriter クラスは、 XML データ・ソースからフォーマット済み文書を作成できるようにします。
- リソース・クラスは、 iSeries リソースのアクセスおよび管理のために共通フレームワークを使用します。
- セキュリティー・クラスは、サーバーとの保護接続を確立したり、iSeries システム上で作業しているユーザーの身元を検査したりします。
- サーブレット・クラスは、 Java サーブレットで使用するデータを検索したりフォーマットしたりするのに使用します。
- ユーティリティー・クラスは、 AS400ToolboxInstaller クラスおよび AS400JarMaker クラスの使用などの管理タスクを行うために使用します。
- Vaccess クラスは、データをビジュアルに表現および操作するために使用します。

IBM Toolbox for Java の アクセス・クラス

IBM Toolbox for Java のアクセス・クラスは、iSeries のデータとリソースを表します。 クラスは iSeries および AS/400 サーバーと一緒に働いて、 サーバーのデータおよびリソースにアクセスしたり更新したりするための、 インターネット対応のインターフェースを提供します。

以下のクラスで、iSeries および AS/400e のリソースにアクセスできます。

- AS400 - サインオン情報の管理、 ソケット接続の作成と保守、およびデータの送受信を行います。
- SecureAS400 - 暗号化データを送信または受信するときに、 AS400 オブジェクトを使用できるようにします。
- AS400JPing - Java プログラムがホスト・サーバーに照会を出して、 実行中のサービスとサービスを提供しているポートが何であるかを調べることができるようにします。
- BidiTransform - 両方向テキストを独自の方法で変換できるようにします。
- » ClusteredHashTable クラス - Java プログラムが、 高可用性クラスター化ハッシュ・テーブル内のノード間で、 非永続データを共用および複製できるようにします。 «
- コマンド呼び出し - iSeries バッチ・コマンドを実行します。
- 接続プール - 接続を共用し、ユーザーが iSeries サーバーに設定できる接続の数を管理するために使用される、 AS400 オブジェクトのプールを管理します。
- データ域 - データ域の作成、アクセス、および削除を行います。
- データ変換および記述 - データを変換および処理し、 データ・バッファーのレコード様式を記述します。
- データ待ち行列 - データ待ち行列の作成、アクセス、変更、 および削除を行います。
- ディジタル証明書 - iSeries サーバーでディジタル証明書を管理します。
- 環境変数 - iSeries の環境変数を管理します。
- イベント・ログ - 例外およびメッセージを、 それらの表示に使用される装置からは独立してログに記録するための手段を提供します。
- 例外 - たとえば、装置エラーまたはプログラミング・エラーが起きた場合に、 エラーを出します。
- FTP - FTP 関数へのプログラマブル・インターフェースを提供します。
- 統合ファイル・システム - ファイルのアクセス、 ファイルのオープン、

入出力ストリームのオープン、 およびディレクトリーの内容のリストを行います。

- Java アプリケーション呼び出し - iSeries Java 仮想マシン上で稼働している iSeries サーバーで Java プログラムを呼び出します。
- JDBC - DB2 UDB for iSeries のデータにアクセスします。
- ジョブ - iSeries ジョブおよびジョブ・ログにアクセスします。
- メッセージ - iSeries システム上のメッセージおよびメッセージ待ち行列にアクセスします。
- NetServer 構成 - iSeries NetServer にアクセスして、 その状態および構成を変更します。
- 許可 - iSeries サーバーでのオブジェクト権限を表示および変更します。
- 印刷 - iSeries の印刷リソースを操作します。
- プロダクト・ライセンス - iSeries プロダクトのライセンスを管理します。
- プログラム呼び出し - iSeries プログラムを呼び出します。
- QSYS オブジェクト・パス名 - iSeries 統合ファイル・システム内のオブジェクトを表します。
- レコード・レベルでのアクセス - iSeries ファイルおよびメンバーの作成、読み取り、更新、 および削除を行います。
- サービス・プログラム呼び出し - iSeries サービス・プログラムを呼び出します。
- システム状況 - システム状況情報の表示およびシステム・プール情報へのアクセスを行います。
- システム値 - システム値およびネットワーク属性の検索と変更を行います。
- トレース (保守容易性) - トレース・ポイントおよび診断メッセージをログに記録します。
- ユーザーおよびグループ - iSeries のユーザーおよびグループにアクセスします。
- ユーザー・スペース - iSeries のユーザー・スペースにアクセスします。

注: Toolbox for Java は、iSeries オブジェクトおよびリストを扱う、 リソース・クラスと呼ばれる 2 番目のクラスの集合を提供します。 リソース・クラスは、さまざまな iSeries オブジェクトおよびリストを扱うための、 汎用フレームワークと一貫性のあるプログラミング・インターフェースを提供します。 アクセス・パッケージおよびリソース・パッケージのクラスに関する記述を読んでから、 ご使用のアプリケーションにとって最善のオブジェクトを選択できます。

AS400 クラス

AS400 クラスは以下のものを管理します。

- iSeries サーバー上のサーバー・ジョブへの一連のソケット接続。
- サーバーのサインオン動作。 これには、サインオン情報の入力のプロンプトや、パスワード・キャッシュ、 デフォルト・ユーザー管理が含まれます。

iSeries にアクセスするクラスのインスタンスを使用する場合、 Java プログラムは AS400 オブジェクトを提供しなければなりません。 たとえば、CommandCall オブジェクトは、 コマンドを iSeries に送信する前に AS400 オブジェクトが必要です。

AS400 オブジェクトが iSeries Java 仮想マシンで実行されている場合、 接続、ユーザー ID、およびパスワードは個別に処理されます。 詳細については、iSeries Java 仮想マシンを参照してください。

»現在 AS400 オブジェクトは、ユーザー ID およびパスワードの代わりに、Java Generic Security Service アプリケーション・プログラミング・インターフェース (JGSS API) をサーバーへの認証のために使用して、 Kerberos 認証をサポートします。

注: Kerberos チケットを使用するには、J2SDK v1.4 をインストールし、 Java Generic Security Services (JGSS) アプリケーション・プログラミング・インターフェースを構成することが必要です。 JGSS に関する詳細は、J2SDK, v1.4 Security Documentation を参照してください。«

AS400 オブジェクトを介して iSeries への接続を管理する方法の詳細については、接続の管理を参照してください。 接続プールからの接続を要求することによって初期接続時間を短縮する方法の詳細については、AS400ConnectionPool を参照してください。

AS400 クラスには、次のようなサインオン機能があります。

- ユーザー・プロファイルの認証
- »プロファイル・トークンの取得と関連したユーザー・プロファイルの認証«
- »プロファイル・トークンの設定«
- デフォルト・ユーザー ID の管理
- キャッシュ・パスワード
- ユーザー ID を尋ねるプロンプト

- パスワードの変更
- iSeries のバージョンおよびリリースの取得

暗号化されたデータを送信または受信するときに AS400 オブジェクトを使用することについての詳細は、 SecureAS400 クラスを参照してください。

SecureAS400 クラス

AS400 オブジェクトがサーバーと通信するとき、 ユーザー・データ (ユーザー・パスワードを除く) は、暗号化されずにサーバーに送信されます。 このため、AS400 オブジェクトに関連付けられている IBM Toolbox for Java オブジェクトは、 通常の接続を介してサーバーとデータを交換します。

IBM Toolbox for Java を使用してサーバーと機密データを交換したい場合には、 Secure Sockets Layer (SSL) を使用してデータを暗号化することができます。 暗号化したいデータを指定するには、 SecureAS400 オブジェクトを使用します。 SecureAS400 オブジェクトに関連付けられている IBM Toolbox for Java オブジェクトは、 セキュア接続を介してサーバーとデータを交換します。

» 詳細については、Secure Sockets Layer および Java Secure Socket Extension を参照してください。 «

SecureAS400 クラスは、 AS400 クラスのサブクラスです。

セキュア・サーバー接続をセットアップするには、以下に示すように、 SecureAS400 オブジェクトのインスタンスを作成します。

- SecureAS400(String systemName, String userID) では、 サインオン情報の入力を求めるプロンプトが出されます。
- SecureAS400(String systemName, String userID, String password) では、 サインオン情報の入力を求めるプロンプトが出されません。

以下の例では、CommandCall コマンドで、 セキュア接続を使用して iSeries システムにコマンドを送信する方法を示します。

```
 // Create a secure AS400 object.  This is the only statement that changes
 // from the non-SSL case.
SecureAS400 sys = new SecureAS400("mySystem.myCompany.com");

// Create a command call object
CommandCall cmd = new CommandCall(sys, "myCommand");

// Run the commands.  A secure connection is made when the
// command is run.  All the information that passes between the
// client and server is encrypted.
cmd.run();
```

AS400JPing

AS400JPing を使用すると、 java プログラムがホスト・サーバーに照会を出して、実行中のサービスとサービスを提供しているポートが何であるかを調べます。 コマンド行からサーバーに照会を出すには、JPing クラスを使用します。

AS400JPing クラスは、以下のいくつかのメソッドを提供します。

- サーバーを PING する (Ping the server)
- サーバー上の特定のサービスを PING する (Ping a specific service)
- PING 情報をログに記録するための PrintWriter オブジェクトを設定する (Set a PrintWriter object)
- PING 操作についてのタイムアウトを設定する (Set the time out)

例: iSeries リモート・コマンド・サービスを PING するために Java プログラム内で AS400JPing を使用する。

```
AS400JPing pingObj = new AS400JPing("myAS400", AS400.COMMAND, false);
  if (pingObj.ping())
    System.out.println("SUCCESS");
  else
    System.out.println("FAILED");
```

BidiTransform クラス

AS400BidiTransform クラスは、 iSeries 形式の両方向テキストを (まず Unicode に変換してから) Java 形式の両方向テキストに変換するか、 または Java 形式から iSeries 形式に変換することを可能にする、レイアウト変換を提供します。

AS400BidiTransform クラスを使用して、以下を行うことができます。

- システム CCSID を取得したり、設定したりする。
- iSeries データのストリング・タイプを取得したり、設定したりする。
- Java データのストリング・タイプを取得したり、設定したりする。
- Java レイアウトから iSeries へのデータの変換 (Convert data) を行う。
- iSeries レイアウトから Java へのデータの変換 (Convert data) を行う。

例: AS400BidiTransform クラスを使用して両方向テキストを変換する

以下の例は、AS400BidiTransform クラスを使用して両方向テキストを変換する方法を示しています。

```
// Java data to iSeries layout:
AS400BidiTransform abt;
abt = new AS400BidiTransform(424);
String dst = abt.toAS400Layout("some bidirectional string");
```

ClusteredHashTable クラス

ClusteredHashTable クラスは、Java プログラムが、 高可用性クラスター化ハッシュ・テーブルを使用して、 クラスター内のノード間でデータの共用とノンパーシスタント・ストレージへの複製を行えるようにします。 ClusteredHashTable クラスを使用するには、 そのデータにノンパーシスタント・ストレージを使用できることを確認してください。 複製されるデータは暗号化されません。

注: 以下の情報は、 iSeries クラスター化に関する一般的な概念および用語を理解していることが前提となっています。 クラスターとその使用方法に関する詳細は、クラスターを参照してください。

ClusteredHashTable クラスの使用には、クラスターが iSeries システム上で定義され、アクティブになっていることが必要です。 さらに、クラスター化ハッシュ・テーブル・サーバーを始動しなければなりません。 詳細は、クラスターの構成および Clustered Hash Table APIs を参照してください。

必須パラメーターは、クラスター化ハッシュ・テーブル・サーバーの名前、 およびそのサーバーを含むシステムを表す AS400 オブジェクトです。

クラスター化ハッシュ・テーブル・サーバーにデータを保管するには、 次のような接続ハンドルおよびキーが必要です。

- 接続をオープンする際、クラスター化ハッシュ・テーブル・サーバーは、 そのサーバーに対する後続の要求で指定する必要のある接続ハンドルを割り当てます。 この接続ハンドルはインスタンス化された AS400 オブジェクトに対してのみ有効ですので、 別の AS400 オブジェクトを使用する場合には他の接続をオープンする必要があります。
- クラスター化ハッシュ・テーブル内のデータにアクセスおよび変更するためのキーを指定しなければなりません。 重複するキーはサポートされません。

ClusteredHashTable クラスは、以下のアクションを実行できるようにするメソッドを提供します。

- クラスター化ハッシュ・テーブル・サーバー・ジョブへの接続をオープンする
- クラスター化ハッシュ・テーブルにデータを保管するための固有キーを生成する
- クラスター化ハッシュ・テーブル・サーバー・ジョブへのアクティブ接続をクローズする

ClusteredHashTable クラスのメソッドの中には、 ClusteredHashTableEntry クラスを使用して以下のアクションを実行するものがあります。

- クラスター化ハッシュ・テーブルからエントリーを取得する
- クラスター化ハッシュ・テーブルにエントリーを保管する

- すべてのユーザー・プロファイル用のクラスター化ハッシュ・テーブルから、エントリーのリストを取得する

以下の例は、CHTSVR01 という名前のクラスター化ハッシュ・テーブルで操作されます。 クラスターおよびクラスター化ハッシュ・テーブル・サーバーがすでにアクティブであることを前提とします。 接続をオープンし、キーを生成し、クラスター化ハッシュ・テーブル内で新しいキーを使用してエントリーを書き込み、 クラスター化ハッシュ・テーブルからエントリーを取得して、接続をクローズします。

```
ClusteredHashTableEntry myEntry = null;

String myData = new String("This is my data.");
System.out.println("Data to be stored: " + myData);

AS400 system = new AS400();

ClusteredHashTable cht = new ClusteredHashTable(system,"CHTSVR01");

// Open a connection.
cht.open();

// Get a key to the hash table.
byte[] key = null;
key = cht.generateKey();

// Prepare some data that you want to store into the hash table.
// ENTRY_AUTHORITY_ANY_USER means that any user can access the
// entry in the clustered hash table.
// DUPLICATE_KEY_FAIL means that if the specified key already exists,
// the ClusteredHashTable.put() request will not succeed.
int timeToLive = 500;
myEntry = new ClusteredHashTableEntry(key,myData.getBytes(),timeToLive,
     ClusteredHashTableEntry.ENTRY_AUTHORITY_ANY_USER,
     ClusteredHashTableEntry.DUPLICATE_KEY_FAIL);

// Store (or put) the entry into the hash table.
cht.put(myEntry);

// Get an entry from the hash table.
ClusteredHashTableEntry output = cht.get(key);

// Close the connection.
cht.close();
```

ClusteredHashTable クラスを使用すると、AS400 オブジェクトはサーバーに接続しま

す。 詳細については、接続の管理を参照してください。«

コマンド呼び出し

CommandCall クラスを使用すると、 Java プログラムが非対話式の iSeries コマンドを呼び出せるようになります。 コマンドの結果は、 AS400Message オブジェクトのリストで入手できます。

CommandCall への入力は次のとおりです。

- 実行したいストリング
- コマンドを実行するシステムを表す AS400 オブジェクト。

コマンド・ストリングは、コンストラクターを使用するか、 または setCommand() メソッド、 あるいは run() メソッドで設定することができます。 コマンド実行後に、Java プログラムは getMessageList() メソッドを使用して、そのコマンドの結果として出された iSeries メッセージを検索することができます。

CommandCall クラスを使用すると、AS400 オブジェクトは iSeries に接続します。

以下の例は、 iSeries システムでコマンドを実行するための CommandCall クラスの使用方法を示しています。

```
                    // Create an AS400 object.
AS400 sys = new AS400("mySystem.myCompany.com");

                    // Create a command call object. This
                    // program sets the command to run
                    // later. It could set it here on the
                    // constructor.
CommandCall cmd = new CommandCall(sys);

                    // Run the CRTLIB command
cmd.run("CRTLIB MYLIB");

                    // Get the message list which
                    // contains the result of the
                    // command.
AS400Message[] messageList = cmd.getMessageList();

                    // ... process the message list.
```

```
                        // Disconnect since I am done sending
                        // commands to the server
    sys.disconnectService(AS400.COMMAND);
```

CommandCall クラスを使用すると、AS400 オブジェクトは iSeries に接続します。 接続を管理する方法の詳細については、接続の管理を参照してください。

» Java プログラムと iSeries サーバー・コマンドが同じサーバー上にある場合、Toolbox for Java のデフォルトの動作では、 システム上のコマンドがスレッド・セーフであるかどうかを調べます。« スレッド・セーフである場合、コマンドはスレッド上で実行されます。 setThreadSafe() メソッドを使用して、 コマンドがスレッド・セーフであることを明示的に指定することにより、実行時の検索を抑制することができます。

例

ユーザー指定のコマンドの実行

接続プール

接続プールを使用し、接続を共用して iSeries サーバーへの接続のセット (プール) を管理します。 たとえば、アプリケーションはプールから接続を取り出し、 それを使用して、その後再使用のためにプールに戻すことができます。

AS400ConnectionPool クラスは、 AS400 オブジェクトのプールを管理します。 AS400JDBCConnectionPool クラスは、 JDBC 2.0 Optional Package API の Toolbox サポートの一部として、 Java プログラムによって使用できる AS400JDBCConnections のプールを表します。 » また JDBC ConnectionPool インターフェースは、Java 2 Platform、Standard Edition、 バージョン 1.4 にバンドルされている、JDBC 3.0 API でサポートされます。«

どちらのタイプの接続プールも、それが作成した接続の数を記録します。 ConnectionPool から継承したメソッドを使用して、 以下を含む接続プールのいくつかのプロパティーを設定できます。

- 1 つのプールが作成できる接続の最大数 (maximum number of connections)
- 1 つの接続の最大存続時間 (maximum lifetime)
- 1 つの接続の最大非活動時間 (maximum inactivity time)

サーバーへの接続は、パフォーマンスの面では負荷のかかる操作です。 接続プールの使用により、何度も接続が繰り返されることを回避してパフォーマンスを向上できます。 たとえば、プールを活動状態の (接続済みの) 接続で満たすことにより、 接続プールを作成する時に接続を作成します。 新しい接続を作成するのではなく、接続プールを使用して接続オブジェクトを簡単に取り出し、 使用し、戻し、再使用することができます。

»接続を取り出すには、AS400ConnectionPool にシステム名、ユーザー ID、 パスワード、およびサービス (オプション) を指定して使用します。«接続先のサービスを指定するには、 AS400 クラスからの定数を使用します。 (FILE、PRINT、COMMAND など)

接続を取り出して使用した後、アプリケーションは接続をプールに戻します。 再使用のために接続をプールに戻すことは、各アプリケーションの責任です。 接続がプールに戻されないと、接続プールのサイズは大きくなり続け、接続は再利用されません。

AS400ConnectionPool クラスの使用時に iSeries への接続がオープンしたときの管理に関する詳細は、接続の管理を参照してください。

例: AS400ConnectionPool を使用して、 AS400 オブジェクトを再使用する

データ域

DataArea クラスは、 iSeries データ域オブジェクトを表す抽象基本クラスです。 この基本クラスには 4 つのサブクラスがあり、 それぞれ、文字データ、10 進数データ、論理データ、および文字データが入っている内部データ域をサポートしています。

DataArea クラスを使用して、以下を行うことができます。

- データ域のサイズを取得する。
- データ域の名前を取得する。
- データ域の AS400 システム・オブジェクトを戻す。
- データ域の属性を最新表示する。
- データ域が存在するシステムを設定する。

DataArea クラスを使用すると、AS400 オブジェクトはサーバーに接続します。 接続を管理する方法の詳細については、接続の管理を参照してください。

CharacterDataArea

CharacterDataArea クラスは、 サーバー上の文字データが入っているデータ域を表します。 文字データ域では正しい CCSID を使用してデータをタグ付けすることができません。 そのため、データ域オブジェクトはそのデータがユーザーの CCSID を使用していると見なします。 書き込みを行う場合、データ域オブジェクトはストリング (Unicode) をユーザーの CCSID に変換してから、 データをサーバーに書き込みます。 読み取りを行う場合には、データ域オブジェクトはデータがユーザーの CCSID であると見なし、 CCSID から Unicode に変換してから、プログラムにストリングを戻します。 データ域からデータを読み取る場合、 読み取られるデータの量は、バイト数ではなく、文字の数で示されます。

CharacterDataArea クラスを使用して、以下を行うことができます。

- データ域の消去を行い、 中身を空にする
- デフォルトのプロパティー値を使用してシステム上に文字データ域を作成する
- 特定の属性 (specific attributes) を使用して文字データ域を作成する
- データが存在するシステムからデータ域を削除する
- データ域によって表されるオブジェクトの統合ファイル・システム・パス名を戻す
- データ域内に入っているすべてのデータの読み取りを行う
- 指定した量のデータを、 オフセット 0 または指定したオフセットから始まるデータ域から読み取る
- データ域の完全修飾統合ファイル・システム・パス名を設定する
- データをデータ域の先頭に書き込む
- 指定した量のデータを、オフセット 0 または指定したオフセットから始まるデータ域に書き込む

DecimalDataArea

DecimalDataArea クラスは、 サーバー上の 10 進数データが入っているデータ域を表します。

DecimalDataArea クラスを使用して、以下を行うことができます。

- データ域の消去を行い、 中身を 0.0 にする
- デフォルトのプロパティー値を使用してシステム上に 10 進データ域を作成する
- 指定した属性 (specified attributes) を使用して 10 進データ域を作成する
- データが存在するサーバーからデータ域を削除する
- データ域で小数点の右に桁数を戻す
- データ域によって表されるオブジェクトの統合ファイル・システム・パス名を戻す
- データ域内に入っているすべてのデータの読み取りを行う
- データ域の完全修飾統合ファイル・システム・パス名を設定する
- データをデータ域の先頭に書き込む

以下の例は、 10 進数データ域を作成し、書き込みを行う方法を示しています。

```
   // Establish a connection to the server "My400".
AS400 system = new AS400("MyServer");
   // Create a DecimalDataArea object.
QSYSObjectPathName path = new QSYSObjectPathName("MYLIB", "MYDATA", "DTAARA");
DecimalDataArea dataArea = new DecimalDataArea(system, path.getPath());
   // Create the decimal data area on the server using default values.
dataArea.create();
   // Clear the data area.
dataArea.clear();
   // Write to the data area.
dataArea.write(new BigDecimal("1.2"));
   // Read from the data area.
BigDecimal data = dataArea.read();
   // Delete the data area from the server.
dataArea.delete();
```

LocalDataArea

LocalDataArea クラスは、 サーバー上の内部データ域を表します。 内部データ域はサーバー上では文字データ域として存在していますが、 内部データ域には注意しなければならない制限があります。

内部データ域はサーバー・ジョブと関連付けられており、 他のジョブからはアクセスできません。 そのため、内部データ域は作成も削除もできません。 サーバー・ジョブが終了すれば、 そのサーバー・ジョブに関連付けられていた内部データ域は自動的に削除され、 そのジョブを参照していた LocalDataArea オブジェクトは有効ではなくなります。 内部データ域は、 サーバー上で 1024 文字にサイズが固定されていることにも注意しなければなりません。

LocalDataArea クラスを使用して、以下を行うことができます。

- データ域の消去を行い、 中身を空にする
- データ域内に入っているすべてのデータの読み取りを行う
- 指定した量のデータを、 指定したオフセットから始まるデータ域から読み取る
- データをデータ域の先頭に書き込む
- 指定した量のデータを先頭文字がオフセットに書き込まれたデータ域に書き込む

LogicalDataArea

LogicalDataArea クラスは、 サーバー上の論理データが入っているデータ域を表します。

LogicalDataArea クラスを使用して、以下を行うことができます。

- 消去を行い、 中身を false にする
- デフォルトのプロパティー値を使用してサーバー上に文字データ域を作成する
- 指定した属性 (specified attributes) を使用して文字データ域を作成する
- データが存在するサーバーからデータ域を削除する
- データ域によって表されるオブジェクトの統合ファイル・システム・パス名を戻す
- データ域内に入っているすべてのデータの読み取りを行う
- データ域の完全修飾統合ファイル・システム・パス名を設定する
- データをデータ域の先頭に書き込む

DataAreaEvent

DataAreaEvent クラスは、 データ域イベントを表します。

任意の DataArea クラスを指定して、 DataAreaEvent クラスを使用することができます。
DataAreaEvent クラスを使用して、以下を行うことができます。

- イベントの識別コードを取得する。

DataAreaListener

DataAreaListener クラスでは、 データ域イベントの受け取りに使用するインターフェースが用意されています。

任意の DataArea クラスを指定して、 DataAreaListener クラスを使用することができます。
以下のいずれかが実行される場合、 DataAreaListener クラスを呼び出すことができます。

- 消去
- 作成
- 削除
- 読み取り
- 書き込み

データ変換および記述

データ変換クラスを使って、 数値と文字データを iSeries 形式と Java 形式との間で変換することができます。 変換は、Java プログラムから iSeries データにアクセスする場合に必要となります。 データ変換クラスは、多様な数値形式の変換や、 EBCDIC コード・ページと Unicode 間の変換をサポートします。

データ記述クラスは、データ変換クラスの上に構築され、 1 つのレコードにあるすべてのフィールドを、単一メソッド呼び出しで変換します。 RecordFormat クラスを使うと、プログラムで、 DataQueueEntry を構成するデータ、 ProgramCall パラメーター、 レコード・レベルでのアクセス・クラスでアクセスしたデータベース・ファイルのレコード、 または iSeries データのバッファーを記述できるようになります。 Record クラスを使うと、プログラムがフィールド名または索引を使用してレコードの内容を変換し、 データにアクセスできるようになります。

データ・タイプ

AS400DataType は、 データ変換に必要なメソッドを定義するインターフェースです。 Java プログラムは、データの部分的な変換が必要な場合にデータ・タイプを使用します。 次のようなデータ・タイプの変換クラスが存在します。

- 数値
- テキスト (文字)
- 複合 (数値およびテキスト)

レコード様式を指定する変換

IBM Toolbox for Java には、一度に 1 フィールドではなく、 1 レコードのデータ変換を行うためのデータ・タイプ・クラスを構築するクラスがあります。 たとえば、Java プログラムがデータ待ち行列からデータを読み取るとします。 データ待ち行列オブジェクトは、iSeries データのバイト配列を Java プログラムに戻します。 この配列には、さまざまなタイプの iSeries データが含まれている可能性があります。 プログラムは、データ・タイプ・クラスを使用して、 バイト配列から一度に 1 フィールドのデータを変換することができます。 また、プログラムは、バイト配列内のフィールドを記述するレコード様式を作成することもできます。 その後、このレコードが変換を行います。

レコード様式による変換は、プログラム呼び出し、データ待ち行列、 およびレコード・レベルでのアクセス・クラスからのデータを処理する場合に役立ちます。 これらのクラスからの入出力データは、 さまざまなタイプの複数のフィールドを含むバイト配列です。 レコード様式コンバーターを使用すること

により、 iSeries 形式と Java 形式間でのこのデータの変換を簡単に行うことができます。

レコード様式による変換では、次の 3 つのクラスを使用します。

- FieldDescription クラスは、 フィールドまたはパラメーターをデータ・タイプおよび名前で識別します。
- RecordFormat クラスは、フィールドのグループを記述します。
- Record クラスは、 RecordFormat クラスにあるレコードの記述を、実際のデータと結合します。
- A LineDataRecordWriter クラスは、 レコードを OutputStream に行データ形式で書き込みます。

例

次の 2 つの例は、レコード様式変換クラスをデータ待ち行列と共に使用する方法を示しています。

- 例: Record および RecordFormat クラスを使用して待ち行列にデータを書き込む
- 例: FieldDescription、RecordFormat、および Record クラスを使用する

数値データの変換クラス

数値データの変換クラスは、 数値データを iSeries または AS/400e サーバーで使用される形式 (以下の表で サーバー形式と呼ばれる) から Java 形式に単純に変換します。 下の表に、サポートされるタイプを示します。

数値タイプ	説明
AS400Bin2	符号付き 2 バイトの数値データと Java Short オブジェクト間の変換
AS400Bin4	符号付き 4 バイトの数値データと Java Integer オブジェクト間の変換
AS400ByteArray	2 バイト配列間の変換。 このタイプは、ターゲット・バッファーに正しく 0 を入れ、 埋め込みを行います。
AS400Float4	サーバー形式の符号付き 4 バイト浮動小数点の数値データと Java Float オブジェクト間の変換
AS400Float8	サーバー形式の符号付き 8 バイト浮動小数点の数値データと Java Double オブジェクト間の変換
AS400PackedDecimal	サーバー形式のパック 10 進の数値データと Java BigDecimal オブジェクト間の変換
AS400UnsignedBin2	サーバー形式の符号なし 2 バイトの数値データと Java Integer オブジェクト間の変換
AS400UnsignedBin4	サーバー形式の符号なし 4 バイトの数値データと Java Long オブジェクト間の変換
AS400ZonedDecimal	サーバー形式のゾーン 10 進の数値データと Java BigDecimal オブジェクト間の変換

以下の例では、サーバー形式の数値データ・タイプを Java int に変換します。

```
                    // Create a buffer to hold the server data
                    // type. Assume the buffer is filled with
                    // numeric data in the server format by data
                    // queues, program call, etc.
byte[] data = new byte[100];

                    // Create a converter for this
                    // server data type.
AS400Bin4 bin4Converter = new AS400Bin4();

                    // Convert from server type to Java
                    // object. The number starts at the
```

```
                    // beginning of the buffer.
Integer intObject = (Integer) bin4Converter.toObject(data,0);

                    // Extract the simple Java type from
                    // the Java object.
int i = intObject.intValue();
```

以下の例では、Java int をサーバー形式の数値データ・タイプに変換します。

```
                    // Create a Java object that contains
                    // the value to convert.
Integer intObject = new Integer(22);

                    // Create a converter for the server
                    // data type.
AS400Bin4 bin4Converter = new AS400Bin4();

                    // Convert from Java object to
                    // server data type.
byte[] data = bin4Converter.toBytes(intObject);

                    // Find out how many bytes of the
                    // buffer were filled with the
                    // server value.
int length = bin4Converter.getByteLength();
```

テキスト変換

文字データは、 AS400Text クラスを介して変換されます。 このクラスでは、EBCDIC コード・ページまたは文字セット (CCSID) と Unicode 間の文字データの変換を行います。 AS400Text オブジェクトが構成されると、 Java プログラムは変換するストリングの長さを指定し、サーバー CCSID またはエンコードを指定します。 Java プログラムの CCSID は »13488 Unicode であることを前提としています。 «toBytes() メソッドは、 Java 形式をサーバー形式のバイト配列に変換します。 toObject() メソッドは、 iSeries 形式のバイト配列を Java 形式に変換します。

AS400BidiTransform クラスは、 iSeries 形式の両方向テキストを (まず Unicode に変換してから) Java 形式の両方向テキストに変換するか、 または Java 形式から iSeries 形式に変換することを可能にする、レイアウト変換を提供します。 デフォルトの変換は、ジョブの CCSID に基づいています。 テキストの方向および形状を変更するには、 BidiStringType を指定します。 IBM Toolbox for Java オブジェクトが内部的に変換を行うときには、DataArea クラスでの場合のように、 そのオブジェクトにストリング・タイプを変更するメソッドがあることに注意してください。 例えば、DataArea クラスには addVetoableChangeListener() メソッドがあります。 このメソッドを指定して、ストリング・タイプを含む特定のプロパティーに対する禁止権の変更を listen することができます。

例えば、 DataQueueEntry オブジェクトが iSeries テキストを EBCDIC で戻すものとします。 以下の例では、このデータを Unicode に変換し、Java プログラムが使用できるようにします。

»

```
        // ... Assume the data queue work has already been done to
        // retrieve the text from the iSeries and the data has been
        // put in the following buffer.
int textLength = 100;
byte[] data = new byte[textLength];

        // Create a converter for the iSeries data type. Note a default
        // converter is being built.  This converter assumes the iSeries
        // EBCDIC code page matches the client's locale.  If this is not
        // true the Java program can explicitly specify the EBCDIC
        // CCSID to use. However, it is recommended that you specify a
        // CCSID whenever possible (see the Notes: below).
AS400Text textConverter = new AS400Text(textLength)

        // Note: Optionally, you can create a converter for a specific
        // CCSID. Use an AS400 object in case the program is running
        // as a Toolbox for Java proxy client.
int ccsid = 37;
AS400 system = ...; // AS400 object
```

```
AS400Text textConverter = new AS400Text(textLength, ccsid, system);

     // Note: You can also create a converter with just the AS400 object.
     // This converter assumes the iSeries code page matches
     // the CCSID returned by the AS400 object.
AS400Text textConverter = new AS400Text(textLength, system);

     // Convert the data from EBCDIC to Unicode. If the length of
     // the AS400Text object is longer than the number of
     // converted characters, the resulting String will be
     // blank-padded out to the specified length.
String javaText = (String) textConverter.toObject(data);
```

複合タイプの変換クラス

複合タイプの変換クラスを次に示します。

- AS400Array - Java プログラムがデータ・タイプの配列を処理できるようにする。
- AS400Structure - Java プログラムが、 データ・タイプを要素として持つ構造を処理できるようにする。

以下の例では、Java 構造をバイト配列に変換し、また Java 構造に戻します。 ここでは、データの送受信で同じデータ・フォーマットが使用されるものとします。

```
                    // Create a structure of data types
                    // that corresponds to a structure
                    // that contains:
                    //    - a four-byte number
                    //    - four bytes of pad
                    //    - an eight-byte number
                    //    - 40 characters
AS400DataType[] myStruct =
{
   new AS400Bin4(),
   new AS400ByteArray(4),
   new AS400Float8(),
   new AS400Text(40)
};

                    // Create a conversion object using
                    // the structure.
AS400Structure myConverter = new AS400Structure(myStruct);

                    // Create the Java object that holds
                    // the data to send to the server.
Object[] myData =
{
     new Integer(88),        // the four-byte number
     new byte[0],            // the pad (let the conversion object 0 pad)
     new Double(23.45),      // the eight-byte floating point number
     "This is my structure"  // the character string
};

                    // Convert from Java object to byte array.
byte[] myAS400Data = myConverter.toBytes(myData);


                    // ... send the byte array to the
                    // server. Get data back from the
```

```
                    // server.  The returned data will
                    // also be a byte array.


                    // Convert the returned data from
                    // iSeries to Java format.
Object[] myRoundTripData =
             (Object[])myConverter.toObject(myAS400Data,0);

                    // Pull the third object out of the
                    // structure. This is the double.
Double doubleObject = (Double) myRoundTripData[2];

                    // Extract the simple Java type from
                    // the Java object.
double d = doubleObject.doubleValue();
```

FieldDescription クラス

FieldDescription クラスを使用すると、 データ・タイプおよびフィールド名を含むストリングを使用して、 Java プログラムがフィールドまたはパラメーターの内容を記述することができます。 プログラムがレコード・レベルでのアクセスによるデータを処理している場合、 iSeries または AS/400e データ記述仕様 (DDS) のキーワードも指定します。 このキーワードは、フィールドを記述します。

フィールド記述クラスには、以下のようなものがあります。

- BinaryFieldDescription
- CharacterFieldDescription
- DateFieldDescription
- DBCSEitherFieldDescription
- DBCSGraphicFieldDescription
- DBCSOnlyFieldDescription
- DBCSOpenFieldDescription
- FloatFieldDescription
- HexFieldDescription
- PackedDecimalFieldDescription
- TimeFieldDescription
- TimestampFieldDescription
- ZonedDecimalFieldDescription

たとえば、データ待ち行列内のデータは同じ形式であるものとします。 各データには、メッセージ番号 (AS400Bin4)、タイム・スタンプ (8 文字)、 およびメッセージ・テキスト (50 文字) があります。 これらは、フィールド記述で次のように記述されます。

```
                    // Create a field description for
                    // the numeric data. Note it uses
                    // the AS400Bin4 data type. It also
                    // names the field so it can be
                    // accessed by name in the record
                    // class.
BinaryFieldDescription bfd = new BinaryFieldDescription(new AS400Bin4(),
                                                        "msgNumber");

                    // Create a field description for
                    // the character data. Note it uses
                    // the AS400Text data type. It also
                    // names the field so it can be
                    // accessed by name by the record
                    // class.
CharacterFieldDescription cfd1 = new CharacterFieldDescription(new AS400Text(8),
                                                               "msgTime");
```

```
                        // Create a field description for
                        // the character data. Note it uses
                        // the AS400Text data type. It also
                        // names the field so it can be
                        // accessed by name by the record
                        // class.
    CharacterFieldDescription cfd2 = new CharacterFieldDescription(new AS400Text(50),
                                                                   "msgText");
```

これで、フィールド記述をレコード様式クラスにまとめることができます。 例の続きは、レコード様式のセクションにあります。

RecordFormat クラス

Recordformat クラスは、 Java プログラムが特定のグループ内のフィールドまたはパラメーターを記述できるようにします。 レコード・オブジェクトには、RecordFormat オブジェクトによって記述されたデータが含まれています。 プログラムがレコード・レベルでのアクセス・クラスを使用している場合、RecordFormat クラスはプログラムもキー・フィールドの記述を指定できるようにします。

RecordFormat オブジェクトには、一連のフィールド記述が含まれています。フィールド記述へのアクセスは、索引と名前のいずれからでもできます。RecordFormat クラスのメソッドは、以下のことを行います。

- レコード様式にフィールド記述を追加する。
- レコード様式にキー・フィールド記述を追加する。
- 索引または名前によるレコード様式からフィールド記述を検索する。
- 索引または名前によるレコード様式からキー・フィールド記述を検索する。
- レコード様式を形成するフィールドの名前を検索する。
- レコード様式を形成するキー・フィールドの名前を検索する。
- レコード様式内にあるフィールドの数を検索する。
- レコード様式内にあるキー・フィールドの数を検索する。
- このレコード様式に基づいた Record オブジェクトを作成する。

たとえば、フィールド記述例で作成したフィールド記述をレコード様式に追加するには、次のようにします。

```
                    // Create a record format object,
                    // then fill it with field
                    // descriptions.
RecordFormat rf = new RecordFormat();
rf.addFieldDescription(bfd);
rf.addFieldDescription(cfd1);
rf.addFieldDescription(cfd2);
```

これで、プログラムがレコード様式からレコードを作成できるようになりました。 例の続きは、Record のセクションにあります。

Record クラス

Record クラスは、Java プログラムがレコード様式クラスで記述されたデータを処理できるようにします。 データ変換は、サーバー・ データを含むバイト配列と Java オブジェクトとの間で行われます。 Record クラスのメソッドは、以下のことを行います。

- Java オブジェクトとしてフィールドの内容を検索 (索引または名前を使用) する
- 数を検索する。
- Java オブジェクトでフィールドの内容を設定 (索引または名前を使用) する
- レコードの内容をサーバー・データとして検索し、 バイト配列または出力ストリームに出力する。
- レコードの内容をバイト配列または入力ストリームから設定する。
- レコードの内容をストリングに変換する。

たとえば、レコード様式の例で作成したレコード様式を使用するには、 次のようにします。

```
                    // Assume data queue setup work has
                    // already been done. Now read a
                    // record from the data queue.
DataQueueEntry dqe = dq.read();

                    // The data from the data queue is
                    // now in a data queue entry. Get
                    // the data out of the data queue
                    // entry and put it in the record.
                    // We obtain a default record from
                    // the record format object and
                    // initialize it with the data from the
                    // data queue entry.
Record dqRecord = rf.getNewRecord(dqe.getData());

                    // Now that the data is in the
                    // record, pull the data out one
                    // field at a time, converting the
                    // data as it is removed. The result
                    // is data in a Java object that the
                    // program can now process.
```

```
Integer msgNumber = (Integer) dqRecord.getField("msgNumber");
String  msgTime   = (String)  dqRecord.getField("msgTime");
String  msgText   = (String)  dqRecord.getField("msgText");
```

LineDataRecordWriter クラス

LineDataRecordWriter クラスは、 レコード・データを行データ形式で OutputStream に書き込みます。 このクラスは指定した CCSID を使用してデータをバイトに変換します。 レコードに関連したレコード様式がデータの形式を決定します。

LineDataRecordWriter を使用するには、次のレコード様式属性を設定する必要があります。

- レコード様式 ID
- レコード様式タイプ

Record または RecordFormat クラスに関連して、 LineDataRecordWriter はレコードを writeRecord() メソッドへの入力として受け取ります。 (レコードはユーザーによってインスタンス化されたときに RecordFormat を入力として受け取ります。)

LineDataRecordWriter クラスには、以下のことを行えるメソッドがあります。

- CCSID を取得する。
- エンコードの名前を取得する。
- OutputStream に行データ形式でレコード・データを書き込む。

例: LineDataRecordWriter クラスを使用する

```
// Example using the LineDataRecordWriter class.
   try
   {
     // create a ccsid
     ccsid_ = system_.getCcsid();

     // create output queue and specify spooled file data to be *LINE
     OutputQueue outQ = new OutputQueue(system_, "/QSYS.LIB/RLPLIB.LIB/LDRW.OUTQ");
     PrintParameterList parms = new PrintParameterList();
     parms.setParameter(PrintObject.ATTR_PRTDEVTYPE, "*LINE");

     // initialize the record format for writing data
     RecordFormat recfmt = initializeRecordFormat();

     // create a record and assign data to be printed...
     Record record = new Record(recfmt);
     createRecord(record);

     SpooledFileOutputStream os = null;
     try {
         // create the output spooled file to hold the record data
         os = new SpooledFileOutputStream(system_, parms, null, outQ);
     }
     catch (Exception e) {
         System.out.println("Error occurred creating spooled file");
         e.printStackTrace();
     }

     // create the line data record writer
```

```
  LineDataRecordWriter ldw;
  ldw = new LineDataRecordWriter(os, ccsid_, system_);

  // write the record of data
  ldw.writeRecord(record);

  // close the outputstream
  os.close();
}

catch(Exception e)
{
  failed(e, "Exception occurred.");
}
```

データ待ち行列

DataQueue クラスを使用すると、 Java プログラムとサーバー・データ待ち行列との間でやり取りができるようになります。 iSeries および AS/400e サーバーのデータ待ち行列には、以下の特性があります。

- データ待ち行列により、ジョブ同士の高速通信が可能になる。 したがって、ジョブ間でデータを同期させたり受け渡しする際の優れた方法といえます。
- 多数のジョブが同時にデータ待ち行列をアクセスできる。
- データ待ち行列上のメッセージが、フリー・フォーマットである。 データベース・ファイルで必要とされるようなフィールドは、 データ待ち行列では必要ありません。
- データ待ち行列は、同期処理または非同期処理のどちらにでも使用できる。
- データ待ち行列上のメッセージは、以下のいずれかの方法で並べることができる。
  - 後入れ先出し法 (LIFO)。 待ち行列から取り出される最初のメッセージは、 データ待ち行列上の最後の (最新の) メッセージになります。
  - 先入れ先出し法 (FIFO)。 待ち行列から取り出される最初のメッセージは、 データ待ち行列上の最初の (一番古い) メッセージになります。
  - キー順。 データ待ち行列上の各メッセージには、それぞれに関連付けられたキーがあります。 メッセージは、それぞれに関連付けられたキーを指定することによってのみ、 待ち行列から取り出すことができます。

データ待ち行列クラスを使用すると、 Java プログラムからサーバー・データ待ち行列をアクセスするための一連の完全なインターフェースが提供されます。 これは、任意のプログラム言語で作成された Java プログラムとサーバー上のプログラムとの間で通信する際の優れた方法と言えます。

各データ待ち行列オブジェクトの必須パラメーターは、 AS400 オブジェクトです。 このオブジェクトは、データ待ち行列のあるサーバー、 あるいはデータ待ち行列を作成するサーバーを表します。

データ待ち行列クラスを使用すると、AS400 オブジェクトとサーバーが接続されます。 接続の管理については、接続の管理を参照してください。

各データ待ち行列オブジェクトには、 そのデータ待ち行列の統合ファイル・システム・パス名が必要です。 このデータ待ち行列のタイプは DTAQ です。 詳細については、統合ファイル・システム・パス名を参照してください。

順次データ待ち行列およびキー付データ待ち行列

データ待ち行列クラスは、次のようなデータ待ち行列をサポートします。

- 順次データ待ち行列

- キー順データ待ち行列

両方のタイプの待ち行列に共通するメソッドは、 BaseDataQueue クラスに含まれています。 DataQueue クラスは、 順次データ待ち行列の実装を完全にするために、 BaseDataQueue クラスを拡張します。 さらに、この BaseDataQueue クラスは、 キー順データ待ち行列の実装を完全にするために、 KeyedDataQueue クラスによって拡張されます。

データがデータ待ち行列から読み取られると、 そのデータは DataQueueEntry オブジェクトに置かれます。 このオブジェクトには、 キー順データ待ち行列と順次データ待ち行列の両方のデータが入れられます。 キー順データ待ち行列からの読み取り時に使用可能な追加データは、 DataQueueEntry クラスを拡張した KeyedDataQueueEntry オブジェクトに入れられます。

このデータ待ち行列クラスは、サーバー・データ待ち行列へ書き込まれるデータ、あるいはサーバー・データ待ち行列から読み取られるデータを変更しません。 Java プログラムでは、このデータを適切にフォーマットする必要があります。 データ変換クラスは、データを変換するためのメソッドを提供します。

次の例では、DataQueue オブジェクトを作成し、 DataQueueEntry オブジェクトからデータを読み取った後、 システムから切断を行います。

```
                    // Create an AS400 object
AS400 sys = new AS400("mySystem.myCompany.com");

                    // Create the DataQueue object
DataQueue dq = new DataQueue(sys, "/QSYS.LIB/MYLIB.LIB/MYQUEUE.DTAQ");

                    // read data from the queue
DataQueueEntry dqData = dq.read();

                    // get the data out of the DataQueueEntry object.
byte[] data = dqData.getData();

                    // ... process the data

                    // Disconnect since I am done using data queues
sys.disconnectService(AS400.DATAQUEUE);
```

順次データ待ち行列

サーバーの順次データ待ち行列上のエントリーは、 先入れ先出し法 (FIFO) または後入れ先出し法 (LIFO) の順序で削除されます。 BaseDataQueue および DataQueue クラスを使用すると、 順次データ待ち行列を処理するための以下のメソッドが提供されます。

- データ待ち行列をサーバー上に作成する。 Java プログラムで、 データ待ち行列上のエントリーの最大サイズを指定する必要があります。 待ち行列の作成時に、 この Java プログラムで追加のデータ待ち行列パラメーター (FIFO と LIFO、 送信者の情報の保管、権限情報の指定、ディスクへの保管、 および待ち行列の説明) を任意に指定することができます。
- 待ち行列からエントリーを削除せずに、 データ待ち行列上のエントリーを照合する。Java プログラムを使用し、 待ち行列上にエントリーがなければすぐに待機したり戻したりすることができます。
- エントリーを待ち行列から読み取る。 Java プログラムを使用し、 待ち行列上に使用可能なエントリーがなければすぐに待機したり戻したりすることができます。
- エントリーを待ち行列に書き込む。
- 待ち行列からすべてのエントリーを消去する。
- 待ち行列を削除する。

BaseDataQueue クラスを使用すると、 データ待ち行列の属性を検索するための追加メソッドが提供されます。

例

順次データ待ち行列の例では、作成側は項目をデータ待ち行列に置き、 使用側はその待ち行列からその項目を取り出して処理します。

- 順次データ待ち行列の作成側の例
- 順次データ待ち行列の使用側の例

キー順データ待ち行列

BaseDataQueue および KeyedDataQueue クラスを使用すると、 キー順データ待ち行列を処理するための以下のメソッドが提供されます。

- キー順データ待ち行列をサーバー上に作成する。 Java プログラムで、待ち行列上のエントリーのキーの長さと最大サイズを指定する必要があります。 この Java プログラムでは、任意で権限情報を指定し、送信者の情報を保管し、 ディスクへ保管し、待ち行列を説明することができます。
- 待ち行列からエントリーを削除せずに、 データ待ち行列上のエントリーを照合する。 Java プログラムを使用し、待ち行列上にキーの基準に適合するエントリーがなければすぐに待機したり戻したりすることができます。
- 指定したキーに基づいて、エントリーを待ち行列から読み取る。 Java プログラムを使用し、 待ち行列上にキーの基準に適合した使用可能なエントリーがなければすぐに待機したり戻したりすることができます。
- キー付きエントリーを待ち行列に書き込む。
- すべてのエントリー、または指定したキーに適合するすべてのエントリーを消去する。
- 待ち行列を削除する。

BaseDataQueue クラスおよび KeyedDataQueue クラスを使用すると、 データ待ち行列の属性を検索するための追加メソッドも提供されます。

例

次のキー順データ待ち行列の例では、作成側は項目をデータ待ち行列に置き、使用側はその待ち行列からその項目を取り出して処理します。

- キー順データ待ち行列の作成側の例
- キー順データ待ち行列の使用側の例

ディジタル証明書

ディジタル証明書とは、インターネットを介したトランザクションの安全性を確保するのに使われる、 ディジタル署名付きのステートメントのことです。 (ディジタル証明書は、OS/400 バージョン 4 リリース 3 (V4R3) 以降で稼働するサーバーで使用できます。) Secure Sockets Layer (SSL) を使って安全な接続を確立するには、ディジタル証明書が必要です。

ディジタル証明書は、次のもので構成されます。

- ユーザーの公開暗号鍵
- ユーザーの名前とアドレス
- 第三者の認証局 (CA) のディジタル署名。 認証局の署名があれば、そのユーザーは信用のあるエンティティーであるということです。
- 証明書の発行日
- 証明書の満了日

セキュア・サーバー管理者は、認証局の「トラステッド・ルート鍵」をサーバーに追加することができます。 これは、該当する認証局によって認証を受けた全員がサーバーで信用されることを意味します。

また、ディジタル証明書でも暗号化を行うことができます。 それによって、秘密暗号鍵を介して確実にデータを安全に転送できるようになります。

ディジタル証明書は、javakey ツールを使って作成できます。 (javakey および Java セキュリティーの詳細については、 Sun Microsystems, Inc. の Java Security のページ を参照してください。) IBM Toolbox for Java ライセンス・プログラムには、 iSeries または AS/400e サーバー上でディジタル証明書を管理するクラスがあります。

AS400Certificate クラスには、 X.509 ASN.1 エンコードの証明書を管理するメソッドが用意されています。 用意されているクラスとその機能は、以下のとおりです。

- 証明書データの取得と設定。
- 妥当性検査リストまたはユーザー・プロファイル別の証明書のリスト。
- 証明書の管理。たとえば、ユーザー・プロファイルへの証明書の追加や、 妥当性検査リストからの証明書の削除。

証明書クラスを使用すると、AS400 オブジェクトがサーバーに接続されます。 接続の管理については、接続の管理を参照してください。

サーバーでは、証明書は、妥当性検査リストまたはユーザー・プロファイルに帰属します。

- AS400CertificateUserProfileUtil クラスでは、 証明書を管理するためのメソッドがユーザー・プロファイルに用意されています。
- AS400CertificateVldlUtil クラスでは、 証明書を管理するためのメソッドが妥当性検査リストに用意されています。

上記の 2 つのクラスは、AS400CertificateUtil を拡張したものです。 このクラスは、両方のサブクラスに共通なメソッドを定義している抽象基本クラスです。

AS400Certificate クラスでは、 証明書データの読み取りおよび書き込みを行うためのメソッドが用意されています。 データには、バイトの配列としてアクセスします。 Java 仮想マシン 1.2 の Java.Security パッケージには、 証明書の個々のフィールドを取得および設定するのに使用できるクラスが用意されています。

証明書のリスト表示

証明書リストを取得するには、Java プログラムで次のようにする必要があります。

1. AS400 オブジェクトを作成します。
2. 正しい証明書オブジェクトを構成します。 ユーザー・プロファイルにある証明書をリストする (AS400CertificateUserProfileUtil) 場合と、 妥当性検査リストにある証明書をリストする (AS400CertificateVldlUtil) 場合とでは、 それぞれ異なるオブジェクトを使用します。
3. 証明書の属性に基づいて選択基準を作成します。 AS400CertificateAttribute クラスには、 選択基準として使用される属性が入っています。 1 つ以上の属性オブジェクトによって、 リストに証明書を追加するために前もって満たしていなければならない基準が定義されます。 たとえば、特定のユーザーまたは企業の証明書しか入っていないリストもあります。
4. サーバー上にユーザー・スペースを作成してから、 証明書をそのユーザー・スペースに入れます。 リスト操作によって、大量のデータが生成されることがあります。 そのデータを Java プログラムで取り出せるようにするには、 あらかじめユーザー・スペースに入っていなければなりません。 証明書をユーザー・スペースに入れるためには、 listCertificates() メソッドを使用します。
5. 証明書をユーザー・スペースから検索するには、 getCertificates() メソッドを使用します。

以下の例は、妥当性検査リスト内の証明書を示しています。 これは、特定の人物に帰属する証明書だけをリストしています。

```
                    // Create an AS400 object.  The
                    // certificates are on this system.
    AS400 sys = new AS400("mySystem.myCompany.com");


                    // Create the certificate object.
    AS400CertificateVldlUtil certificateList =
            new AS400CertificateVldlUtil(sys, "/QSYS.LIB/MYLIB.LIB/CERTLIST.VLDL");


                    // Create the certificate attribute
                    // list. We only want certificates
                    // for a single person so the list
                    // consists of only one element.
    AS400CertificateAttribute[] attributeList = new AS400CertificateAttribute[1];
    attributeList[0] = new
AS400CertificateAttribute(AS400CertificateAttribute.SUBJECT_COMMON_NAME, "Jane Doe");


                    // Retrieve the list that matches
                    // the criteria. User space "myspace"
                    // in library "mylib" will be used
                    // for storage of the certificates.
                    // The user space must exist before
                    // calling this API.
    int count = certificateList.listCertificates(attributeList,
                                                 "/QSYS.LIB/MYLIB.LIB/MYSPACE.USRSPC");
```

```
                    // Retrieve the certificates from
                    // the user space.
     AS400Certificates[] certificates =
certificateList.getCertificates("/QSYS.LIB/MYLIB.LIB/MYSPACE.USRSPC", 0, 8);


                    // ... process the certificates
```

EnvironmentVariable クラス

EnvironmentVariable クラスおよび EnvironmentVariableList クラスを使用して、 iSeries システム・レベル環境変数にアクセスしてそれを設定することができます。

各変数には、システム名および環境変数名という固有 ID があります。 各環境変数は変数の内容が保管されている場所を示す CCSID (デフォルトでは現行ジョブ) に関連しています。

注: 環境変数とシステム値はしばしば同じ目的で使用されますが、 これらは異なるものです。 システム値にアクセスする方法についての詳細は、SystemValues を参照してください。

EnvironmentVariable オブジェクトを使用して、以下のアクションを環境変数に対して実行します。

- 名前の取得および設定
- システムの取得および設定
- 値の取得および設定 (これにより、CCSID の変更が可能になる)
- 値の最新表示

例: 環境変数の作成、設定、および取得

以下の例では、2 つの EnvironmentVariables を作成して、それらの値を設定および取得します。

```
// Create the iSeries system object.
AS400 system = new AS400("mySystem");
// Create the foreground color environment variable and set it to red.
EnvironmentVariable fg = new EnvironmentVariable(system, "FOREGROUND");
fg.setValue("RED");
// Create the background color environment variable and get its value.
EnvironmentVariable bg = new EnvironmentVariable(system, "BACKGROUND");
String background = bg.getValue();
```

例外

装置エラー、物理制限、プログラミング・エラー、またはユーザー入力エラーが起きたとき、 IBM Toolbox for Java のアクセス・クラスは例外を出します。 例外クラスは、エラーの発生箇所ではなく、起きたエラーのタイプに基づいています。

ほとんどの例外に含まれる情報は、以下の 3 つです。

- エラー・タイプ - 出される例外オブジェクトは、 エラーのタイプを示します。 同じタイプのエラーは、グループにまとめられて例外クラスになります。
- エラー詳細 - 例外には、 エラーの原因をさらに識別するための戻りコードが含まれています。 戻りコード値は、例外クラス内の固定情報です。
- エラー・テキスト - 例外には、 エラーを説明するテキスト文字列が含まれています。 この文字列は、クライアント Java 仮想マシンのロケールで変換されます。

以下の例は、例外をキャッチし、戻りコードを取得して、 例外テキストを表示する方法を示しています。

```
                    // ... all the setup work to delete
                    // a file on the server through the
                    // IFSFile class is done. Now try
                    // deleting the file.
try
{
   aFile.delete();

}

                    // The delete failed.
catch (ExtendedIOException e)
{
                    // Display the translated string
                    // containing the reason that the
                    // delete failed.
   System.out.println(e);

                    // Get the return code out of the
                    // exception and display additional
                    // information based on the return
```

```
                    // code.
    int rc = e.getReturnCode()

    switch (rc)
    {
       case ExtendedIOException.FILE_IN_USE:
          System.out.println("Delete failed, file is in use "):
          break;

       case ExtendedIOException.PATH_NOT_FOUND:
          System.out.println("Delete failed, path not found ");
          break;

                    // ... for every specific error you
                    // want to track

       default:
          System.out.println("Delete failed, rc = ");
          System.out.println(rc);
    }
}
```

FTP クラス

FTP クラスは、FTP 関数へのプログラマブル・インターフェースを提供します。java.runtime.exec() を使ったり、 別のアプリケーションで FTP コマンドを実行するようにユーザーに指示したりする必要はもはやありません。 つまり、アプリケーション内に FTP 関数を直接プログラミングすることができます。 このため、プログラム内で以下を行うことができます。

- FTP サーバーに接続する。
- サーバーにコマンドを送信する。
- ディレクトリー内のファイルをリストする。
- サーバーからファイルを取得し、 そして
- サーバーへのファイルの書き込みを行う。

たとえば、FTP クラスでは、 サーバー上のディレクトリーから一連のファイルをコピーできます。

```
FTP client = new FTP("myServer", "myUID", "myPWD");
client.cd("/myDir");
client.setDataTransferType(FTP.BINARY);
String [] entries = client.ls();

for (int i = 0; i < entries.length; i++)
{
  System.out.println("Copying " + entries[i]);
  try
  {
    client.get(entries[i], "c://ftptest//" + entries[i]);
  }
    catch (Exception e)
  {
    System.out.println("  copy failed, likely this is a directory");
  }
}

client.disconnect();
```

FTP は、多くの異なった FTP サーバーで動作する汎用インターフェースです。 ですから、サーバーのセマンティクスと一致させるのはプログラマーの仕事です。

FTP サブクラス

FTP クラスは汎用 FTP インターフェースですが、 AS400FTP サブクラスは、 サーバー上の FTP サーバー用に特別に作成されたものです。 つまり、このクラスは iSeries または AS/400e サーバー上の FTP サーバーのセマンティクスを理解するので、 プログラマーはそのことを気にする必要はありません。 たとえば、このクラスは保管ファイルをサーバーに転送するのに必要なさまざまなステップを理解して、 それらのステップを自動的に実行します。 さらに、AS400FTP は、 IBM Toolbox for Java のセキュリティー機能と連携します。 他の IBM Toolbox for Java クラスと同様、 AS400FTP は AS400 オブジェクトからシステム名、 ユーザー ID、およびパスワードを求めます。

以下の例は、保管ファイルをサーバーに置きます。 アプリケーションがデータ転送タイプをバイナリーに設定したり、 Toolbox CommandCall を使って保管ファイルを作成したりすることはないことに注意してください。 このファイルの拡張子は .savf なので、 AS400FTP クラスは書き込まれるファイルが保管ファイルであることを検出して、 以下のステップを自動的に行います。

```
AS400 system = new AS400();
AS400FTP ftp = new AS400FTP(system);
ftp.put("myData.savf", "/QSYS.LIB/MYLIB.LIB/MYDATA.SAVF");
```

統合ファイル・システム

統合ファイル・システム・クラスを使用すると、Java プログラムを使って、iSeries または AS/400e サーバーの統合ファイル・システム内のファイルを、 バイトのストリームあるいは文字のストリームとしてアクセスできるようになります。 java.io パッケージでは、 ファイルの出力および他の iSeries 機能が提供されていないため、 この統合ファイル・システム・クラスが作成されています。

IFSFile クラスで提供されているこの機能は、 java.io パッケージの file IO クラスで提供されている機能のスーパーセットです。 java.io FileInputStream、FileOutputStream、および RandomAccessFile にあるメソッドはすべて、 統合ファイル・システム・クラスに入っています。

これらのメソッドに加え、 このクラスには以下を実行するためのメソッドが含まれます。

- ファイルが使用中のときにそのファイルへのアクセスを拒否するファイル共用モードを指定する
- ファイルをオープン、作成、または置換するファイル作成モードを指定する
- ファイルが使用中のときにそのファイルの特定セクションへのアクセスを拒否するために、 ファイルのその部分をロックする
- ディレクトリーの内容をより効率的にリストする
- ディレクトリーの内容をキャッシュに入れて、 サーバーへの呼び出しを制限することにより、パフォーマンスを改善する
- サーバー・ファイル・システム上で使用可能なバイト数を判別する
- Java アプレットがサーバー・ファイル・システム内のファイルにアクセスできるようにする
- データをバイナリー・データではなくテキストとして読み書きする
- オブジェクトが QSYS.LIB ファイル・システムにあるとき、 ファイル・オブジェクトのタイプ (論理、物理、保管、その他) を判別する

Java プログラムでは、 この 統合ファイル・システム・クラスを介して iSeries 上のストリーム・ファイルに直接アクセ スすることができます。 Java プログラムで java.io パッケージを使用することもできますが、 その場合はクライアントのオペレーティング・システム側で出力のメソッドを提供しなければなりません。 たとえば、 Windows 95 あるいは Windows NT オペレーティング・システム上で Java プログラムを実行している場合、 java.io 呼び出しを iSeries へ転送するには、 iSeries Access for Windows の Network Drives 機能が必要です。 統合ファイル・システム・クラスを使用すると、iSeries Access for Windows は

必要ありません。

統合ファイル・システム・クラスの必須パラメーターは、 AS400 オブジェクトです。 このオブジェクトは、ファイルを含む iSeries システムを表します。 統合ファイル・システム・クラスを使用すると、 AS400 オブジェクトと iSeries が接続されます。 接続の管理については、接続の管理を参照してください。

統合ファイル・システム・クラスでは、 その統合ファイル・システムにあるオブジェクトの階層名が必要です。 パスの区切り文字としては、スラッシュを使用してください。 以下の例では、 ディレクトリー・パス DIR1/DIR2 にある FILE1 にアクセスする方法を示します。

```
/DIR1/DIR2/FILE1
```

統合ファイル・システム・クラスは、以下のとおりです。

統合ファイル・システム・クラス	説明
IFSFile	統合ファイル・システム内のファイルを表します。
IFSJavaFile	統合ファイル・システム内のファイルを表します (java.io.File を拡張する)。
IFSFileInputStream	iSeries 上のファイルからデータを読み取るときの入力ストリームを表します。
IFSTextFileInputStream	ファイルから読み取られる文字データのストリームを表します。
IFSFileOutputStream	iSeries ファイルへデータを書き込むときの出力ストリームを表します。
IFSTextFileOutputStream	ファイルに書き込まれる文字データのストリームを表します。
IFSRandomAccessFile	データを読み書きするときの iSeries 上のファイルを表します。
IFSFileDialog	ユーザーがファイル・システム内を移動し、 そのファイル・システム内にある特定ファイルを選択できるようにします。

例

IFSCopyFile の例は、 統合ファイル・システム・クラスを使って、1 つのディレクトリーから、 iSeries 上の別のディレクトリーにファイルをコピーする方法を示します。

ファイル・リストの例は、 統合ファイル・システム・クラスを使って、 iSeries

上のディレクトリーの内容をリスト表示する方法を示します。

IFSFile クラス

IFSFile クラスは、 iSeries システムの統合ファイル・システムにあるオブジェクトを表します。 IFSFile でのメソッドは、一般的には、オブジェクト上で行われる操作を表します。 IFSFileInputStream、IFSFileOutputStream、 および IFSRandomAccessFile を使用して、ファイルへの読み書きを行うことができます。 この IFSFile クラスを使用すると、 Java プログラムを使って以下のことを行えるようになります。

- オブジェクトが存在するかどうか、 およびそれがディレクトリーとファイルのどちらであるかを判別する。
- Java プログラムがファイルから読み取りを行うのか、 またはファイルへの書き込みを行うのかを判別する。
- ファイルの長さを判別する。
- オブジェクトの許可の判別およびオブジェクトの許可の設定を行う。
- ディレクトリーを作成する。
- ファイルまたはディレクトリーを削除する。
- ファイルまたはディレクトリーをリネームする。
- ファイルの最終変更日付を取得または設定する。
- ディレクトリーの内容をリストする。
- ディレクトリーの内容をリストし、 属性情報をローカル・キャッシュに保存する。
- システム上で使用可能なスペースを判別する。
- ファイル・オブジェクトが QSYS.LIB ファイル・システムにあるときに、 ファイル・オブジェクトのタイプを判別する。

list() メソッドまたは listFiles() メソッドのいずれかを使用して、ディレクトリー内のファイルのリストを取得することができます。

- listFiles() メソッドは、最初の呼び出しで各ファイルの情報を検索してキャッシュに入れます。 listFiles() を呼び出した後で、 他のメソッドを使用してファイルの詳細結果を照会すると、 情報がキャッシュから取り出されるため、パフォーマンスが改善されます。 たとえば、listFiles() で戻された IFSFile オブジェクトに対して isDirectory() 実行するのにサーバーを呼び出す必要がありません。
- list() メソッドは各ファイルについての情報をサーバーに対する別個の要求によって検索するため、 速度が遅くなり、要求するサーバー・リソースも多くなります。

注: listFiles() メソッドを使用する場合、キャッシュ内の情報が古くなっていく

ので、 listFiles() メソッドを再度呼び出してデータを最新表示してください。

例

以下の例では、IFSFile クラスを使用する方法を示します。

- 例: ディレクトリーを作成する
- 例: 暗号化を使用してエラーを追跡する
- 例: .txt の拡張子が付けられたファイルをリストする
- 例: listFiles() を使用して、 ディレクトリーの内容をリストする

IFSJavaFile クラス

IFSJavaFile は iSeries 統合ファイル・システム内のファイルを表すクラスで、java.io.File クラスを拡張したものです。 IFSJavaFile を使用することにより、java.io.File インターフェース用の iSeries 統合ファイル・システムにアクセスするファイルを作成することができます。

IFSJavaFile は java.io.File と互換性のある可搬インターフェースを作成し、java.io.File が使用するエラーと例外だけを使用します。 IFSJavaFile は java.io.File のセキュリティー・マネージャー機能を使用しますが、 java.io.File とは異なり、IFSJavaFile はセキュリティー機能を継続的に使用することです。

IFSJavaFile は、IFSFileInputStream および IFSFileOutputStream を指定して使用することができます。 java.io.FileInputStream および java.io.FileOutputStream はサポートしていません。

IFSJavaFile は IFSFile に基づいていますが、 インターフェースとしては IFSFile よりは java.io.File に近いといえます。 IFSFile は、IFSJavaFile クラスの代わりとして使用することができます。

list() メソッドまたは listFiles() メソッドのいずれかを使用して、 ディレクトリー内のファイルのリストを取得することができます。

- listFiles() メソッドは、最初の呼び出しで各ファイルの情報を検索してキャッシュし、 それ以降は各ファイルの情報がそのキャッシュから検索されるので、パフォーマンスが良くなります。
- list() メソッドは各ファイルについての情報を別個の要求によって検索するため、 検索速度が遅くなり、要求するサーバー・リソースも多くなります。

注: listFiles() では、キャッシュ内の情報が古くなっていくため、 データを最新表示する必要が生じることがあります。

以下の例では、IFSJavaFile クラスの使用方法を示します。

```
// Work with /Dir/File.txt on the system flash.
AS400 as400 = new AS400("flash");
IFSJavaFile file = new IFSJavaFile(as400, "/Dir/File.txt");


// Determine the parent directory of the file.
String directory = file.getParent();
```

```
// Determine the name of the file.
String name = file.getName();


// Determine the file size.
long length = file.length();


// Determine when the file was last modified.
Date date = new Date(file.lastModified());


// Delete the file.
if (file.delete() == false)
{
  // Display the error code.
  System.err.println("Unable to delete file.");
}


try
{
  IFSFileOutputStream os = new IFSFileOutputStream(file.getSystem(),
                               file,
                               IFSFileOutputStream.SHARE_ALL,
                               false);
  byte[] data = new byte[256];
  int i = 0;
  for (; i < data.length; i++)
  {
    data[i] = (byte) i;
    os.write(data[i]);
  }
  os.close();
}
catch (Exception e)
{
  System.err.println ("Exception: " + e.getMessage());
}
```

IFSFileInputStream

IFSFileInputStream クラスは、 サーバー上のファイルからデータを読み取るための入力ストリームを表します。 IFSFile クラスの場合と同様、メソッドは IFSFileInputStream にありますが、 これは java.io パッケージの FileInputStream にあるメソッドと重複します。 これらのメソッドに加え、 IFSFileInputStream には iSeries および AS/400e サーバー固有の追加メソッドがあります。 この IFSFileInputStream クラスを使用すると、 Java プログラムを使って以下のことを行えるようになります。

- 読み取りのためにファイルをオープンする。 このクラスはサーバー上ではファイルを作成しないため、ファイルが存在していなければなりません。 コンストラクターを使用すれば、ファイル共用モードを指定することができます。
- ストリーム内のバイト数を判別する。
- ストリームからバイトを読み取る。
- ストリーム内のバイトをスキップする。
- ストリーム内のバイトをロックまたはロック解除する。
- ファイルをクローズする。

java.io の FileInputStream での場合と同様、このクラスを使用すると、 Java プログラムはファイルからバイトのストリームを読み取れるようになります。 Java プログラムは、 ストリーム内のバイトをスキップするオプションだけを追加し、 順番にバイトを読み取ります。

以下の例では、IFSFileInputStream クラスの使用方法を示します。

```
                    // Create an AS400 object.
AS400 sys = new AS400("mySystem.myCompany.com");

                    // Open a file object that
                    // represents the file.
IFSFileInputStream aFile =
              new IFSFileInputStream(sys,"/mydir1/mydir2/myfile");

                    // Determine the number of bytes in
                    // the file.
int available = aFile.available();

                    // Allocate a buffer to hold the data
byte[] data = new byte[10240];
```

```
                    // Read the entire file 10K at a time
for (int i = 0; i < available; i += 10240)
{
    aFile.read(data);
}

                    // Close the file.
aFile.close();
```

FileInputStream のメソッドに加え、 IFSFileInputStream を使用すると Java プログラムで以下のオプションを指定できるようになります。

- ストリーム内のバイトをロックおよびロック解除する。 詳細については、IFSKey を参照してください。
- ファイルのオープン時に共用モードを指定する。 詳細については、共用モードを参照してください。

IFSTextFileInputStream クラス

IFSTextFileInputStream クラスは、 ファイルから読み取られる文字データのストリームを表します。 IFSTextFileInputStream オブジェクトから読み取られたデータは、 Java String オブジェクトの形式で Java プログラムに提供されるため、常に Unicode となります。 このファイルをオープンすると、IFSTextFileInputStream オブジェクトは、 ファイル内のデータの CCSID を判別します。 入れられているデータが Unicode 以外でエンコードされている場合、 IFSTextFileInputStream オブジェクトは、データを Java プログラムに送る前に、 ファイルのエンコードから Unicode に変換します。 データを変換できない場合、 UnsupportedEncodingException が送出されます。

以下の例では、IFSTextFileInputStream を使用する方法を示します。

```
                    // Work with /File on the system
                    // mySystem.
AS400 as400 = new AS400("mySystem");
IFSTextFileInputStream file = new IFSTextFileInputStream(as400, "/File");

                    // Read the first four characters of
                    // the file.
String s = file.read(4);

                    // Display the characters read. Read
                    // the first four characters of the
                    // file. If necessary, the data is
                    // converted to Unicode by the
                    // IFSTextFileInputStream object.
System.out.println(s);

                    // Close the file.
file.close();
```

IFSFileOutputStream

IFSFileOutputStream クラスは、 サーバー上のファイルにデータを書き込むための出力ストリームを表します。 IFSFile クラスの場合と同様、 メソッドは IFSFileOutputStream にありますが、 これは java.io パッケージの FileOutputStream にあるメソッドと重複します。 IFSFileOutputStream にも、サーバーに固有の追加メソッドがあります。 この IFSFileOutputStream クラスを使用すると、 Java プログラムを使って以下のことを行えるようになります。

- 書き込みのためにファイルをオープンする。 ファイルがすでに存在している場合、置き換えられます。 コンストラクターを使用すれば、ファイル共用モード、および既存のファイルの内容が追加されたかどうかを指定することができます。
- バイトをストリームに書き込む。
- ストリームに書き込まれたバイトをディスクにコミットする。
- ストリーム内のバイトをロックまたはロック解除する。
- ファイルをクローズする。

java.io の FileOutputStream での場合と同様、このクラスを使用すると、 Java プログラムはバイトのストリームをファイルへ書き込めるようになります。

以下の例では、IFSFileOutputStream クラスの使用方法を示します。

```
                    // Create an AS400 object
AS400 sys = new AS400("mySystem.myCompany.com");

                    // Open a file object that
                    // represents the file.
IFSFileOutputStream aFile =
              new IFSFileOutputStream(sys,"/mydir1/mydir2/myfile");

                    // Write to the file
byte i = 123;
aFile.write(i);

                    // Close the file.
aFile.close();
```

FileOutputStream のメソッドに加え、 IFSFileOutputStream を使用すると Java プログラムで以下のオプションを指定できるようになります。

- ストリーム内のバイトをロックおよびロック解除する。 詳細については、IFSKey を参照してください。

- ファイルのオープン時に共用モードを指定する。 詳細については、共用モードを参照してください。

IFSTextFileOutputStream クラス

IFSTextFileOutputStream クラスは、 ファイルに書き込まれる文字データのストリームを表します。 この IFSTextFileOutputStream オブジェクトに提供されるデータは、 Java String オブジェクト形式であるため、入力は常に Unicode になります。 しかし、 IFSTextFileOutputStream オブジェクトは、 データをファイルに書き込むときには、 そのデータを別の CCSID に変換することができます。 デフォルトの動作は Unicode 文字をファイルに書き込むことですが、 Java プログラムはファイルをオープンする前にターゲット CCSID を設定することができます。 この場合、IFSTextFileOutputStream オブジェクトは文字をファイルに書き込む前に、 それらの文字を Unicode から指定した CCSID に変換することができます。 データを変換できない場合、 UnsupportedEncodingException が送出されます。

以下の例では、IFSTextFileOutputStream の使用方法を示します。

```
                    // Work with /File on the system
                    // mySystem.
AS400 as400 = new AS400("mySystem");
IFSTextFileOutputStream file = new IFSTextFileOutputStream(as400, "/File");

                    // Write a String to the file.
                    // Because no CCSID was specified
                    // before writing to the file,
                    // Unicode characters will be
                    // written to the file. The file
                    // will be tagged as having Unicode
                    // data.
file.write("Hello world");

                    // Close the file.
file.close();
```

IFSRandomAccessFile

IFSRandomAccessFile クラスは、 データを読み書きするためのサーバー上のファイルを表します。 Java プログラムは、 データを順番にあるいはランダムに読み書きすることができます。 IFSFile の場合と同様、 メソッドは IFSRandomAccessFile にありますが、 これは java.io パッケージの RandomAccessFile にあるメソッドと重複します。 これらのメソッドに加え、 IFSRandomAccessFile には iSeries または AS/400e サーバー固有の追加メソッドがあります。 Java プログラムは IFSRandomAccessFile を使用して以下を行えます。

- 読み取り、書き込み、または読み取り/書き込みアクセスのために、ファイルをオープンする。 Java プログラムでは、 任意でファイル共用モードおよび存在オプションを指定することができます。
- 現行オフセットのデータをファイルから読み取る。
- 現行オフセットのデータをファイルに書き込む。
- ファイルの現行オフセットを取得または設定する。
- ファイルをクローズする。

以下の例では、IFSRandomAccessFile クラスを使用して、 1K の間隔で 4 バイトをファイルに書き込む方法を示します。

```
                    // Create an AS400 object.
AS400 sys = new AS400("mySystem.myCompany.com");

                    // Open a file object that represents
                    // the file.
IFSRandomAccessFile aFile =
              new IFSRandomAccessFile(sys,"/mydir1/myfile", "rw");

                    // Establish the data to write.
byte i = 123;

                    // Write to the file 10 times at 1K
                    // intervals.
for (int j=0; j<10; j++)
{
                    // Move the current offset.
   aFile.seek(j * 1024);

                    // Write to the file. The current
                    // offset advances by the size of
```

```
                    // the write.
    aFile.write(i);
}

                    // Close the file.
aFile.close();
```

java.io RandomAccessFile のメソッドに加え、 IFSRandomAccessFile を使用すると Java プログラムで以下のオプションを指定できるようになります。

- 書き出しバイト数をディスクにコミットする。
- ファイル内のバイトをロックまたはロック解除する。
- ストリーム内のバイトをロックおよびロック解除する。 詳細については、IFSKey を参照してください。
- ファイルのオープン時に共用モードを指定する。 詳細については、共用モードを参照してください。
- ファイルのオープン時に存在オプションを指定する。 Java プログラムでは、以下のいずれかを選択することができます。
  - ファイルが存在する場合にはオープンし、 存在しない場合にはファイルを作成する。
  - ファイルが存在する場合には置換し、存在しない場合にはファイルを作成する。
  - ファイルが存在する場合にはオープンを失敗させ、 存在しない場合にはファイルを作成する。
  - ファイルが存在する場合にはオープンし、 存在しない場合にはオープンを失敗させる。
  - ファイルが存在する場合には置換し、 存在しない場合にはオープンを失敗させる。

IFSFileDialog

IFSFileDialog クラスを使用すると、 ユーザーはファイル・システムを走査し、目的のファイルを選択できます。 このクラスでは、 iSeries または AS/400e サーバー上の統合ファイル・システムにあるディレクトリーおよびファイルのリストを走査するために、IFSFile クラスを使用します。 このクラスのメソッドにより、 Java プログラムはテキスト付きの押しボタンをダイアログに表示し、 フィルターを設定できるようになります。 Swing 1.1 に準拠する IFSFileDialog クラスも使えることに注意してください。

FileFilter クラスによって、 フィルターを設定することができます。 ユーザーがダイアログにあるファイルを選択する場合、 getFileName() メソッドを使用して、 選択したファイルの名前を知ることができます。 getAbsolutePath() メソッドを使用すると、 選択したファイルのパスと名前を知ることができます。

以下の例では、2 種類のフィルターが含まれているダイアログを設定する方法、 およびテキスト付きの押しボタンをダイアログに表示する方法を示します。

```
                    // Create an AS400 object.
AS400 sys = new AS400("mySystem.myCompany.com");

                    // Create a dialog object setting
                    // the text of the dialog's title
                    // bar and the server to traverse.
IFSFileDialog dialog = new IFSFileDialog(this, "Title Bar Text", sys);

                    // Create a list of filters then set
                    // the filters in the dialog. The
                    // first filter will be used when
                    // the dialog is first displayed.
FileFilter[] filterList = {new FileFilter("All files (*.*)", "*.*"),
                           new FileFilter("HTML files (*.HTML", "*.HTM")};

dialog.setFileFilter(filterList, 0);

                    // Set the text on the buttons of
                    // the dialog.
dialog.setOkButtonText("Open");
dialog.setCancelButtonText("Cancel");

                    // Show the dialog. If the user
                    // selected a file by pressing the
                    // Open button, get the file the
                    // user selected and display it.
if (dialog.showDialog() == IFSFileDialog.OK)
   System.out.println(dialog.getAbsolutePath());
```

IFSKey クラス

Java プログラムを使用することにより、 他のプログラムが特定のファイルに同時にアクセスできるようになる場合、 その Java プログラムは一定期間そのファイル内のバイトをロックすることができます。 その期間内は、 プログラムはそのファイルの該当セクションを排他的に使用します。 ロックに成功すると、 統合ファイル・システム・クラスは IFSKey オブジェクトを戻します。 このオブジェクトは、どのバイトをアンロックするかを示す unlock() メソッドに渡されます。 このファイルをクローズすると、 システムはファイル上で有効なロックをすべて解除します (プログラムが解除しなかったそれぞれのロックについては、 システムが解除します)。

以下の例は、IFSKey クラスの使用方法を示しています。

```
                    // Create an AS400 object.
AS400 sys = new AS400("mySystem.myCompany.com");

                    // Open an input stream. This
                    // constructor opens with share_all
                    // so other programs can open this
                    // file.
IFSFileInputStream aFile =
              new IFSFileInputStream(sys,"/mydir1/mydir2/myfile");

                    // Lock the first 1K bytes in the
                    // file. Now no other instance can
                    // read these bytes.
IFSKey key = aFile.lock(1024);

                    // Read the first 1K of the file.
byte data[] = new byte[1024];
aFile.read(data);

                    // Unlock the bytes of the file.
aFile.unlock(key);

                    // Close the file.
aFile.close();
```

ファイル共用モード

Java プログラムでは、 ファイルのオープン時に共用モードを指定することができます。 プログラムは、他のプログラムにファイルを一度にオープンさせたり、 そのファイルに排他的にアクセスさせたりすることができます。

以下の例は、ファイル共用モードの指定方法を示しています。

```
                    // Create an AS400 object.
AS400 sys = new AS400("mySystem.myCompany.com");

                    // Open a file object that
                    // represents the file. Since this
                    // program specifies share-none, all
                    // other open attempts fail until
                    // this instance is closed.
IFSFileOutputStream aFile =
              new IFSFileOutputStream(sys,
                                      "/mydir1/mydir2/myfile",
                                      IFSFileOutputStream.SHARE_NONE,
                                      false);

                    // ... perform operations on the
                    // file.

                    // Close the file. Now other open
                    // requests succeed.
aFile.close();
```

JavaApplicationCall

JavaApplicationCall クラスを使用すれば、クライアントがサーバー JVM を使用して、 サーバーに常駐する Java プログラムを実行することができます。

クライアントからサーバーへの接続を確立した後、 JavaApplicationCall クラスで以下のものを構成できます。

1. setClassPath() メソッドで、サーバー上の CLASSPATH 環境変数を設定する。
2. setParameters() メソッドを使ってプログラムのパラメーターを定義する。
3. run() を使ってプログラムを実行する。
4. クライアントからの入力を Java プログラムに送信する。 Java プログラムは、 sendStandardInString() メソッドを使って設定される標準入力を介して入力を読み取ります。 getStandardOutString() と getStandardErrorString() を使って、 標準出力および標準エラーを Java プログラムからクライアントに宛先変更することができます。

JavaApplicationCall は、Java プログラムから呼び出すクラスです。 ただし、IBM Toolbox for Java には、 サーバーに常駐する Java プログラムを呼び出すためのユーティリティーも用意されています。 これらのユーティリティーは、ご使用のワークステーションから実行できる完全な Java プログラムです。 詳細については、RunJavaApplication クラスを参照してください。

例

この例では、 クライアントからサーバー上で「Hello World!」を出力するプログラムを実行する方法を示します。

JDBC

JDBC(TM) は、Java プログラムを広範囲のデータベースに接続するための、Java プラットフォームに組み込ま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) です。

IBM Toolbox for Java JDBC ドライバーにより、JDBC API インターフェースを使用して、 構造化照会言語 (SQL) ステートメントを発行し、サーバー上のデータベースからの結果を処理することができます。 »'ネイティブ' JDBC ドライバーと呼ばれる IBM Developer Kit for Java の JDBC ドライバーを使用することもできます。

- Java プログラムが、あるシステム上にあり、データベース・ファイルが別のシステム上にある場合、 クライアント/サーバー環境の場合と同様に、Toolbox JDBC ドライバーを使用します。
- Java プログラムとデータベース・ファイルが同じ iSeries システム上にある場合、 ネイティブ JDBC ドライバーを使用します。

今回の機能強化の詳細については、V5R2 の新機能および Toolbox for Java JDBC サポートの拡張を参照してください。 «

JDBC のさまざまなバージョン

JDBC API にはさまざまなバージョンが存在します。 IBM Toolbox for Java JDBC ドライバーは次のバージョンをサポートします。

- JDBC 1.2 API (java.sql パッケージ) は、Java Platform 1.1 core API および JDK 1.1 に含まれます。
- JDBC 2.1 core API (java.sql パッケージ) は、Java 2 Platform、 Standard Edition (J2SE) および Java 2 Platform Enterprise Edition (J2EE) の両方に含まれます。
- JDBC 2.0 Optional Package API (javax.sql パッケージ) は J2EE に含まれており、 separate download from Sun として使用可能です。 これらの拡張機能は、以前は JDBC 2.0 Standard Extension API という名前でした。
- » JDBC 3.0 API (java.sql および javax.sql パッケージ) は J2SE、バージョン 1.4 に組み込まれています。 «

サポートされるインターフェース

以下の表は、サポートされる JDBC インターフェースと、 それらを使用するのに必要な API をリストしています。

サポートされる JDBC インターフェース	必要な API

Blob インターフェースは、 バイナリー・ラージ・オブジェクト (BLOB) へのアクセスを提供します。	JDBC 2.1 core
CallableStatement は、SQL ストアード・プロシージャーを実行します。	JDK 1.1
Clob は、 キャラクター・ラージ・オブジェクト (CLOB) へのアクセスを提供します。	JDBC 2.1 core
Connection は、特定のデータベースへの接続を確立します。	JDK 1.1
» ConnectionPool は、接続オブジェクトのプールを表します。	JDBC 2.0 Optional Package «
ConnectionPoolDataSource は、プールに入れられた AS400JDBCPooledConnection オブジェクトを表します。	JDBC 2.0 Optional Package
DatabaseMetaData は、 データベース全体に関する情報を提供します。	JDK 1.1
DataSource は、 データベース接続用のファクトリーを表します。	JDBC 2.0 Optional Package
Driver は、接続を作成し、 ドライバーのバージョンに関する情報を戻します。	JDK 1.1
» ParameterMetaData は、 PreparedStatement オブジェクトのパラメーターのタイプおよびプロパティーに関する情報を取得できるようにします。	JDBC 3.0 API «
PreparedStatement は、コンパイルされた SQL ステートメントを実行します。	JDK 1.1
ResultSet は、 SQL 照会または DatabaseMetaData カタログ・メソッドを実行して生成されるデータのテーブルへのアクセスを提供します。	JDK 1.1
ResultSetMetaData は、特定の ResultSet についての情報を提供します。	JDK 1.1
RowSet は、ResultSet をカプセル化する行のセットです。	JDBC 2.0 Optional Package
» Savepoint は、トランザクションのより緻密な制御を提供します。	JDBC 3.0 API «
Statement は、SQL ステートメントを実行し、 結果を取得します。	JDK 1.1
XAConnection は、 グローバル XA トランザクションに参加するデータベース接続です。	JDBC 2.0 Optional Package

XAResource は、 XA トランザクションで使用するためのリソース・マネージャーです。	JDBC 2.0 Optional Package

簡単に参照できるように、 JDBC プロパティーをリストするテーブルが含まれています。

例

以下の例では、IBM Toolbox for Java JDBC ドライバーの使用方法を示します。

- JDBC ドライバーを使用してテーブルを作成し、そこにデータを挿入する
- JDBC ドライバーを使用してテーブルを照会し、その内容を出力する

»

Toolbox for Java JDBC サポートの拡張

OS/400 バージョン 5 リリース 2 用に拡張された JDBC 機能には以下のものがあります。

- 'FOR UPDATE' の制限の撤廃
- データ切り捨ての変更
- 名前による列およびパラメーターの取得と変更
- 自動生成されたキーの検索
- SQL 挿入ステートメントがバッチで実行されている場合のパフォーマンスの向上
- ResultSet.getRow() の拡張されたサポート
- 列名での大文字小文字混合の使用に関する改良されたサポート
- Statements、CallableStatements、 および PreparedStatements 用の保持能力の指定
- 拡張されたトランザクション分離サポート

'FOR UPDATE' の制限の撤廃

更新可能カーソルを保証するために、SELECT ステートメントの FOR UPDATE を指定する必要はもうありません。 OS/400 の V5R1 以降のバージョンに接続する場合、 Toolbox for Java はステートメントの作成時に渡される並行性をすべて受け入れます。 並行性を指定しない場合、引き続きデフォルトは読み取り専用カーソルです。

データ切り捨ては、 切り捨てられた文字データがデータベースに書き込まれた時にのみ例外を出します。

現在 Toolbox for Java のデータ切り捨ての規則は、 IBM Developer Kit for Java の JDBC ドライバーの規則と同じです。 詳細については、IBM Toolbox for Java の JDBC プロパティーを参照してください。

名前による列およびパラメーターの取得と変更

新しいメソッドを使用することにより、 ResultSet の列名によって情報を取得して更新し、 CallableStatement のパラメーター名によって情報を取得して設定できるようになります。 たとえば、ResultSet において、これまでは以下のものを使用していました。

```
ResultSet rs = statement.executeQuery( SELECT * FROM MYCOLLECTION/MYTABLE );
rs.getString(1);
```

現在は、次のように使用できます。

```
ResultSet rs = statement.executeQuery( SELECT * FROM MYCOLLECTION/MYTABLE );
rs.getString( 'STUDENTS' );
```

インデックスによってパラメーターに接続すると、 名前によって接続するより良いパフォーマンスが得られるということに注意してください。 CallableStatement に設定するのにもパラメーター名を指定できます。 CallableStatement でこれまでは、次のようにしてきたかもしれません。

```
CallableStatement cs = connection.prepareCall( CALL MYPGM (?) );
cs.setString( 1 );
```

現在は、次のように使用できます。

```
CallableStatement cs = connection.prepareCall( CALL MYPGM (?) );
cs.setString( 'PARAM_1' );
```

これらの新しいメソッドを使用するには、JDBC 3.0 以降、 および Java 2 Platform、バージョン 1.4 (Standard または Enterprise Edition) が必要です。

自動生成されたキーの検索

AS400JDBCStatement の getGeneratedKeys() メソッドは、 Statement オブジェクトの実行の結果として作成されるすべての自動生成されたキーを検索します。 Statement オブジェクトがキーを全く生成しない場合、空の ResultSet オブジェクトが戻されます。 現在、サーバーは 1 つの自動生成された

キーのみの戻りをサポートします (最後に挿入された行のキー)。 以下の例では、テーブルに値を挿入し、その後自動生成されたキーを取得する方法を示します。

```
Statement s = statement.executeQuery
     ("INSERT INTO MYSCHOOL/MYSTUDENTS (FIRSTNAME) VALUES ('JOHN'");
ResultSet rs = s.getGeneratedKeys();
     // Currently the iSeries server supports returning only one auto-generated
     // key -- the key for the last inserted row.
rs.next();
String autoGeneratedKey = rs.getString(1);
     // Use the auto-generated key, for example, as the primary key in another table
```

自動生成されたキーを検索するには、JDBC 3.0 以降、 および Java 2 Platform、バージョン 1.4 (Standard または Enterprise Edition) が必要です。 自動生成されたキーを検索するには、OS/400 の V5R2 以降のバージョンへ接続することも必要です。

SQL 挿入ステートメントがバッチで実行されている場合のパフォーマンスの向上

バッチでの SQL 挿入ステートメントのパフォーマンスが向上しました。 AS400JDBCStatement、 AS400JDBCPreparedStatement、 および AS400JDBCCallableStatement で使用可能な別の addBatch() メソッドを使用することにより、SQL ステートメントをバッチで実行します。 拡張されたバッチ・サポートは、挿入の要求にのみ効力を持ちます。 たとえば、いくつかの挿入を処理するバッチ・サポートの使用は、サーバーへの 1 回のパスだけが関係します。 しかし、挿入して更新、さらに削除を処理するバッチ・サポートの使用は、個々の要求を別々に送信します。

バッチ・サポートを使用するには、JDBC 2.0 以降、 および Java 2 Platform、バージョン 1.2 (Standard または Enterprise Edition) が必要です。

ResultSet.getRow() の拡張されたサポート

これまでは、IBM Toolbox for Java JDBC ドライバーは、 ResultSet の getRow() メソッドのサポートにのみ限定されていました。 特に、現在行番号で作成された負の値を伴う ResultSet.last()、 ResultSet.afterLast()、 および ResultSet.absolute() は利用不能でした。 これまでの制限は解除され、このメソッドは完全に機能するようになります。

列名での大文字小文字混合の使用

Toolbox for Java メソッドは、ユーザーによって提供される列名、 またはデータベース・テーブル上の名前を用いてアプリケーションによって提供される列名のどちらかと一致する必要があります。 いずれにしても、列名が引用符で囲まれていない場合、 Toolbox for Java はサーバーの名前とマッチングする前にその名前を大文字に変更します。 列名が引用符が囲まれている場合、サーバー上の名前と正確に一致している必要があります。 そうでない場合、Toolbox for Java は例外を出します。

作成された Statements、CallableStatements、 および PreparedStatements の保持能力の指定

AS400JDBCConnection での新しいメソッドを使用することにより、作成した Statements、CallableStatements、 および PreparedStatements の保持能力を指定できるようになります。 保持能力 (Holdability) により、トランザクションをコミットする際にカーソルをオープンのままにしておくのか、 クローズするのかを判別します。 これで、接続オブジェクトとは異なる保持能力を有するステートメントを持つことができます。 さらに、接続オブジェクトは複数のオープン・ステートメントを持つことができ、 それぞれは別個に指定された保持能力があります。 コミットを呼び出すと、各ステートメントはそのステートメント用に指定された保持能力に応じて処理されます。

保持能力は以下の優先順位で引き出されます。

1. Connection クラス・メソッド、createStatement()、prepareCall()、 または prepareStatement() を使用することによってステートメント作成で指定された保持能力。
2. Connection.setHoldability(int) を使用して指定された保持能力。
3. Toolbox for Java の JDBC カーソル保持プロパティー によって指定された保持能力 (1. または 2. のメソッドが使用されない場合)。

これらのメソッドを使用するには、JDBC 3.0 以降、 および Java 2 Platform、バージョン1.4 (Standard または Enterprise Edition) が必要です。 また OS/400 の V5R1 以前のバージョンを実行しているサー

バーは、 JDBC cursor hold プロパティーによって指定された保持能力のみを使用できます。

拡張されたトランザクション分離サポート

現在 IBM Toolbox for Java の JDBC ドライバーは、 接続が作成された後にトランザクション分離の *NONE のレベルへの切り替えをサポートします。 V5R2 までは、 接続が作成された後に *NONE に切り替えられると Toolbox for Java の JDBC ドライバーは例外を出していました。 «

IBM Toolbox for Java の JDBC プロパティー

JDBC を使ってサーバー・データベースに接続するときに、多数のプロパティーを指定することができます。 すべてのプロパティーは任意選択であり、URL の一部として指定しても、 または java.util.Properties オブジェクト内に指定してもかまいません。 URL とプロパティー・オブジェクトの両方にプロパティーを設定すると、URL の値が使用されます。

注: 以下のリストには DataSource プロパティーは含まれません。

以下の表は、このドライバーで認識されているさまざまな接続プロパティーを示しています。 これらのプロパティーの中には、パフォーマンスに影響を与えるものがありますが、 サーバー・ジョブ属性であるものもあります。 この表では、プロパティーを以下のカテゴリー別に編成しています。

一般プロパティー

一般プロパティーは、ユーザー、パスワード、 およびサーバーに接続するのにプロンプトが必要かどうかを指定するシステム属性です。

一般プロパティー	説明	必須かどうか	選択項目	デフォルト値
"password"	サーバーへの接続のためのパスワードを指定します。 これを指定しないと、ユーザーに対してプロンプトが表示されます。ただし、 "prompt" プロパティーが "false" に設定されていない場合に限ります。 設定されている場合、 接続しようとしても失敗します。	いいえ	サーバーのパスワード	(ユーザーに対してプロンプトが表示されます。)
"prompt"	サーバーに接続するのにユーザー名またはパスワードが必要な場合に、 ユーザーに対してプロンプトを表示するかどうかを指定します。 ユーザーにプロンプトを表示しないと接続を確立できない場合に、 このプロパティーが "false" に設定されていると、接続しようとしても失敗します。	いいえ	"true" "false"	"true"

"user"	サーバーに接続するためのユーザー名を指定します。これを指定しないと、ユーザーに対してプロンプトが表示されます。ただし、"prompt" プロパティーが "false" に設定されていない場合に限ります。設定されている場合、 接続しようとしても失敗します。	いいえ	サーバーのユーザー	(ユーザーに対してプロンプトが表示されます。)

サーバーのプロパティー

サーバーのプロパティーは、トランザクション、ライブラリー、およびデータベースを制御する属性を指定します。

サーバーのプロパティー	説明	必須かどうか	選択項目	デフォルト値
"cursor hold"	複数のトランザクションにまたがってカーソルを保留するかどうかを指定します。 このプロパティーを "true" に設定すると、 トランザクションのコミットまたはロールバックが完了してもカーソルはクローズされません。 その作業単位中に獲得したすべてのリソースは保留されたままになりますが、暗黙で獲得された特定の行およびオブジェクトに対するロックは解放されます。	いいえ	"true" "false"	"true"
»"cursor sensitivity"	データベースから要求されるカーソル感度を指定します。 動作は resultSetType に依存します。 ● ResultSet.TYPE_FORWARD_ONLY または ResultSet.TYPE_SCROLL_SENSITIVE は、Java プログラムがどのカーソル感度をデータベースから要求するのかをこのプロパティーの値で制御することを示します。 ● ResultSet.TYPE_SCROLL_INSENSITIVE の場合、このプロパティーは無視されます。 V5R1 および 旧バージョンの OS/400 を実行しているシステムに接続する場合、 このプロパティーは無視されます。	いいえ	"asensitive" "insensitive" "sensitive"	"asensitive"«

»"database name"	独立補助記憶域プールに保管されているものも含め、 接続用に使用するデータベースを指定します。 このプロパティーは、OS/400 の V5R2 以降のバージョンへ接続する場合にのみ適用されます。 データベース名を指定する場合、サーバーの関連データベース・ディレクトリーに存在する名前でなければなりません。 以下の基準により、アクセスするデータベースを判別します。 ● このプロパティーがデータベースを指定する場合、指定されたデータベースを使用します。 指定されたデータベースが存在しない場合、接続は失敗します。 ● このプロパティーがデータベース名として *SYSBAS を指定する場合、 システムのデフォルト・データベースを使用します。 ● このプロパティーが省略される場合、 ユーザー・プロファイルのジョブ記述で指定されたデータベース名を使用します。 ジョブ記述がデータベース名を指定していない場合、システムのデフォルト・データベースを使用します。	いいえ	データベース名 "*SYSBAS"	ユーザー・プロファイルのジョブ記述で指定されたデータベース名を使用します。 ジョブ記述がデータベース名を指定していない場合、システムのデフォルト・データベースを使用します。 «
"libraries"	サーバー・ジョブのライブラリー・リストに追加、またはそれと置き換えたいライブラリーを 1 つ以上指定し、 デフォルト・ライブラリーを任意選択で設定します (デフォルト・スキーマ)。 ライブラリー・リスト サーバーは指定されたライブラリーを使用して、修飾なしのストアード・プロシージャー名を解決し、ストアード・プロシージャーはそれらのライブラリーを使用して、修飾なしの名前を解決します。 複数のライブラリーを指定するには、コンマまたはスペースで個々の項目を区切ってください。 サーバー・ジョブの現行ライブラリー・リストのために、 *LIBL をプレースホルダーとして使用することができます。 最初の項目が *LIBL の場合、 指定されたライブラリーがサーバー・ジョブの現行ライブラリー・リストに加えられます。 *LIBL を使用しない場合、 指定されたライブラリーがサーバー・ジョブの現行ライブラリー・リストを置き換えます。 デフォルト・スキーマ サーバーはデフォルト・スキーマを使用して、SQL ステートメント内の修飾なしの名前を解決します。 たとえば、"SELECT * FROM MYTABLE" というステートメントでは、 サーバーは MYTABLE のデフォルト・スキーマのみを調べます。 接続 URL にデフォルト・スキーマを指定できます。 接続 URL にデフォルト・スキーマを指定しない場合、 SQL ネーミングまたはシステム・ネーミングのどちらを使用するかによって、下記の条件が当てはまりま	いいえ	コンマまたはスペースで区切ったサーバー・ライブラリーのリスト	"*LIBL"

	す。 ● SQL ネーミング 接続 URL にデフォルト・スキーマを指定しない場合 ❍ 最初の項目は（*LIBL でない限り）デフォルト・スキーマになる。 ❍ 最初の項目が *LIBL の場合、2 番目の項目がデフォルト・スキーマになる。 ❍ このプロパティーを設定しない場合、またはこのプロパティーが *LIBL のみを含む場合、 ユーザー・プロファイルがデフォルト・スキーマになる。 ● システム・ネーミング 接続 URL にデフォルト・スキーマを指定しない場合 ❍ デフォルト・スキーマは設定されず、サーバーは指定されたライブラリーを使用して、修飾なしの名前を検索する。 ❍ このプロパティーを設定しない場合、またはこのプロパティーが *LIBL のみを含む場合、 サーバーはサーバー・ジョブの現行ライブラリー・リストを使用して、修飾なしの名前を検索する。			
"transaction isolation"	デフォルトのトランザクション分離を指定します。	いいえ	"none" "read uncommitted" "read committed" "repeatable read" "serializable"	"read uncommitted"

形式のプロパティー

形式のプロパティーは、SQL ステートメントで使用される日時形式、日付および数値区切り、 またテーブルの命名規則を指定します。

形式のプロパティー	説明	必須かどうか	選択項目	デフォルト値

"date format"	SQL ステートメント内の日付リテラルで使用される日付の形式を指定します。	いいえ	"mdy" "dmy" "ymd" "usa" "iso" "eur" "jis" "julian"	(サーバー・ジョブ)
"date separator"	SQL ステートメント内の日付リテラルで使用される日付区切り文字を指定します。"date format" プロパティーが "julian"、"mdy"、"dmy"、 または "ymd" に設定されていないと、このプロパティーには効力はありません。	いいえ	"/" (スラッシュ) "-" (ダッシュ) "." (ピリオド) "," (コンマ) "b" (スペース)	(サーバー・ジョブ)
"decimal separator"	SQL ステートメント内の数値リテラルで使用される 10 進数区切り文字を指定します。	いいえ	"." (ピリオド) "," (コンマ)	(サーバー・ジョブ)
"naming"	テーブルを参照する際に使用する命名規則を指定します。	いいえ	"sql" (schema.table 内のとおり) "system" (schema/table 内のとおり)	"sql"
"time format"	SQL ステートメント内の時刻リテラルで使用される時刻形式を指定します。	いいえ	"hms" "usa" "iso" "eur" "jis"	(サーバー・ジョブ)
"time separator"	SQL ステートメント内の時刻リテラルで使用される時刻区切り文字を指定します。"time format" プロパティーが "hms" に設定されていないと、このプロパティーには効力はありません。	いいえ	":" (コロン) "." (ピリオド) "," (コンマ) "b" (スペース)	(サーバー・ジョブ)

パフォーマンスのプロパティー

パフォーマンスのプロパティーは、 パフォーマンスに影響するキャッシュ、データ変換、データ圧縮、および事前取り出しを含む属性です。

パフォーマンスのプロパティー	説明	必須かどうか	選択項目	デフォルト値

"big decimal"	パック 10 進数とゾーン 10 進数の変換に中間の java.math.BigDecimal オブジェクトを使用するかどうかを指定します。 このプロパティーを "true" に設定すると、JDBC 仕様に従って、 パック 10 進数とゾーン 10 進数の変換に中間の java.math.BigDecimal オブジェクトが使用されます。 このプロパティーを "false" に設定すると、 パック 10 進数とゾーン 10 進数の変換に中間オブジェクトは使用されません。 つまり、そのような値は直接 Java 値に (または Java 値から) 変換されます。 この場合の変換のほうが早いですが、 JDBC 仕様に定められている変換とデータ切り捨てのすべての規則が守られるとは限らない可能性があります。	いいえ	"true" "false"	"true"
"block criteria"	データをレコード・ブロック単位でサーバーから取り出すための基準を指定します。 このプロパティーに非ゼロ値を指定すると、サーバーとの通信の頻度が減るので、 パフォーマンスが高まります。 カーソルを後続の更新でも使用する場合はレコード・ブロックがオフになっていることを確認してください。 オフになっていないと、更新される行は必ずしも現在行であるとは限りません。	いいえ	"0" (レコード・ブロックを指定しない) "1" (FOR FETCH ONLY が指定されている場合はブロック化) "2" (FOR UPDATE が指定されていない場合はブロック化)	"2"
"block size"	サーバーから取り出してクライアントのキャッシュに入れるブロックのサイズ (K バイト単位) を指定します。 "block criteria" プロパティーが非ゼロでないと、このプロパティーには効力はありません。 ブロック・サイズを大きくするとサーバーとの通信の頻度が減るので、 パフォーマンスが高まる可能性があります。	いいえ	"0" "8" "16" "32" "64" "128" "256" "512"	"32"
"data compression"	ResultSet のデータを圧縮するかどうかを指定します。 このプロパティーを "true" に設定すると、 ResultSet のデータは圧縮されます。 このプロパティーを "false" に設定すると、 ResultSet のデータは圧縮されません。 データ圧縮を使うと、大きな ResultSet を取り出すときのパフォーマンスが高まる可能性があります。	いいえ	"true" "false"	"true"

"extended dynamic"	拡張動的サポートを使用するかどうかを指定します。 拡張動的サポートには、サーバーで動的 SQL ステートメントをキャッシュに入れるためのメカニズムが備えられています。» 特定の SQL ステートメントが初めて準備されると、 そのステートメントはサーバーの SQL パッケージに保管されます。 パッケージが存在しない場合は、自動的に作成されます。 その後同じ SQL ステートメントが準備されるたびに、 SQL パッケージに保管されている情報をサーバーが使用するなら、処理の大部分をスキップすることができます。« これを "true" に設定する場合、 "package" プロパティーを使ってパッケージ名を設定しなければなりません。	いいえ	"true" "false"	"false"
"lazy close"	後続の要求が出されるまでカーソルのクローズを遅らせるかどうかを指定します。 この場合、合計要求数を減らせばパフォーマンスが全体的に向上します。	いいえ	"true" "false"	"false"
"lob threshold"	ResultSet の一部として取り出せる LOB (ラージ・オブジェクト) の最大サイズ (バイト単位) を指定します。 このしきい値より大きい LOB の場合、サーバーとの余分な通信が行われ、いくつかの部分に分けて取り出されます。 LOB のしきい値を大きくすると、サーバーとの通信の頻度は減りますが、 使用しない LOB データまでダウンロードすることになります。 LOB のしきい値を小さくすると、サーバーとの通信の頻度は増えますが、 必要な LOB データしかダウンロードされません。	いいえ	"0" - "16777216"	"0"
"package"	SQL パッケージのベース名を指定します。» サーバーの SQL パッケージの名前を生成するのに、最初の 7 文字しか使用されないことに注意してください。« "extended dynamic" プロパティーが "true" に設定されていないと、このプロパティーには効力はありません。 さらに、"extended dynamic" プロパティーを "true" に設定する場合は、 このプロパティーを設定しなければなりません。	いいえ	SQL パッケージ	""

"package add"	» 新規に準備されたステートメントを "package" プロパティーで指定された SQL パッケージに追加するかどうかを指定します。 "extended dynamic" プロパティーが "true" に設定されていないと、 このプロパティーには効力はありません。 «	いいえ	"true" "false"	"true"
"package cache"	» SQL パッケージ情報のサブセットを、クライアント・メモリーにキャッシュするかどうかを指定します。 SQL パッケージをローカルでキャッシュに入れると、サーバーとの準備および記述のための通信量が減ります。 "extended dynamic" プロパティーが "true" に設定されていないと、 このプロパティーには効力はありません。 «	いいえ	"true" "false"	"false"
"package criteria"	SQL パッケージに保管する SQL ステートメント・タイプを指定します。 これは、複雑な結合条件のパフォーマンスを高めるのに役立ちます。 "extended dynamic" プロパティーが "true" に設定されていないと、このプロパティーには効力はありません。	いいえ	"default" (パラメーター・マーカーの付いた SQL ステートメントだけをパッケージに保管します) » "select" (すべての SQL SELECT ステートメントをパッケージに保管します) «	"default"
"package error"	SQL パッケージにエラーが起きたときに取るアクションを指定します。 SQL パッケージにエラーが起きると、 このプロパティーの値に基づいてドライバーは任意に SQLException をスローするか、または接続に対する警告を出します。 "extended dynamic" プロパティーが "true" に設定されていないと、このプロパティーには効力はありません。	いいえ	"exception" "warning" "none"	"warning"
"package library"	SQL パッケージのライブラリーを指定します。 "extended dynamic" プロパティーが "true" に設定されていないと、このプロパティーには効力はありません。	いいえ	SQL パッケージのライブラリー	"QGPL"
"prefetch"	SELECT ステートメントを実行すると同時にデータを事前取り出しするかどうかを指定します。 これを指定すると、ResultSet 内の初期の行にアクセスするときのパフォーマンスが向上します。	いいえ	"true" "false"	"true"

ソートのプロパティー

ソートのプロパティーはサーバーが保管およびソートを実行する方法を指定します。

ソートのプロパティー	説明	必須かどうか	選択項目	デフォルト値
"sort"	クライアントに送る前にサーバーがレコードをソートする方法を指定します。	いいえ	"hex" (16 進数値に基づいたソート) "job" (サーバー・ジョブの設定値に基づいたソート) "language" ("sort language" プロパティーに設定されている言語に基づいたソート) "table" ("sort table" プロパティーに設定されているソート・シーケンス・テーブルに基づいたソート)	"job"
"sort language"	ソート・シーケンスを選択する際に使用する 3 文字の言語 ID を指定します。 "sort" プロパティーが "language" に設定されていないと、このプロパティーには効力はありません。	いいえ	言語 ID	ENU
"sort table"	サーバーに保管されるソート・シーケンス・テーブルのライブラリーおよびファイル名を指定します。 "sort" プロパティーが "table" に設定されていないと、このプロパティーには効力はありません。	いいえ	ソート・テーブルの修飾名	""
"sort weight"	サーバーがレコードをソートするときに大文字小文字を処理する方法を指定します。 "sort" プロパティーが "language" に設定されていないと、このプロパティーには効力はありません。	いいえ	"shared" (大文字と小文字を同一文字としてソートします) "unique" (大文字と小文字を別々の文字としてソートします)	"shared"

その他のプロパティー

その他のプロパティーは、簡単にカテゴリー化できないプロパティーです。 これらのプロパティーは、使用される JDBC ドライバーを判別し、 またデータベースへのアクセス・レベルに関連したオプション、 両方向ストリング・タイプ、データ切り捨てなどを指定します。

その他のプロパティー	説明	必須かどうか	選択項目	デフォルト値
"access"	接続でのデータベースへのアクセス・レベルを指定します。	いいえ	"all" (すべての SQL ステートメントを指定できます) "read call" (SELECT および CALL ステートメントを指定できます) "read only" (SELECT ステートメントのみ)	"all"
»"behavior override"	どの IBM Toolbox for Java JDBC ドライバーが動作オーバーライドするのかを指定します。 定数を加算し、その合計をこのプロパティーに渡すという組み合わせにより、複数の動作を変更することができます。 変更された動作をアプリケーションが正確に取り扱うことができるか確認してください。	いいえ	"" (どの動作もオーバーライドしない) "1" (Statement.executeQuery() または PreparedStatement.executeQuery() が ResultSet を戻さない場合、例外を投じるのではなく、ResultSet としてヌルを戻す)	""«
"bidirectional string type"	両方向データの出力ストリング・タイプを指定します。 詳細については、BidiStringType を参照してください。	いいえ	"" (CCSID を使って両方向ストリング・タイプを指定します) "0" (非両方向データ (LTR) のデフォルトのストリング・タイプ) "4" "5" "6" "7" "8" "9" "10" "11"	""
"data truncation"	» 文字データの切り捨ての際に警告および例外を生成するかどうかを指定します。 このプロパティーが "true" の場合、以下の事柄が適用されます。 ● 切り捨てられたデータがデータベースに書き込まれると、例外が出されます。 ● 切り捨てられたデータを照会に使用すると、警告が通知されます。 このプロパティーが "false" の場合、切り捨てられたデータのデータベースへの書き込み、 またはそのようなデータの照会での使用によっても、例外または警告は生成されません。	いいえ	"true" "false"	"true"

	デフォルト値は "true" です。 このプロパティーは、数値データには効力がありません。 切り捨てられた数値データがデータベースに書き込まれると必ずエラーが生じ、 切り捨てられた数値データを照会に使用すると必ず警告が通知されます。 «			
"driver"	JDBC ドライバーのインプリメンテーションを指定します。 IBM Toolbox for Java の JDBC ドライバーは、 環境に基づいて別の JDBC ドライバー・インプリメンテーションを使用できます。 プログラムの接続先のデータベースと同じサーバー上の iSeries JVM 環境の場合、 ネイティブの IBM Developer Kit for Java の JDBC ドライバーを使用できます。 その他の環境ではすべて、IBM Toolbox for Java の JDBC ドライバーが使われます。 "secondary URL" プロパティーが設定されていると、このプロパティーには効力はありません。	いいえ	"toolbox" (IBM Toolbox for Java の JDBC ドライバーのみを使用) "native" (サーバー上で実行する場合には IBM Developer Kit for Java の JDBC ドライバーを使用し、 その他の場合は Toolbox for Java の JDBC ドライバーを使用)	"toolbox"
"errors"	サーバー上で起こったエラーに関するメッセージをどれだけ詳しいものにするかを指定します。	いいえ	"basic" "full"	"basic"
» "extended metadata"	ドライバーが拡張メタデータをサーバーから要求するかどうかを指定します。 このプロパティーを true に設定すると、 以下の ResultSetMetaData メソッドから戻される情報の正確性が増します。 ● getColumnLabel(int) ● isReadOnly(int) ● isSearchable(int) ● isWriteable(int) さらに、このプロパティーを true に設定することにより、 ResultSetMetaData.getSchemaName(int) メソッドをサポートできます。 このプロパティーを true に設定すると、サーバーからより多くの情報を検索する必要があるため、パフォーマンスが遅くなる可能性があります。 リストされているメソッドからの特定の追加情報を必要とするのでない限り、 プロパティーをデフォルト (false) のままにしてください。 たとえば、このプロパティーがオフ (false) の場合、 ドライバーには判断を下すのに十分なサーバーからの情報がないため、	いいえ	"true" "false"	"false"«

	ResultSetMetaData.isSearchable(int) は常に "true" を戻します。 このプロパティーをオン (true) にすると、 ドライバーがサーバーから適切なデータを取得するよう強制します。 OS/400 V5R2 以降が実行されているサーバーへの接続に関してのみ、拡張メタデータを使用できます。			
"full open"	照会ごとにサーバーがファイルを完全にオープンするかどうかを指定します。 デフォルトでは、サーバーはオープン要求を最適化します。 この最適化によってパフォーマンスは向上しますが、 データベース・モニターが活動中に照会が複数回実行されると、最適化は失敗することがあります。 モニターが活動状態にあるときは、同じ照会を発行する場合だけプロパティーを true に設定します。	いいえ	"true" "false"	"false"
"key ring name"	サーバーとの SSL 接続で使用するキー・リング・クラス名を指定します。 "secure" を true に設定し、 しかも "key ring password" プロパティーを使ってキー・リング・パスワードを設定しない限り、 key ring name プロパティーには効力はありません。	いいえ	"key ring name"	""
"key ring password"	サーバーとの SSL 通信で使用するキー・リング・クラスのパスワードを指定します。 "secure" を true に設定し、 しかも "key ring name" プロパティーを使ってキー・リング名を設定しない限り、 key ring password プロパティーには効力はありません。	いいえ	"key ring password"	""
"proxy server"	Proxy サーバーが稼働する中間層マシンのホスト名とポートを指定します。 その形式は hostname[:port] です。port は任意選択です。 これを設定しないと、ホスト名とポートは *com.ibm.as400.access.AS400.proxyServer* プロパティーから取り出されます。 デフォルト・ポートは *3470* です (ただし SSL を使用する接続の場合ポートは *3471* です)。 Proxy サーバーは中間層マシンで稼働していなければなりません。 2 層環境では中間層マシンの名前は無視されます。	いいえ	Proxy サーバーのホスト名とポート	(proxyServer プロパティーの値。ただし、設定されていなければ、なし。)

"remarks"	DatabaseMetaData メソッドで戻される ResultSets 内の REMARKS 列にあるテキストのソースを指定します。	いいえ	"sql" (SQL オブジェクトの注記) "system" (OS/400 オブジェクト記述)	"system"
»"シリアライズされた場合のパスワードの保存"	このデータ・ソース・オブジェクトをシリアライズする場合に、 残りのプロパティーと共にパスワードをローカルに保管するかどうかを指定します。 パスワードを保管することは、サーバーへの接続に必要なすべての情報がオブジェクトに含まれているために、シリアライズされた形のオブジェクトをアプリケーションが保護しなければならないことを意味します。 このプロパティーを "true" に設定する前に、パスワードを残りのプロパティーと共に保管することによって生じるセキュリティー・リスクを考慮してください。	いいえ	"true" "false"	"false"«
"secondary URL"	複数層環境内の中間層の DriverManager での接続で使用する URL が、 すでに指定されているものと異なる場合に、その URL を指定します。 このドライバーを使って iSeries または AS/400e サーバー以外のデータベースに接続するときにこのプロパティーを使います。 URL 内のバックスラッシュとセミコロンの前のエスケープ文字としてバックスラッシュを使います。	いいえ	JDBC URL	(現在の JDBC URL)
"secure"	サーバーと通信するのに Secure Sockets Layer (SSL) 接続を使用するかどうかを指定します。 SSL 接続を使えるのは、サーバーへの接続が V4R4 以上の場合だけです。	いいえ	"true" (クライアント/サーバー通信をすべて暗号化します) "false" (パスワードだけを暗号化します)	"false"
» "server trace"	JDBC サーバー・ジョブのトレースのレベルを指定します。 トレースが使用可能な場合、クライアントがサーバーに接続する時にトレースを開始し、 接続が切断される時に終了します。 クライアントは接続時のみにサーバーのトレースができるので、サーバーに接続する前にトレースを開始する必要があります。	いいえ	"0" (トレースはアクティブではない) "2" (JDBC サーバー・ジョブのデータベース・モニターを開始する) "4" (JDBC サーバー・ジョブのデバッグを開始する) "8" (JDBC サーバー・ジョブが終了する際にジョブ・ログを保管する) "16" (JDBC サーバー・ジョブのジョブのトレースを開始する) "32" (SQL 情報を保管する) 複数タイプのトレースは、これらの値を加算することによって	"0"«

			開始できます。 たとえば、"6" はデータベース・モニターおよびデバッグを開始します。	
"thread used"	ホスト・サーバーとの通信でスレッドを使用する必要があるかどうかを指定します。	いいえ	"true" "false"	"true"
"trace"	トレース・メッセージを記録するかどうかを指定します。 JDBC を呼び出すプログラムをデバッグする際にトレース・メッセージが役に立ちます。 ただし、トレース・メッセージのログ記録をとることはパフォーマンス上は好ましくないので、 このプロパティーは、デバッグの場合にのみ "true" に設定してください。 トレース・メッセージは System.out に記録されます。	いいえ	"true" "false"	"false"
"translate binary"	バイナリー・データを変換するかどうかを指定します。 このプロパティーを "true" に設定すると、 BINARY フィールドと VARBINARY フィールドは、CHAR フィールドと VARCHAR フィールドとして扱われます。	いいえ	"true" "false"	"false"

AS400JDBCBlob クラス

AS400JDBCBlob オブジェクトを使用して、 サウンド・バイト・ファイル (.wav) やイメージ・ファイル (.gif) などの、 バイナリー・ラージ・オブジェクト (BLOB) にアクセスすることができます。

AS400JDBCBlob クラスと AS400JDBCBlobLocator クラスの間の重要な違いは、 BLOB の保管に関してです。 AS400JDBCBlob クラスを使用すると、 BLOB がデータベースに保管されます。 これにより、データベース・ファイルのサイズが増加します。 AS400JDBCBlobLocator クラスでは、 BLOB の場所を指すロケーター (ポインターと考えることができる) がデータベース・ファイルに保管されます。

AS400JDBCBlob クラスでは、 LOB の限界値プロパティーを使用することができます。 このプロパティーでは、 ResultSet の一部として検索できるラージ・オブジェクト (LOB) の最大サイズ (K バイト) を指定します。 すべての LOB を合計したサイズがこの限界値より大きくなると、 いくつかに分けられ、サーバーへの通信がその分追加されて検索が行われます。 LOB の限界値を大きくすると、 サーバーへの通信の頻度は減りますが、 使用されないにもかかわらずダウンロードされる LOB データが増えます。 LOB の限界値を小さくすると、 サーバーへの通信の頻度は増えますが、 必要な LOB データだけをダウンロードすることになります。 使用可能な追加のプロパティーについては、JDBC プロパティーを参照してください。

AS400JDBCBlob クラスを使用して、以下を行うことができます。

- BLOB 全体を解釈されていないバイト・ストリームとして戻す
- BLOB の内容の一部を戻す
- BLOB の長さを戻す
- » バイナリー・ストリームを作成し BLOB に書き込む «
- » BLOB にバイト配列を書き込む «
- » BLOB にバイト配列の全体または一部を書き込む «
- » BLOB を切り捨てる «

»例

例: AS400JDBCBlob クラスを使用して BLOB から読み取る

```
Blob blob = resultSet.getBlob(1);
long length = blob.length();
byte[] bytes = blob.getBytes(1, (int) length);
```

例: AS400JDBCBlob クラスを使用して BLOB を更新する

```
ResultSet rs = statement.executeQuery ("SELECT BLOB FROM MYTABLE");
rs.absolute(5);
Blob blob = rs.getBlob(1);
```

```
        // Change the bytes in the blob, starting at the seventh byte
        // of the blob
    blob.setBytes (7, new byte[] { (byte) 57, (byte) 58, (byte) 98});
        //Update the blob in the result set, changing the blob starting
        // at the seventh byte of the blob (1-based) and truncating the
        // blob at the end of the updated bytes (the blob now has 9 bytes).
    rs.updateBlob(1, blob);
        // Update the database with the change.  This will change the blob
        // in the database starting at the seventh byte of the blob, and
        // truncating at the end of the updated bytes.
    rs.updateRow();
    rs.close();
```

«

AS400JDBCBlobLocator クラス

AS400JDBCBlobLocator オブジェクトを使用して、バイナリー・ラージ・オブジェクトにアクセスすることができます。

AS400JDBCBlobLocator クラスを使用して、以下を行うことができます。

- BLOB 全体を解釈されていないバイト・ストリームとして戻す
- BLOB の内容の一部を戻す
- BLOB の長さを戻す
- » バイナリー・ストリームを作成し BLOB に書き込む «
- » BLOB にバイト配列を書き込む «
- » BLOB にバイト配列の全体または一部を書き込む «
- » BLOB を切り捨てる «

CallableStatement インターフェース

CallableStatement オブジェクトを使用して、SQL ストアード・プロシージャーを実行できます。 呼び出されるストアード・プロシージャーは、 すでにデータベース内に保管されていなければなりません。 CallableStatement にはストアード・プロシージャーは含まれておらず、 ストアード・プロシージャーを呼び出すだけです。

ストアード・プロシージャーは 1 つまたは複数の ResultSet オブジェクトを戻すことができ、 IN パラメーター、OUT パラメーター、および INOUT パラメーターを使用することができます。 新たに CallableStatement オブジェクトを作成するには、 Connection.prepareCall() を使用します。

CallableStatement オブジェクトを使用することにより、バッチ・サポートを使って、 複数の SQL コマンドを 1 つのグループとしてデータベースへ実行依頼することができます。 操作をグループで処理すると 1 度に 1 つずつ処理するより速いため、バッチ・サポートを使用して、 より良いパフォーマンスを得られるかもしれません。 バッチ・サポートの使用の詳細については、JDBC サポートの拡張を参照してください。

» 列索引を使用するほうが良いパフォーマンスが得られますが、CallableStatement を使用することにより、 名前によってパラメーターおよび列を取得・設定できます。 «

以下の例は、CallableStatement インターフェースを使用する方法を示しています。

```
                    // Connect to the server.
Connection c = DriverManager.getConnection("jdbc:as400://mySystem");

                    // Create the CallableStatement
                    // object. It precompiles the
                    // specified call to a stored
                    // procedure. The question marks
                    // indicate where input parameters
                    // must be set and where output
                    // parameters can be retrieved.
                    // The first two parameters are
                    // input parameters, and the third
                    // parameter is an output parameter.
CallableStatement cs = c.prepareCall("CALL MYLIBRARY.ADD (?, ?, ?)");

                    // Set input parameters.
cs.setInt (1, 123);
cs.setInt (2, 234);
```

```
                    // Register the type of the output
                    // parameter.
cs.registerOutParameter (3, Types.INTEGER);

                    // Run the stored procedure.
cs.execute ();

                    // Get the value of the output
                    // parameter.
int sum = cs.getInt (3);

                    // Close the CallableStatement and
                    // the Connection.
cs.close();
c.close();
```

AS400JDBCClob クラス

AS400JDBCClob オブジェクトを使用して、 大きな文書などのキャラクター・ラージ・オブジェクト (CLOB) にアクセスすることができます。

AS400JDBCClob クラスと AS400JDBCClobLocator クラスの間の重要な違いは、BLOB の保管に関してです。 AS400JDBCClob クラスを使用すると、CLOB がデータベースに保管されます。 これにより、データベース・ファイルのサイズが増加します。 AS400JDBCClobLocator クラスでは、 CLOB の場所を指すロケーター (ポインターと考えることができる) がデータベース・ファイルに保管されます。

AS400JDBCClob クラスでは、 LOB の限界値プロパティーを使用することができます。 このプロパティーでは、 ResultSet の一部として検索できるラージ・オブジェクト (LOB) の最大サイズ (K バイト) を指定します。 すべての LOB を合計したサイズがこの限界値より大きくなると、 いくつかに分けられ、サーバーへの通信がその分追加されて検索が行われます。 LOB の限界値を大きくすると、 サーバーへの通信の頻度は減りますが、 使用されないにもかかわらずダウンロードされる LOB データが増えます。 LOB の限界値を小さくすると、 サーバーへの通信の頻度は増えますが、 必要な LOB データだけをダウンロードすることになります。 使用可能な追加のプロパティーについては、JDBC プロパティーを参照してください。

AS400JDBCClob クラスを使用して、以下を行うことができます。

- CLOB 全体を ASCII 文字のストリームとして戻す
- CLOB の内容を文字ストリームとして戻す
- CLOB の内容の一部を戻す
- CLOB の長さを戻す
- »Unicode 文字ストリームまたは ASCII 文字ストリームを作成して clob に書き込む«
- »CLOB にストリングを書き込む«
- »CLOB を切り捨てる«

»例

例: AS400JDBCClob クラスを使用して CLOB から読み取る

```
Clob clob = rs.getClob(1);
int length = clob.length();
String s = clob.getSubString(1, (int) length);
```

例: AS400JDBCClob クラスを使用して CLOB を更新する

```
ResultSet rs = statement.executeQuery ("SELECT CLOB FROM MYTABLE");
rs.absolute(4);
Clob clob = rs.getClob(1);
     // Change the characters in the clob, starting at the third character
```

```
        // of the clob
    clob.setString (3, "Small");
        // Update the clob in the result set, starting at the third character
        // of the clob and truncating the clob at the end of the update string
        // (the clob now has 7 characters).
    rs.updateClob(1, clob);
        // Update the database with the updated clob.  This will change the
        // clob in the database starting at the third character of the clob,
        // and truncating at the end of the update string.
    rs.updateRow();
    rs.close();
```

«

AS400JDBCClobLocator クラス

AS400JDBCClobLocator オブジェクトを使用して、 キャラクター・ラージ・オブジェクト (CLOB) にアクセスすることができます。

AS400JDBCClobLocator クラスを使用して、以下を行うことができます。

- CLOB 全体を ASCII 文字のストリームとして戻す
- CLOB 全体を文字ストリームとして戻す
- CLOB の内容の一部を戻す
- CLOB の長さを戻す
- »Unicode 文字ストリームまたは ASCII 文字ストリームを作成して CLOB に書き込む«
- »CLOB にストリングを書き込む«
- »CLOB を切り捨てる«

AS400JDBCConnection クラス

AS400JDBCConnection クラスは、特定の DB2 UDB for iSeries データベースへの JDBC 接続を提供します。 DriverManager.getConnection() を使用して、新しい AS400JDBCConnection オブジェクトを作成します。 詳細については、AS400JDBCDriver を参照してください。

接続が作成された時に指定できる、オプションのプロパティーが多数あります。 プロパティーは URL の一部として指定しても、 または java.util.Properties オブジェクト内に指定してもかまいません。 AS400JDBCDriver によってサポートされているプロパティーの完全なリストは、JDBC のプロパティー を参照してください。

注: 接続は最大 9999 個のオープン・ステートメントを含むことができます。

AS400JDBCConnection には、保管点およびステートメント・レベルでの保持能力のサポート、 および自動生成されたキーを戻すための限定されたサポートが含まれます。 これらおよび他の拡張に関する詳細は、Toolbox for Java JDBC サポートの拡張を参照してください。

Kerberos チケットを使用するには、 JDBC URL オブジェクト上にシステム名だけを設定します (パスワードは設定しない)。 ユーザーの識別は、Java Generic Security Services (JGSS) フレームワークによって検索されますので、 JDBC URL でユーザーを改めて指定する必要はありません。 AS400JDBCConnection オブジェクトで 1 度に 1 つの認証方法だけを設定できます。 パスワードを設定すると、すべての Kerberos チケットまたはプロファイル・トークンが消去されます。 詳細については、AS400 クラスおよび J2SDK, v1.4 Security Documentation を参照してください。

AS400JDBCConnection クラスを使用して、以下を行うことができます。

- ステートメントを作成する (Statement、PreparedStatement、 または CallableStatement オブジェクト)
- 特定の ResultSet タイプおよび並行性を持つステートメントを作成する (Statement、PreparedStatement、 または CallableStatement オブジェクト)
- データベースへの変更をコミットおよびロールバックして、 現在掛けられているデータベース・ロックを解放する
- 自動的に解放されるのを待つのではなくサーバー・リソースをすぐにクローズして、 接続をクローズする
- 接続の保持能力を設定し、 保持能力を取得する
- 接続のトランザクション分離を設定し、 トランザクション分離を取得する

- 接続のメタデータを取得する
- 自動コミットをオンまたはオフに設定する
- 接続に対応するホスト・サーバー・ジョブのジョブ識別子を取得する

JDBC 3.0 を使用し、OS/400 V5R2 を実行しているサーバーへ接続している場合、 AS400JDBCConnection を使用して以下のアクションを実行できます。

- 特定の ResultSet 保持能力を持つステートメントを作成する (Statement、PreparedStatement、または CallableStatement オブジェクト)
- 自動生成されたすべてのキーを戻す preparedStatement を作成する (getGeneratedKeys() が Statement オブジェクトに呼び出される場合)
- トランザクションへのより細かい制御を提供する、保管点を使用する
  - 保管点を設定する
  - 保管点をロールバックする
  - 保管点を解放する «

AS400JDBCConnectionPool

AS400JDBCConnectionPool クラスは、JDBC 2.0 Optional Package API の Toolbox サポートの一部として、 Java プログラムによって使用できる、AS400JDBCConnection オブジェクトのプールを表します。

AS400JDBCConnectionPoolDataSource を使用して、 プールで作成された接続にプロパティーを指定できます。 以下の例にあるとおりです。

接続を要求し、プールが使用中になると、接続プール・データ・ソースを変更できません。 接続プール・データ・ソースをリセットするには、 まずプール上で、close() を呼び出します。

AS400JDBCConnection オブジェクト上で、close() を使用して、AS400JDBCConnectionPool に接続を戻します。

注: 接続がプールに戻されないと、 接続プールのサイズは大きくなり続け、接続は再利用されません。

ConnectionPool から継承されるメソッドを使用して、プール上でプロパティーを設定します。 設定可能なプロパティーは、以下のものがあります。

- プールで許可される接続の最大数
- 接続の最大存続時間
- 接続の最大非活動時間

また、Java Naming and Directory Interface (TM) (JNDI) サービス・プロバイダーを使用して、 AS400JDBCConnectionPoolDataSource オブジェクトを登録することもできます。 JNDI サービス・プロバイダーの詳細については、IBM Toolbox for Java 参照リンクを参照してください。

例: 接続プールを使用する

以下の例では、JNDI から接続プール・データ・ソースを取得し、 それを使用して 10 個の接続を持つ接続プールを作成します。

```
        // Obtain an AS400JDBCConnectionPoolDataSource object from JNDI
        // (assumes JNDI environment is set).
    Context context = new InitialContext(environment);
    AS400JDBCConnectionPoolDataSource datasource =

        (AS400JDBCConnectionPoolDataSource)context.lookup("jdbc/myDatabase");

        // Create an AS400JDBCConnectionPool object.
    AS400JDBCConnectionPool pool = new AS400JDBCConnectionPool(datasource);

        // Adds 10 connections to the pool that can be used by the
```

```
     // application (creates the physical database connections based on
     // the data source).
pool.fill(10);

     // Get a handle to a database connection from the pool.
Connection connection = pool.getConnection();

... Perform miscellaneous queries/updates on the database.

     // Close the connection handle to return it to the pool.
connection.close();

... Application works with some more connections from the pool.

// Close the pool to release all resources.
pool.close();
```

DatabaseMetaData インターフェース

DatabaseMetaData オブジェクトを使用して、 カタログ情報に加え、データベース全体に関する情報を取得することができます。

以下の例は、テーブルのリストを戻す方法 (カタログ関数) を示しています。

```
                    // Connect to the server.
Connection c = DriverManager.getConnection("jdbc:as400://mySystem");

                    // Get the database metadata from
                    // the connection.
DatabaseMetaData dbMeta = c.getMetaData();

                    // Get a list of tables matching the
                    // following criteria.
String catalog = "myCatalog";
String schema  = "mySchema";
String table   = "myTable%"; // % indicates search pattern
String types[]  = {"TABLE", "VIEW", "SYSTEM TABLE"};
ResultSet rs = dbMeta.getTables(catalog, schema, table, types);

                    // ... iterate through the ResultSet
                    // to get the values

                    // Close the Connection.
c.close();
```

AS400JDBCDataSource クラス

AS400JDBCDataSource クラスは、iSeries データベース接続用のファクトリーを表します。 AS400JDBCConnectionPoolDataSource クラスは、 AS400JDBCPooledConnection オブジェクト用のファクトリーを表します。

Java Naming and Directory Interface (JNDI) サービス・プロバイダーを使用して、 どの種類のデータ・ソース・オブジェクトでも登録できます。 JNDI サービス・プロバイダーの詳細については、IBM Toolbox for Java 参照リンクを参照してください。

例

以下の例は、AS400JDBCDataSource オブジェクトを作成して使用する方法を示しています。 最後の 2 つの例は、JNDI を使用して AS400JDBCDataSource オブジェクトを登録してから、 JNDI から戻されるオブジェクトを使ってデータベース接続を獲得する方法を示します。 異なる JNDI サービス・プロバイダーを使用する場合であっても、 コードが非常に類似していることに注意してください。

例: AS400JDBCDataSource オブジェクトを作成する

以下の例は、AS400JDBCDataSource オブジェクトを作成し、それをデータベースに接続する方法を示しています。

```
// Create a data source for making the connection.
AS400JDBCDataSource datasource = new AS400JDBCDataSource("myAS400");
datasource.setUser("myUser");
datasource.setPassword("MYPWD");

// Create a database connection to the iSeries.
Connection connection = datasource.getConnection();
```

例: JDBC 接続をキャッシュに入れるために使用できる AS400JDBCConnectionPoolDataSource オブジェクトを作成する

以下の例は、AS400JDBCConnectionPoolDataSource を使用して JDBC 接続をキャッシュに入れる方法を示しています。

```
// Create a data source for making the connection.
AS400JDBCConnectionPoolDataSource dataSource = new
AS400JDBCConnectionPoolDataSource("myAS400");
datasource.setUser("myUser");
datasource.setPassword("MYPWD");

// Get the PooledConnection.
PooledConnection pooledConnection = datasource.getPooledConnection();
```

例: JNDI サービス・プロバイダー・クラスを使用して AS400JDBCDataSource オブジェクトを保管する

以下の例は、JNDI サービス・プロバイダー・クラスを使用して、 サーバー上の IFS ファイル・システムに DataSource オブジェクトを直接保管する方法を示しています。

```
// Create a data source to the iSeries database.
AS400JDBCDataSource dataSource = new AS400JDBCDataSource();
dataSource.setServerName("myAS400");
dataSource.setDatabaseName("myAS400 Database");

// Register the datasource with the Java Naming and Directory Interface (JNDI).
Hashtable env = new Hashtable();
```

```
        env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.fscontext.RefFSContextFactory");
        Context context = new InitialContext(env);
        context.bind("jdbc/customer", dataSource);

        // Return an AS400JDBCDataSource object from JNDI and get a connection.
        AS400JDBCDataSource datasource = (AS400JDBCDataSource) context.lookup("jdbc/customer");
        Connection connection = datasource.getConnection("myUser", "MYPWD");
```

例: Lightweight Directory Access Protocol (LDAP) ディレクトリー・サーバーで、 AS400JDBCDataSource オブジェクトおよび IBM SecureWay ディレクトリー・クラスを使用する

以下の例は、IBM SecureWay ディレクトリー・クラスを使用して、 オブジェクトを Lightweight Directory Access Protocol (LDAP) ディレクトリー・サーバーに保管する方法を示しています。

```
        // Create a data source to the iSeries database.
        AS400JDBCDataSource dataSource = new AS400JDBCDataSource();
        dataSource.setServerName("myAS400");
        dataSource.setDatabaseName("myAS400 Database");

        // Register the datasource with the Java Naming and Directory Interface (JNDI).
        Hashtable env = new Hashtable();
        env.put(Context.INITIAL_CONTEXT_FACTORY, "com.ibm.jndi.LDAPCtxFactory");
        Context context = new InitialContext(env);
        context.bind("cn=myDatasource, cn=myUsers, ou=myLocation,o=myCompany,c=myCountryRegion",
dataSource);

        // Return an AS400JDBCDataSource object from JNDI and get a connection.
        AS400JDBCDataSource datasource = (AS400JDBCDataSource) context.lookup("cn=myDatasource,
             cn=myUsers, ou=myLocation,o=myCompany,c=myCountryRegion");
        Connection connection = datasource.getConnection("myUser", "MYPWD");
```

JDBC ドライバーの登録

JDBC を使用してサーバー・データベース・ファイル内のデータにアクセスする前に、 IBM Toolbox for Java ライセンス・プログラム用の JDBC ドライバーを DriverManager に登録する必要があります。 ドライバーの登録は、Java システム・プロパティーを使用する方法、 または Java プログラムにドライバーを登録させる方法のいずれかで行えます。

- システム・プロパティーを使用しての登録

  仮想マシンには、それぞれにシステム・プロパティーを設定する独自の方法があります。 たとえば、JDK の Java コマンドではシステム・プロパティーの設定に -D オプションを使用します。 システム・プロパティーを使用してドライバーを設定するには、次のように指定します。

  ```
  "-Djdbc.drivers=com.ibm.as400.access.AS400JDBCDriver"
  ```

- »Java プログラムを使用しての登録

  Toolbox for Java の JDBC ドライバーをロードするには、 Java プログラムの最初の JDBC 呼び出しの前に、次の文を追加します。

  ```
  Class.forName("com.ibm.as400.access.AS400JDBCDriver");
  ```

  Toolbox for Java の JDBC ドライバーは、ロードされる際に自らを登録します。 これはドライバーを登録する望ましい方法です。 以下の方法で、Toolbox JDBC ドライバーを明示的に登録することもできます。

  ```
  java.sql.DriverManager.registerDriver (new com.ibm.as400.access.AS400JDBCDriver ());
  ```

  «

IBM Toolbox for Java の JDBC ドライバーは、 サーバーからデータを取得する他の IBM Toolbox for Java クラスと同様に、 入力パラメーターとしての AS400 オブジェクトは必要としません。 ただし、AS400 オブジェクトは、 デフォルトのユーザーおよびパスワード・キャッシュを管理するために内部で使用されます。 初めてサーバーに接続すると、ユーザーはユーザー ID とパスワードを入力するようプロンプト指示されます。 ユーザーは、入力したユーザー ID をデフォルト・ユーザー ID として保管し、 入力したパスワードをパスワード・キャッシュに追加することもできます。 他の IBM Toolbox for Java 機能と同じく、ユーザー ID とパスワードが Java プログラムによって提供される場合は、 デフォルト・ユーザーが設定されることも、パスワードがキャッシュされることもありません。 接続を管理する方法の詳細については、接続の管理を参照してください。

JDBC ドライバーを使用してサーバー上のデータベースに接続する

DriverManager.getConnection() メソッドを使用して、サーバー・データベースに接続できます。 DriverManager.getConnection() は引き数に URL ストリングを取ります。 JDBC ドライバー・マネージャーは、 URL で表されるデータベースに接続可能なドライバーがある場所を特定しようとします。 IBM Toolbox for Java ドライバーを使用する場合、URL 用に次の構文を使用してください。

```
"jdbc:as400://systemName/defaultSchema;listOfProperties"
```

注: URL では systemName または defaultSchema を省略できます。

» Kerberos チケットを使用するには、 JDBC URL オブジェクト上にシステム名だけを設定します (パスワードは設定しない)。 ユーザーの識別は、Java Generic Security Services (JGSS) フレームワークによって検索されますので、 JDBC URL でユーザーを改めて指定する必要はありません。 AS400JDBCConnection オブジェクトで 1 度に 1 つの認証方法だけを設定できます。 パスワードを設定すると、すべての Kerberos チケットまたはプロファイル・トークンが消去されます。 詳細については、AS400 クラスおよび J2SDK, v1.4 Security Documentation を参照してください。 «

例: JDBC ドライバーを使用したサーバー・データベースへの接続

例 1: システム名が指定されていない URL の使用。 この場合、接続しようとするシステムの名前を入力するよう、 ユーザーに対してプロンプトが出されます。

```
"jdbc:as400:"
```

例 2: サーバー・データベースへの接続 (デフォルト・スキーマまたはプロパティーが指定されていない場合)。

```
                    // Connect to system 'mySystem'. No
                    // default schema or properties are
                    // specified.
Connection c  = DriverManager.getConnection("jdbc:as400://mySystem");
```

例 3: サーバー・データベースへの接続 (デフォルト・スキーマが指定されている場合)。

```
                    // Connect to system 'mySys2'. The
                    // default schema 'myschema' is
                    // specified.
Connection c2 = DriverManager.getConnection("jdbc:as400://mySys2/mySchema");
```

例 4: サーバー・データベースへの接続 (java.util.Properties を使ってプロパティーが指定されている場合)。 Java プログラムは、java.util.Properties インターフェースを使用するか、 または URL の一部として特性を指定することによって、JDBC 特性のセットを指定することができます。 サポートされているプロパティーのリストは、JDBC のプロパティーを参照してください。

たとえば、Properties インターフェースを使用してプロパティーを指定するには、 次のようなコーディング例を使います。

```
                    // Create a properties object.
Properties p = new Properties();

                    // Set the properties for the
                    // connection.
p.put("naming", "sql");
p.put("errors", "full");

                    // Connect using the properties
                    // object.
Connection c = DriverManager.getConnection("jdbc:as400://mySystem",p);
```

例 5: サーバー・データベースへの接続 (URL を使ってプロパティーが指定されている場合)。

```
                    // Connect using properties. The
                    // properties are set on the URL
                    // instead of through a properties
                    // object.
Connection c = DriverManager.getConnection(
                     "jdbc:as400://mySystem;naming=sql;errors=full");
```

例 6: サーバー・データベースへの接続 (ユーザー ID とパスワードが指定されている場合)。

```
                    // Connect using properties on the
                    // URL and specifying a user ID and
                    // password
Connection c = DriverManager.getConnection(
                     "jdbc:as400://mySystem;naming=sql;errors=full",
```

```
                    "auser",
                    "apassword");
```

例 7: データベースからの切断。 Connecting オブジェクト上で close() メソッドを使用して、サーバーから切断します。 上記の例で作成された接続をクローズするには、 次のようなステートメントを使います。

```
c.close();
```

AS400JDBCParameterMetaData

AS400JDBCParameterMetaData クラスにより、 プログラムは PreparedStatement および CallableStatement オブジェクトのパラメーターのプロパティーに関する情報を検索できます。

AS400JDBCParameterMetaData は、以下のアクションを実行できるようにするメソッドを提供します。

- パラメーターのクラス名を取得する
- PreparedStatement のパラメーター数を取得する
- パラメーターの SQL タイプを取得する
- パラメーターのデータベース指定のタイプ名を取得する
- パラメーターの精度を取得するまたは位取りを取得する

例: AS400JDBCParameterMetaData を使用する

以下の例は、AS400JDBCParameterMetaData を使用して動的に生成された PreparedStatement オブジェクトからパラメーターを検索する方法を示します。

```
// Get a connection from the driver.
Class.forName("com.ibm.as400.access.AS400JDBCDriver");
Connection connection =
   DriverManager.getConnection
      ("jdbc:as400://myAS400",
       "myUserId",
       "myPassword");
// Create a prepared statement object.
PreparedStatement ps =
   connection.prepareStatement
      ("SELECT STUDENTS FROM STUDENTTABLE WHERE STUDENT_ID= ?");
// Set a student ID into parameter 1.
ps.setInt(1, 123456);
// Retrieve the parameter meta data for the prepared statement.
ParameterMetaData pMetaData = ps.getParameterMetaData();
// Retrieve the number of parameters in the prepared statement.
// Returns 1.
int parameterCount = pMetaData.getParameterCount();
// Find out what the parameter type name of parameter 1 is.
// Returns INTEGER.
```

```
String getParameterTypeName = pMetaData.getParameterTypeName(1);
```
«

PreparedStatement インターフェース

SQL ステートメントを何回も実行する場合には、PreparedStatement オブジェクトを使用できます。 SQL ステートメントはプリコンパイルすることができます。 「準備済み」ステートメントは、プリコンパイル済みの SQL ステートメントです。 この方法は、 Statement オブジェクトを使用して同じステートメントを何回も実行するより効率的です。 Statement オブジェクトの場合、 ステートメントを実行する度に毎回ステートメントをコンパイルすることになるからです。 さらに、PreparedStatement オブジェクト内の SQL ステートメントには、 1 つまたは複数の IN パラメーターを含めることができます。 PreparedStatement オブジェクトを作成するには、 Connection.prepareStatement() を使用します。

PreparedStatement オブジェクトを使用することにより、バッチ・サポートを使って、 複数の SQL コマンドを 1 つのグループとしてデータベースへ実行依頼することができます。 操作をグループで処理すると 1 度に 1 つずつ処理するより速いため、バッチ・サポートを使用して、 より良いパフォーマンスを得られるかもしれません。 バッチ・サポートの使用の詳細については、JDBC サポートの拡張を参照してください。

以下の例は、PreparedStatement インターフェースを使用する方法を示しています。

```
                    // Connect to the server.
    Connection c = DriverManager.getConnection("jdbc:as400://mySystem");

                    // Create the PreparedStatement
                    // object. It precompiles the
                    // specified SQL statement. The
                    // question marks indicate where
                    // parameters must be set before the
                    // statement is run.
    PreparedStatement ps = c.prepareStatement("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES (?,
?)");

                    // Set parameters and run the
                    // statement.
    ps.setString(1, "JOSH");
    ps.setInt(2, 789);
    ps.executeUpdate();

                    // Set parameters and run the
                    // statement again.
    ps.setString(1, "DAVE");
    ps.setInt(2, 456);
    ps.executeUpdate();

                    // Close PreparedStatement and the
                    // Connection.
    ps.close();
    c.close();
```

ResultSet

照会の実行によって生成されたデータ・テーブルにアクセスするには、 ResultSet オブジェクトを使用します。 テーブル行は、順次検索されます。 1 つの行の中では、列値へは任意の順序でアクセスできます。

ResultSet に保管されているデータの検索は、検索するデータのタイプに応じて、 さまざまな get メソッドを使用して行われます。 next() メソッドは、次の行に移動する際に使用されます。

» 列索引を使用するほうが良いパフォーマンスが得られますが、ResultSet を使用することにより、名前によって列を取得および設定できます。 «

カーソル移動

ResultSet は、Java プログラムがアクセスしている ResultSet 内の行を指すのに、 カーソル (内部ポインター) を使用します。

» getRow() メソッドのパフォーマンスが向上しました。 V5R2 までは、現在行番号で作成された負の値を伴う ResultSet.last()、ResultSet.afterLast()、 および ResultSet.absolute() は利用不能でした。 これまでの制限は解除され、getRow() メソッドは完全に機能するようになります。 «

JDBC 2.0 以降の JDBC 仕様には、データベース内の特定の位置へアクセスするための追加メソッドがあります。

スクロール可能カーソル位置	
absolute afterLast beforeFirst first getRow isAfterLast isBeforeFirst	isFirst isLast last moveToCurrentRow moveToInsertRow previous relative

スクロール機能

ステートメントを実行して ResultSet が作成されたら、 テーブル内の行を逆方向 (最後から最初へ) または順方向 (最初から最後へ) で移動 (スクロール) することができます。

このような移動をサポートしている ResultSet を、 スクロール可能 ResultSet と呼びます。 スクロール可能 ResultSet は、 相対位置決めおよび絶対位置決めもサポートしています。 相対位置決めでは、 現在行を基準として相対的な位置を指定し、 ResultSet 内の行に移動することができます。 絶対位置決めでは、 ResultSet 内での位置を指定し、 目的の行へ直接移動することができます。

JDBC 2.0 以降の JDBC 仕様を使用すれば、 ResultSet クラスで作業する際に 2 つの追加スクロール機能 (スクロール非認識 ResultSet とスクロール認識 ResultSet) を使用することができます。

スクロール非認識 ResultSet は、オープン中に行われた変更を通常認識しません。 それに対し、スクロール認識 ResultSet は変更を認識します。 » IBM Toolbox for Java JDBC ドライバーは、スクロール非認識 ResultSet をサポートしていません。 «

更新可能 ResultSet

アプリケーションで ResultSet を使用する際に、 同時読み取り専用か同時更新可能のいずれかを使用することができます。 同時読み取り専用ではデータを更新できないのに対し、 同時更新可能ではデータへの更新を行うことができ、 さらにデータベースへの書き込みをロックして、 別のトランザクションによる同じデータ項目へのアクセスを制御することもできます。 更新可能 ResultSet では、 行の更新、挿入、および削除を行うことができます。 プログラムで使用できるたくさんの更新メソッドがありますが、以下はその一例です。

- ASCII ストリームの更新
- Big Decimal の更新
- 2 進ストリームの更新

ResultSet インターフェースで使用できる更新メソッドの詳細なリストについては、メソッドの索引を参照してください。

例: 更新可能 ResultSet

以下の例は、 データを更新でき (同時更新)、オープン中に ResultSet へ変更を行うことのできる (スクロール認識) ResultSet を使用する方法を示しています。

```
                    // Connect to the server.
Connection c = DriverManager.getConnection("jdbc:as400://mySystem");

                    // Create a Statement object.  Set the result set
            // concurrency to updatable.
Statement s = c.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                ResultSet.CONCUR_UPDATABLE);

                    // Run a query. The result is placed
                    // in a ResultSet object.
ResultSet rs = s.executeQuery ("SELECT NAME,ID FROM MYLIBRARY.MYTABLE FOR UPDATE");

                    // Iterate through the rows of the ResultSet.
                // As we read the row, we will update it with
                    // a new ID.
int newId = 0;
while (rs.next ())
{

                    // Get the values from the ResultSet.
                    // The first value is a string, and
                    // the second value is an integer.
    String name = rs.getString("NAME");
    int id = rs.getInt("ID");

    System.out.println("Name = " + name);
    System.out.println("Old id = " + id);

          // Update the id with a new integer.
    rs.updateInt("ID", ++newId);
```

```
                    // Send the updates to the server.
    rs.updateRow ();

    System.out.println("New id = " + newId);
}

                    // Close the Statement and the
                    // Connection.
s.close();
c.close();
```

ResultSetMetaData

ResultSetMetaData インターフェースは、ResultSet の列のタイプとプロパティーを判別します。

»OS/400 V5R2 以降が実行されているサーバーへの接続の際、 拡張されたメタデータ・プロパティーを使用することにより、 以下の ResultSetMetaData メソッドの正確性が増します。

- getColumnLabel(int)
- isReadOnly(int)
- isSearchable(int)
- isWriteable(int)

さらに、このプロパティーを true に設定することにより、 ResultSetMetaData.getSchemaName(int) メソッドをサポートできます。 拡張されたメタデータ・プロパティーを使用すると、 サーバーからより多くの情報を検索する必要があるため、パフォーマンスが遅くなる可能性があります。«

AS400JDBCRowSet

AS400JDBCRowSet クラスは、 JDBC ResultSet をカプセル化する、接続済みの行セットを表します。 AS400JDBCRowSet のメソッドは、 AS400JDBCResultSet のメソッドに非常に類似しています。 データベース接続は使用中に保守されます。

AS400JDBCDataSource オブジェクトまたは AS400JDBCConnectionPoolDataSource オブジェクトを使用して、AS400JDBCRowSet 用のデータにアクセスするために使用するデータベースへの接続を作成できます。

例

以下の例は、AS400JDBCRowSet クラスの使用方法を示しています。
例: AS400JDBCRowSet オブジェクトを作成、データ入力、および更新する

```
        DriverManager.registerDriver(new AS400JDBCDriver());
        // Establish connection by using a URL.
        AS400JDBCRowSet rowset = new AS400JDBCRowSet("jdbc:as400://mySystem","myUser", "myPassword");

        // Set the command used to populate the list.
        rowset.setCommand("SELECT * FROM MYLIB.DATABASE");

        // Populate the rowset.
        rowset.execute();

        // Update the customer balances.
        while (rowset.next())
        {
           double newBalance = rowset.getDouble("BALANCE") +
july_statements.getPurchases(rowset.getString("CUSTNUM"));
           rowset.updateDouble("BALANCE", newBalance);
           rowset.updateRow();
        }
```

例: JNDI からデータ・ソースを取得する際に AS400JDBCRowSet オブジェクトを作成し、データを移植する

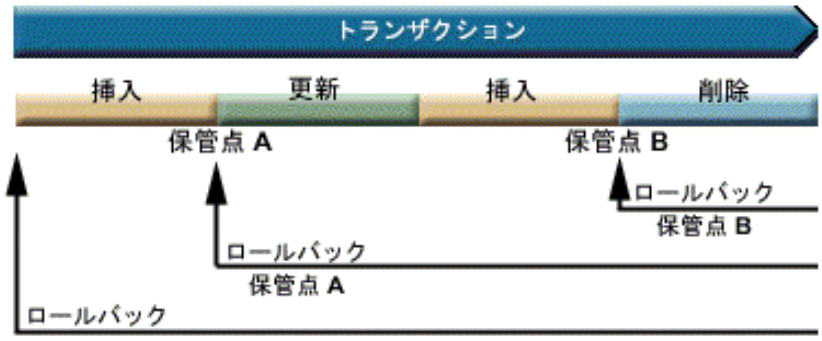
```
        // Get the data source that is registered in JNDI (assumes JNDI environment is set).
        Context context = new InitialContext();
        AS400JDBCDataSource dataSource = (AS400JDBCDataSource) context.lookup("jdbc/customer");

        AS400JDBCRowSet rowset = new AS400JDBCRowSet();
        // Establish connection by setting the data source name.
        rowset.setDataSourceName("jdbc/customer");
        rowset.setUsername("myuser");
        rowset.setPassword("myPasswd");

        // Set the prepared statement and initialize the parameters.
        rowset.setCommand("SELECT * FROM MYLIBRARY.MYTABLE WHERE STATE = ? AND BALANCE > ?");
        rowset.setString(1, "MINNESOTA");
        rowset.setDouble(2, MAXIMUM_LIMIT);

        // Populate the rowset.
        rowset.execute();
```

»

AS400JDBCSavepoint クラス

AS400JDBCSavepoint クラスは、トランザクションの論理上の切れ目を表します。 保管点を使用すると、トランザクションをロールバックする際に及ぶ変更を、一層細かく制御できます。

図 1: トランザクションのロールバックを制御するために保管点を使用する



たとえば、図 1 は 2 つの保管点 A および B を含むトランザクションを示します。 トランザクションをどちらかの保管点にロールバックすると、 ロールバックが呼び出されたポイントから保管点までの変更だけを取り消す (元に戻す) ことになります。 これにより、トランザクション全体の変更すべてを取り消すということを避けられます。 一度保管点 A にロールバックすると、 後から保管点 B にロールバックすることはできないことに注意してください。 保管点 B を越えてロールバックした後には、そこにはアクセスできません。

例: 保管点を使用する

このシナリオでは、アプリケーションが生徒のレコードを更新するものとします。 各生徒レコードの特定のフィールドを更新し終える際に、コミットを実行します。 コードはこのフィールドの更新に関連した特定のエラーを検出し、 このエラーの発生時に行われた作業をロールバックします。 この特定のエラーは、現行レコードで実行された作業のみに影響することが分かっています。

それで、各生徒レコードの更新と更新の間に保管点を設定します。 このエラーが発生した時点で、生徒テーブルの最後の更新のみをロールバックします。 これで、膨大な作業をロールバックするのではなく、少量の作業のみをロールバックできます。

以下のコード例は、この保管点の使用法を理解するのに役立ちます。 この例では、John の生徒 ID (student ID) は 123456 で、Jane の生徒 ID は 987654 です。

```
// Get a connection from the driver
Class.forName("com.ibm.as400.access.AS400JDBCDriver");
// Get a statement object
Statement statement = connection.createStatement();
// Update John's record with his 'B' grade in gym.
int rows = statement.executeUpdate("UPDATE STUDENTTABLE SET GRADE_SECOND_PERIOD = 'B'
       WHERE STUDENT_ID= '123456'");
// Set a savepoint marking an intermediate point in the transaction
Savepoint savepoint1 = connection.setSavepoint("SAVEPOINT_1");
// Update Jane's record with her 'C' grade in biochemistry.
int rows = statement.executeUpdate("UPDATE STUDENTTABLE SET GRADE_SECOND_PERIOD = 'C'
       WHERE STUDENT_ID= '987654'");
```

```
// An error is detected, so we need to roll back Jane's record, but not John's.
// Rollback the transaction to savepoint 1. The change to Jane's record is
// removed while the change to John's record remains.
connection.rollback(savepoint1);
// Commit the transaction; only John's 'B' grade is committed to the database.
connection.commit();
```

考慮事項および制約事項

保管点を使用するには、以下の考慮事項および制約事項を理解している必要があります。

考慮事項

Toolbox for Java は、カーソルおよび保存されたロックにロールバックがどのような影響を及ぼすかに関して、 データベースの規則に従います。 たとえば、従来のロールバックの後にカーソルをオープンの状態に保持する接続オプションを設定すると、 保管点にロールバックした後もカーソルはオープンした状態に維持されます。 言い換えると、ロールバックの要求が保管点に関連して発生しても、 基礎となるデータベースが保管点をサポートしていなければ、 Toolbox for Java はカーソルを移動またはクローズしません。

保管点を使用してトランザクションをロールバックすると、 ロールバックを開始したポイントから保管点までに実行されたアクションのみを取り消します。 その保管点より前に実行されたアクションは残ります。 前の例にあったように、特定の保管点より前に実行された作業は含まれますが、 保管点より後に実行された作業は含まないトランザクションをコミットできるという点に注意してください。

トランザクションがコミットされる、またはトランザクション全体がロールバックされると、 すべての保管点は解放されて無効になります。 Connection.releaseSavepoint()を呼び出すことによっても、保管点を解放できます。

制約事項

以下の制約事項は保管点を使用する際に適用されます。

- 保管点の名前は固有でなければなりません。
- 保管点が解放される、コミットされる、またはロールバックされるまで、 保管点名は再使用できません。
- 保管点を有効にするには、自動コミットを「オフ」に設定する必要があります。 Connection.setAutoCommit(false) を使用して自動コミットを「オフ」に設定できます。 保管点の使用時に自動コミットを使用可能にすると、例外が出されます。
- 保管点は、XA 接続では無効です。 XA 接続で保管点を使用すると、例外が出されます。
- サーバーは OS/400 バージョン 5 リリース 2 以降を実行していることが必要です。 OS/400 の V5R1 以前のバージョンを実行しているサーバーに接続 (またはすでに接続済み) の場合に保管点を使用すると、 例外が出されます。«

Statement オブジェクトを使用した SQL ステートメントの実行

SQL ステートメントを実行して、SQL ステートメントによって作成された ResultSet を任意で取得する際には、 Statement オブジェクトを使用します。

PreparedStatement は Statement から継承し、 CallableStatement は PreparedStatement から継承します。 異なる複数の SQL ステートメントを実行するには、 以下の Statement オブジェクトを使用します。

- Statement - パラメーターを取らない単純な SQL ステートメントを実行する場合。
- PreparedStatement - IN パラメーターを取り得る (取らない場合もある) プリコンパイル済み SQL ステートメントを実行する場合。
- CallableStatement - データベース・ストアード・プロシージャーへの呼び出しを実行する場合。 CallableStatement は、IN、OUT、および INOUT パラメーターを取ることができます (取らなくても構いません)。

Statement オブジェクトを使用することにより、バッチ・サポートを使って、 複数の SQL コマンドを 1 つのグループとしてデータベースへ実行依頼することができます。 操作をグループで処理すると 1 度に 1 つずつ処理するより速いため、バッチ・サポートを使用して、 より良いパフォーマンスを得られるかもしれません。 バッチ・サポートの使用の詳細については、JDBC サポートの拡張を参照してください。

バッチ更新を使用する場合、 通常、自動コミットはオフにしなければなりません。 自動コミットをオフにすることにより、 エラーが発生してコマンドのいくつかが実行できない場合、 プログラムがトランザクションをコミットすべきかどうかを決定することができます。 JDBC 2.0 以降の JDBC 仕様では、グループとして正常に実行依頼し、実行できるコマンドのリストを、 Statement オブジェクトが常に保持することができます。 executeBatch() メソッドでバッチ・コマンドのこのリストを実行する場合、 リストに追加された順番でコマンドが実行されます。

AS400JDBCStatement は、 以下のアクションを含む多くのアクションを実行できるようにするメソッドを提供します。

- さまざまな種類のステートメントの実行
- 以下のものを含む、Statement オブジェクトのさまざまなパラメーターの値の検索
  - 接続
  - Statement の実行の結果として作成される、すべての自動生成されたキー
  - フェッチ・サイズおよびフェッチ方向
  - 最大フィールド・サイズおよび最大行制限
  - 現行 ResultSet、 次の ResultSet、 ResultSet のタイプ、 ResultSet の並行性、 および ResultSet のカーソル保持能力
- 現行バッチへの SQL ステートメントの追加
- SQL ステートメントの現行バッチの実行

Statement インターフェース

新規の Statement オブジェクトの作成には、 Connection.createStatement() を使用します。

以下の例は、Statement オブジェクトを使用する方法を示しています。

```
                    // Connect to the server.
Connection c = DriverManager.getConnection("jdbc:as400://mySystem");

                    // Create a Statement object.
Statement s = c.createStatement();

                    // Run an SQL statement that creates
                    // a table in the database.
s.executeUpdate("CREATE TABLE MYLIBRARY.MYTABLE (NAME VARCHAR(20), ID INTEGER)");

                    // Run an SQL statement that inserts
                    // a record into the table.
s.executeUpdate("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES ('DAVE', 123)");

                    // Run an SQL statement that inserts
                    // a record into the table.
s.executeUpdate("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES ('CINDY', 456)");

                    // Run an SQL query on the table.
ResultSet rs = s.executeQuery("SELECT * FROM MYLIBRARY.MYTABLE");

                    // Close the Statement and the
                    // Connection.
s.close();
c.close();
```

JDBC XA 分散トランザクション管理

JDBC XA 分散トランザクション管理クラスを使用すると、 分散トランザクション内で IBM Toolbox for Java JDBC ドライバーを使用できます。 XA クラスを使用して IBM Toolbox for Java JDBC ドライバーを使用可能にすると、 そのクラスは複数のデータ・ソースにまたがるトランザクションに参加できます。

通常、XA 分散トランザクション管理クラスは、 JDBC ドライバーとは別個のトランザクション・マネージャーによって直接使用および制御されます。 分散トランザクション管理インターフェースは、 JDBC 2.0 Optional Package および Java Transaction API (JTA) の一部として定義されます。 どちらも、jar ファイルとして Sun から使用できます。 » また、分散トランザクション管理インターフェースが、 Java 2 Platform、Standard Edition、バージョン 1.4 にバンドルされている、 JDBC 3.0 API でサポートされます。 «

詳細については、Sun Web サイトの JDBC、 および JTA を参照してください。

次のオブジェクトを使用すると、IBM Toolbox for Java JDBC ドライバーを XA 分散トランザクションに参加させることができます。

- AS400JDBCXADataSource - AS400JDBCXAConnection オブジェクト用のファクトリー。 これは、 AS400JDBCDataSource のサブクラスです。
- AS400JDBCXACConnection - 接続プール管理および XA リソース管理用のフックを提供する、 プールに入れられた接続オブジェクト。
- AS400JDBCXAResource - XA トランザクション管理での使用のためのリソース・マネージャー。

例: XA クラスを使用する

以下の例は、XA クラスの基本的な使用法を示しています。 詳細部分は他のデータ・ソースを使用する作業で補われるということを念頭に置いてください。 通常、このタイプのコードはトランザクション・マネージャー内に表示されます。

```
// Create an XA data source for making the XA connection.
AS400JDBCXADataSource xaDataSource = new AS400JDBCXADataSource("myAS400");
xaDataSource.setUser("myUser");
xaDataSource.setPassword("myPasswd");

// Get an XAConnection and get the associated XAResource.
// This provides access to the resource manager.
XAConnection xaConnection = xaDataSource.getXAConnection();
XAResource xaResource = xaConnection.getXAResource();

// Generate a new Xid (this is up to the transaction manager).
Xid xid = ...;

// Start the transaction.
```

```
xaResource.start(xid, XAResource.TMNOFLAGS);

// ...Do some work with the database...

// End the transaction.
xaResource.end(xid, XAResource.TMSUCCESS);

// Prepare for a commit.
xaResource.prepare(xid);

// Commit the transaction.
xaResource.commit(xid, false);

// Close the XA connection when done.  This implicitly
// closes the XA resource.
xaConnection.close();
```

jobs クラス

IBM Toolbox for Java の Jobs クラス (アクセス・パッケージ内にある) を使用すると、 Java プログラムがジョブ情報を検索したり、変更することができます。

注: Toolbox for Java は、 さまざまな iSeries オブジェクトおよびリストを扱うための汎用フレームワークおよび一貫性のあるプログラミング・インターフェースを提供するリソース・クラスも備えています。 アクセス・パッケージおよびリソース・パッケージのクラスに関する記述を読んでから、 ご使用のアプリケーションにとって最善のオブジェクトを選択できます。 ジョブを扱うリソース・クラスは、RJob、 RJobList、 および RJobLog です。

Jobs クラスを使用して、以下のタイプのジョブ情報を処理することができます。

- 日時情報
- ジョブ待ち行列
- 言語識別コード
- メッセージ・ログ
- 出力待ち行列
- プリンター情報

access パッケージ内にある job クラスは次のとおりです。

- Job - iSeries ジョブ情報の検索および変更を行います。
- JobList - 検索を行い、iSeries ジョブをリストします
- JobLog - iSeries のジョブ・ログを表します

例

特定のユーザーに帰属するジョブをリストし、ジョブ状況情報を使ってジョブをリストします。

ジョブ・ログ内のメッセージを表示します。

値の設定および取得を行う際に、 以下のようにキャッシュを使用します。

```
  try {
      // Creates AS400 object.
      AS400 as400 = new AS400("systemName");
      // Constructs a Job object
      Job job = new Job(as400,"QDEV002");
      // Gets job information
      System.out.println("User of this job :" + job.getUser());
      System.out.println("CPU used :" + job.getCPUUsed();
      System.out.println("Job enter system date : " + job.getJobEnterSystemDate());
      // Sets cache mode
      job.setCacheChanges(true);
      // Changes will be store in the cache.
      job.setRunPriority(66);
      job.setDateFormat("*YMD");
```

```
    // Commit changes. This will change the value on the iSeries.
    job.commitChanges();
    // Set job information to system directly(without cache).
    job.setCacheChanges(false);
    job.setRunPriority(60);
} catch (Exception e)
{
    System.out.println(quot;error :" + e)
}
```

Job

job クラス (アクセス・パッケージ内にある) を使用すると、 Java プログラムでサーバー・ジョブ情報を検索したり、変更することができます。

注: Toolbox for Java は、 さまざまな iSeries オブジェクトおよびリストを扱うための汎用フレームワークおよび一貫性のあるプログラミング・インターフェースを提供するリソース・クラスも備えています。 アクセス・パッケージおよびリソース・パッケージのクラスに関する記述を読んでから、 ご使用のアプリケーションにとって最善のオブジェクトを選択できます。 ジョブを扱うリソース・クラスは、 RJob、 RJobList、 および RJobLog です。

job クラスでは、以下のタイプのジョブ情報の検索および変更を行うことができます。

- ジョブ待ち行列
- 出力待ち行列
- メッセージ・ログ
- 印刷装置
- 国および地域別 ID
- 日付形式

また、job クラスでは、一度に 1 つずつ値を変更したり、 setCacheChanges(true) メソッドを使用して多くの変更をキャッシュできます。また、 commitChanges() メソッドを使用して、変更をコミットできます。キャッシュがオンになっていない場合、 コミットを行う必要はありません。

setRunPriority() メソッドを使用した実行優先順位の設定、 および setDateFormat() メソッドを使用した日付形式の設定を行うための、 キャッシュへの値の出し入れの方法については、 この例を使用してください。

JobList クラス

JobList クラス (アクセス・パッケージ内にある) を使用して、 iSeries ジョブをリストできます。

注: Toolbox for Java は、 さまざまな iSeries オブジェクトおよびリストを扱うための汎用フレームワークおよび一貫性のあるプログラミング・インターフェースを提供するリソース・クラスも備えています。 アクセス・パッケージおよびリソース・パッケージのクラスに関する記述を読んでから、 ご使用のアプリケーションにとって最善のオブジェクトを選択できます。 ジョブを扱うリソース・クラスは、 RJob、 RJobList、 および RJobLog です。

JobList クラスを使うと、以下のものを検索できます。

- すべてのジョブ
- 名前、ジョブ番号、またはユーザーごとのジョブ

getJobs() メソッドを使用すれば、 iSeries ジョブのリストを戻すことができ、 getLength() メソッドを使用すれば、前回の getJobs() で検索されたジョブの数を戻すことができます。

以下の例では、システム上のすべてのアクティブなジョブをリストします。

```
                    // Create an AS400 object. List the
                    // jobs on this iSeries.
AS400 sys = new AS400("mySystem.myCompany.com");

                    // Create the job list object.
JobList jobList = new JobList(sys);

                    // Get the list of active jobs.
Enumeration list = jobList.getJobs();

                    // For each active job on the system
                    // print job information.
while (list.hasMoreElements())
{
    Job j = (Job) list.nextElement();

    System.out.println(j.getName() + "." +
                       j.getUser() + "." +
                       j.getNumber());
}
```

JobLog

JobLog クラス (アクセス・パッケージ内にある) は、 getMessages() を呼び出して、サーバー・ジョブのジョブ・ログ内のメッセージを検索します。

注: Toolbox for Java は、 さまざまな iSeries オブジェクトおよびリストを扱うための汎用フレームワークおよび一貫性のあるプログラミング・インターフェースを提供するリソース・クラスも備えています。 アクセス・パッケージおよびリソース・パッケージのクラスに関する記述を読んでから、 ご使用のアプリケーションにとって最善のオブジェクトを選択できます。 ジョブを扱うリソース・クラスは、 RJob、 RJobList、 および RJobLog です。

以下の例は、 指定されたユーザー別にジョブ・ログ内のすべてのメッセージを印刷します。

```
                    // ... Setup work to create an AS400
                    // object and a jobList object has
                    // already been done

                    // Get the list of active jobs on
                    // the iSeries
Enumeration list = jobList.getJobs();

                    // Look through the list to find a
                    // job for the specified user.
while (list.hasMoreElements())
{
   Job j = (Job) list.nextElement();

   if (j.getUser().trim().equalsIgnoreCase(userID))
   {
                    // A job matching the current user
                    // was found. Create a job log
                    // object for this job.
      JobLog jlog = new JobLog(system,
                               j.getName(),
                               j.getUser(),
                               j.getNumber());

                    // Enumerate the messages in the job
                    // log then print them.
      Enumeration messageList = jlog.getMessages();

      while (messageList.hasMoreElements())
      {
          AS400Message message = (AS400Message) messageList.nextElement();
```

```
            System.out.println(message.getText());
        }

    }
}
```

Message クラス

AS400Message

AS400Message オブジェクトを使用すると、 Java プログラムは、以前の操作から (たとえば、コマンド呼び出しから) 生成された iSeries メッセージを取り出すことができます。Java プログラムでは、メッセージ・オブジェクトから次のものを取り出すことができます。

- メッセージを含む iSeries ライブラリーおよびメッセージ・ファイル
- メッセージ ID
- メッセージ・タイプ
- メッセージ重大度
- メッセージ・テキスト
- メッセージ・ヘルプ・テキスト

以下の例では、AS400Message オブジェクトを使用する方法を示します。

```
                    // Create a command call object.
CommandCall cmd = new CommandCall(sys, "myCommand");

                    // Run the command
cmd.run();

                    // Get the list of messages that are
                    // the result of the command that I
                    // just ran
AS400Message[] messageList = cmd.getMessageList();

                   // Iterate through the list
                   // displaying the messages
for (int i = 0; i < messageList.length; i++)
{
   System.out.println(messageList[i].getText());
}
```

例

以下の例は、CommandCall および ProgramCall によってメッセージ・リストを使用する方法を示します。

- 例: CommandCall によってメッセージ・リストを使用する
- 例: ProgramCall によってメッセージ・リストを使用する

QueuedMessage

QueuedMessage クラスは、AS400Message クラスを拡張します。

注: Toolbox for Java は、 さまざまな iSeries オブジェクトおよびリストを扱うための汎用フレームワークおよび一貫性のあるプログラミング・インターフェースを提供するリソース・クラスも備えています。 アクセス・パッケージおよびリソース・パッケージのクラスに関する記述を読んでから、 ご使用のアプリケーションにとって最善のオブジェクトを選択できます。 待ち行列に入れられたメッセージを扱うリソース・クラスは、 RQueuedMessage です。

QueuedMessage クラスは、 iSeries メッセージ待ち行列上のメッセージに関する情報にアクセスします。 このクラスを使用すると、Java プログラムでは次のものを取り出すことができます。

- メッセージの発信元、たとえば、プログラム、ジョブ名、ジョブ番号、およびユーザーなどについての情報
- メッセージ待ち行列
- メッセージ・キー
- メッセージ応答状況

以下の例では、現行 (サインオン) ユーザーのメッセージ待ち行列内のすべてのメッセージを印刷します。

```
                    // The message queue is on this iSeries.
AS400 sys = new AS400(mySystem.myCompany.com);

                    // Create the message queue object.
                    // This object will represent the
                    // queue for the current user.
MessageQueue queue = new MessageQueue(sys, MessageQueue.CURRENT);

                    // Get the list of messages currently
                    // in this user's queue.
Enumeration e = queue.getMessages();

                    // Print each message in the queue.
while (e.hasMoreElements())
{
```

```
        QueuedMessage msg = e.getNextElement();
        System.out.println(msg.getText());
    }
```

MessageFile

MessageFile クラスを使用すれば、 iSeries メッセージ・ファイルからメッセージを受け取ることができます。 MessageFile クラスはメッセージを含んでいる AS400Message オブジェクトを戻します。 MessageFile クラスを使用して、以下を行うことができます。

- メッセージを含むメッセージ・オブジェクトを戻す。
- メッセージ内に 置換テキストを含むメッセージ・オブジェクトを戻す。

以下の例では、メッセージを検索して印刷する方法を示します。

```
AS400 system = new AS400("mysystem.mycompany.com");
MessageFile messageFile = new MessageFile(system);
messageFile.setPath("/QSYS.LIB/QCPFMSG.MSGF");
AS400Message message = messageFile.getMessage("CPD0170");
System.out.println(message.getText());
```

MessageQueue

MessageQueue クラスを使用すると、 Java プログラムは iSeries メッセージ待ち行列と対話することができます。

注: Toolbox for Java は、 さまざまな iSeries オブジェクトおよびリストを扱うための汎用フレームワークおよび一貫性のあるプログラミング・インターフェースを提供するリソース・クラスも備えています。 アクセス・パッケージおよびリソース・パッケージのクラスに関する記述を読んでから、 ご使用のアプリケーションにとって最善のオブジェクトを選択できます。 メッセージ待ち行列を扱うリソース・クラスは、 RMessageQueue です。

MessageQueue クラスは、QueuedMessage クラスのコンテナーの役割を果たします。 getMessages() メソッドでは、 QueuedMessage オブジェクトのリストが戻されます。 MessageQueue クラスでは、次のことを行うことができます。

- メッセージ待ち行列属性の設定
- メッセージ待ち行列の取得
- メッセージ待ち行列からのメッセージの受信
- メッセージ待ち行列へのメッセージの送信
- メッセージへの応答

以下の例では、現行ユーザーのメッセージ待ち行列内のメッセージをリストします。

```
                    // The message queue is on this iSeries.
AS400 sys = new AS400(mySystem.myCompany.com);

                    // Create the message queue object.
                    // This object will represent the
                    // queue for the current user.
MessageQueue queue = new MessageQueue(sys, MessageQueue.CURRENT);

                    // Get the list of messages currently
                    // in this user's queue.
Enumeration e = queue.getMessages();

                    // Print each message in the queue.
while (e.hasMoreElements())
{
   QueuedMessage msg = e.getNextElement();
   System.out.println(msg.getText());
}
```

NetServer

NetServer クラスは、iSeries サーバー上の NetServer サービスを表します。 NetServer オブジェクトを使用すると、NetServer サービスの状態と構成を照会し、変更できます。

たとえば、NetServer クラスを使って以下のようなことができます。

- NetServer の開始または停止
- すべての現行のファイルの共用および印刷の共用のリストの入手
- すべての現行セッションのリストの入手
- 照会および属性値の変更 (ChangeableResource から継承したメソッドを使用)

注: NetServer クラスを使用するには、 *IOSYSCFG 権限を持つサーバー・ユーザー・プロファイルが必要です。

NetServer クラスは、ChangeableResource および Resource の拡張であるため、 さまざまな NetServer 値および属性を表す「属性」のコレクションを提供しています。 ご使用の NetServer の構成にアクセスするか、または変更するために、 属性を照会 または 変更 することができます。 NetServer 属性の一部は、以下のとおりです。

- NAME
- NAME_PENDING
- DOMAIN
- ALLOW_SYSTEM_NAME
- AUTOSTART
- CCSID
- WINS_PRIMARY_ADDRESS

保留中の属性

NetServer 属性の多くは保留中です (たとえば、 NAME_PENDING)。 保留中の属性は、次回サーバー上で NetServer を開始 (または再始動) する際に有効な NetServer 値を表します。

関連属性の対があり、一方が保留中で、他方が保留中でない場合、次のことがあてはまります。

- 保留中の属性は読み取り/書き込み可能なので、変更することができます。
- 保留中でない属性は読み取り専用なので、照会はできますが変更できません。

その他の NetServer クラス

関連する NetServer クラスを使用すると、特定の接続、セッション、 ファイル共用、および印刷共用についての詳細情報を取得し、設定できます。

- NetServerConnection - NetServer の接続を表します。

- NetServerFileShare - NetServer のファイル・サーバー共用を表します。
- NetServerPrintShare - NetServer のプリント・サーバー共用を表します。
- NetServerSession - NetServer セッションを表します。
- NetServerShare - NetServer 共用を表します。

例: NetServer オブジェクトを使用して NetServer の名前を変更する

```
// Create a system object to represent the iSeries server.
AS400 system = new AS400("MYSYSTEM", "MYUSERID", "MYPASSWD");
// Create an object with which to query and modify the NetServer.
NetServer nServer = new NetServer(system);
// Set the "pending name" to NEWNAME.
nServer.setAttributeValue(NetServer.NAME_PENDING, "NEWNAME");
// Commit the changes.  This sends the changes to the server.
nServer.commitAttributeChanges();
// The NetServer name will get set to NEWNAME the next time the NetServer
// is ended and started.
```

Permission クラス

Permission を使うと、オブジェクト権限情報を取得して設定することができます。 オブジェクト権限情報は、許可ともいいます。 Permission クラスは、特定のオブジェクトに対して多数のユーザーが持つ権限の集合を表します。 UserPermission クラスは、特定のオブジェクトに対して単一のユーザーが持つ権限を表します。

Permission クラス

Permission クラスを使うと、 オブジェクト権限情報を取り出して変更することができます。 Permission クラスには、オブジェクトの権限を持つ多数のユーザーの集合が含まれています。 許可オブジェクトを使って Java プログラムは、 commit() メソッドが呼び出されるまで権限変更をキャッシュすることができます。 commit() メソッドが呼び出されると、 その時点までに行われた変更はすべてサーバーに送信されます。 Permission クラスで提供される機能には、以下の機能が含まれます。

- addAuthorizedUser(): 許可ユーザーを追加する。
- commit(): 許可変更を サーバーにコミットする。
- getAuthorizationList(): オブジェクトの権限リストを戻す。
- getAuthorizedUsers(): 許可ユーザーのリストを戻す。
- getOwner(): オブジェクト所有者の名前を戻す。
- getSensitivityLevel(): オブジェクトの機密レベルを戻す。
- getType(): オブジェクト権限タイプ (QDLO、QSYS、または Root) を戻す。
- getUserPermission(): オブジェクトに対する特定ユーザーの許可を戻す。
- getUserPermissions(): オブジェクトに対するユーザー許可のリストを戻す。
- setAuthorizationList(): オブジェクトの権限リストを戻す。
- setSensitivityLevel(): オブジェクトの機密レベルを戻す。

例

この例では、許可を作成する方法とオブジェクトに許可ユーザーを追加する方法を示します。

```
// Create AS400 object
AS400 as400 = new AS400();

// Create Permission passing in the AS400 and object
Permission myPermission = new Permission(as400, "QSYS.LIB/myLib.LIB");

// Add a user to be authorized to the object
myPermission.addAuthorizedUser("User1");
```

UserPermission クラス

UserPermission クラスは、特定の 1 人のユーザーが持つ権限を表します。 UserPermission には、オブジェクト・タイプに基づいて権限を処理する 3 つのサブクラスがあります。

UserPermission クラス	説明
DLOPermission	QDLS に格納されている文書ライブラリー・オブジェクト (DLO) に対するユーザーの権限を表します。
QSYSPermission	QSYS.LIB に格納され、サーバーに含まれているオブジェクトに対するユーザーの権限を表します。
RootPermission	ルート・ディレクトリー構造に含まれるオブジェクトに対するユーザーの権限を表します。 RootPermissions オブジェクトは、 QSYS.LIB または QDLS に含まれていないオブジェクトです。

UserPermission クラスでは、以下のことが行えます。

- ユーザー・プロファイルが グループ・プロファイルであるかどうかを判別する。
- ユーザー・プロファイル名を戻す。
- ユーザーに 権限があるかどうかを示す。
- 権限リスト管理の権限を設定する。

例

以下の例では、オブジェクトに対する権限を持つユーザーおよびグループを検索して、 それらを 1 つずつ印刷する方法を示します。

```
// Create a system object.
AS400 sys = new AS400("MYAS400", "USERID", "PASSWORD");

// Represent the permissions to an object on the system, such as a library.
Permission objectInQSYS = new Permission(sys, "/QSYS.LIB/FRED.LIB");

// Retrieve the various users/groups that have permissions set on that object.
Enumeration enum = objectInQSYS.getUserPermissions();
while (enum.hasMoreElements())
{
  // Print out the user/group profile names one at a time.
  UserPermission userPerm = (UserPermission)enum.nextElement();
  System.out.println(userPerm.getUserID());
}
```

DLOPermission クラス

DLOPermission は、UserPermission のサブクラスです。 DLOPermission を使用すれば、 文書ライブラリー・オブジェクト (DLO) に対して持つ権限 (許可と呼ばれる) を表示し、設定することができます。

各ユーザーには、以下の権限値のいずれかが割り当てられます。

権限値	説明
*ALL	ユーザーは、 権限リスト管理で制御される操作以外のすべての操作を実行できます。
*AUTL	権限リストは、文書の権限を判別するために使用します。
*CHANGE	ユーザーは、オブジェクトに対する基本機能を変更し、実行することができます。
*EXCLUDE	ユーザーはオブジェクトにアクセスできません。
*USE	ユーザーは、オブジェクト操作権、読み取り権限、および実行権限を持ちます。

以下のメソッドの 1 つを使用して、 ユーザー権限を変更または判別する必要があります。

- getDataAuthority() を使用して、ユーザーの権限値を表示します。
- setDataAuthority() を使って、 ユーザーの権限値を設定します。

許可の設定後、Permissions クラスの commit() メソッドを使用して、 変更をサーバーに送信することが重要です。

許可および権限に関する詳細は、iSeries 機密保護解説書の 第 5 章:『リソース・セキュリティー』を参照してください。

例

この例では、 DLO 許可 (各許可のユーザー・プロファイルも含めて) を検索して印刷する方法を示します。

```
// Create a system object.

AS400 sys = new AS400("MYAS400", "USERID", "PASSWORD");
// Represent the permissions to a DLO object.
Permission objectInQDLS = new Permission(sys, "/QDLS/MyFolder");

// Print the object pathname and retrieve its permissions.
System.out.println("Permissions on "+objectInQDLS.getObjectPath()+" are as follows:");
Enumeration enum = objectInQDLS.getUserPermissions();
while (enum.hasMoreElements())
{
  // For each of the permissions, print out the user profile name
  // and that user's authorities to the object.
  DLOPermission dloPerm = (DLOPermission)enum.nextElement();
  System.out.println(dloPerm.getUserID()+": "+dloPerm.getDataAuthority());
}
```

QSYSPermission

QSYSPermission は、UserPermission クラスのサブクラスです。 QSYSPermission を使用すれば、QSYS.LIB に保管された従来の iSeries あるいは AS/400e ライブラリー構造にあるオブジェクトに対してユーザーが持つ許可を表示し、 設定することができます。 QSYS.LIB に保管されているオブジェクトの権限は、システム定義の権限値を設定するか、 またはオブジェクト権限とデータ権限を個別に設定することによって設定できます。

以下の表では、有効なシステム定義の権限値をリストし、説明しています。

システム定義の権限値	説明
*ALL	ユーザーは、 権限リスト管理で制御される操作以外のすべての操作を実行できます。
*AUTL	権限リストは、文書の権限を判別するために使用します。
*CHANGE	ユーザーは、オブジェクトに対する基本機能を変更し、実行することができます。
*EXCLUDE	ユーザーはオブジェクトにアクセスできません。
*USE	ユーザーは、オブジェクト操作権、読み取り権限、および実行権限を持ちます。

各システム定義の権限値は、実際には、個別のオブジェクト権限とデータ権限の組み合わせを表します。 以下の表は、個別のオブジェクト権限およびデータ権限に対するシステム定義権限の関係を示しています。

システム定義の権限	オブジェクト権限					データ権限				
	Opr	Mgt	Exist	Alter	Ref	Read	Add	Upd	Dlt	Exe
すべて	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
変更	Y	n	n	n	n	Y	Y	Y	Y	Y
除外	n	n	n	n	n	n	n	n	n	n
特殊値	Y	n	n	n	n	Y	n	n	n	Y
Autl	ユーザー (*PUBLIC) および個別のオブジェクト権限とデータ権限を判別する特定の権限リストにのみ有効。									

Y は、割り当て可能な権限を指します。
n は、割り当て不可能な権限を指します。

システム定義の権限を指定すると、それに応じた個別の権限が自動的に割り当てられます。 同様に、さまざまな個別の権限を指定すると、それに応じた個別の権限値が変更されます。 個別のオブジェクト権限とデータ権限の組み合わせが、単一のシステム定義の権限値にマップしない時は、 その単一値は、「ユーザー定義」になります。

現行のシステム定義の権限を表示するには、getObjectAuthority() メソッドを使用します。 単一値を使用して現行のシステム定義の権限を設定するには、 setObjectAuthority() メソッドを使用します。

個別のオブジェクト権限値をオンまたはオフに設定するには、該当する set メソッドを使用してください。

- setAlter()
- setExistence()
- setManagement()
- setOperational()
- setReference()

個別のデータ権限値をオンまたはオフに設定するには、該当する set メソッドを使用してください。

- setAdd()
- setDelete()
- setExecute()
- setRead()
- setUpdate()

種々の権限の詳細については、iSeries 機密保護解説書  の第 5 章：『リソース・セキュリティー』を参照してください。 オブジェクト権限の権限付与および編集をするために iSeries CL コマンドを使用する時の詳細については、 オブジェクト権限を認可する (GRTOBJAUT) およびオブジェクト権限を編集する (EDTOBJAUT) を参照してください。

例

この例では、QSYS オブジェクトの許可を検索して印刷する方法を示します。

```
// Create a system object.
AS400 sys = new AS400("MYAS400", "USERID", "PASSWORD");

// Represent the permissions to a QSYS object.
Permission objectInQSYS = new Permission(sys, "/QSYS.LIB/FRED.LIB");

// Print the object pathname and retrieve its permissions.
System.out.println("Permissions on "+objectInQSYS.getObjectPath()+" are as follows:");
Enumeration enum = objectInQSYS.getUserPermissions();
while (enum.hasMoreElements())
{
  // For each of the permissions, print out the user profile name
  // and that user's authorities to the object.
  QSYSPermission qsysPerm = (QSYSPermission)enum.nextElement();
  System.out.println(qsysPerm.getUserID()+": "+qsysPerm.getObjectAuthority());
}
```

RootPermission

RootPermission は、UserPermission クラスのサブクラスです。 RootPermission クラスを使用すると、ルート・ディレクトリー構造に含まれているオブジェクトを使用するユーザーの許可を表示したり設定することができます。

ルート・ディレクトリー構造に置かれているオブジェクトでは、 データ権限またはオブジェクト権限を設定することができます。 データ権限は、以下の表にリストする値に設定できます。 現在の値を表示するには、getDataAuthority() メソッドを使用し、 データ権限を設定するには、setDataAuthority() メソッドを使用します。

以下の表では、有効なデータ権限値をリストし、説明しています。

データ権限値	説明
*none	ユーザーに、オブジェクトに対する権限を与えません。
*RWX	ユーザーに、読み取り、追加、更新、削除、および実行権限を与えます。
*RW	ユーザーに、読み取り、追加、および削除権限を与えます。
*RX	ユーザーに、読み取りおよび実行権限を与えます。
*WX	ユーザーに、追加、更新、削除、および実行権限を与えます。
*R	ユーザーに、読み取り権限を与えます。
*W	ユーザーに、追加、更新、および削除権限を与えます。
*X	ユーザーに、実行権限を与えます。
*EXCLUDE	ユーザーはオブジェクトにアクセスできません。
*AUTL	このオブジェクトに対する共通権限は、権限リストによって与えられます。

オブジェクト権限は、alter、existence、management、または reference のいずれか 1 つまたは複数に設定できます。 setAlter()、 setExistence()、 setManagement()、 または setReference() メソッドを使用して、値をオンまたはオフに設定することができます。

オブジェクトのデータ権限またはオブジェクト権限を設定した後は、 許可 クラスの commit() メソッドを使用して、 変更をサーバーに送信することが重要です。

種々の権限の詳細については、iSeries 機密保護解説書 の第 5 章：『リソース・セキュリティー』を参照してください。

例

この例では、ルート・オブジェクトの許可を検索して印刷する方法を示します。

```
// Create a system object.
AS400 sys = new AS400("MYAS400", "USERID", "PASSWORD");

// Represent the permissions to an object in the root file system.
Permission objectInRoot = new Permission(sys, "/fred");

// Print the object pathname and retrieve its permissions.
System.out.println("Permissions on "+objectInRoot.getObjectPath()+" are as follows:");
Enumeration enum = objectInRoot.getUserPermissions();
while (enum.hasMoreElements())
{
  // For each of the permissions, print out the user profile name
  // and that user's authorities to the object.
```

```
  RootPermission rootPerm = (RootPermission)enum.nextElement();
  System.out.println(rootPerm.getUserID()+": "+rootPerm.getDataAuthority());
}
```

印刷クラス

スプール・ファイル、出力待ち行列、プリンター、プリンター・ファイル、 書き込み機能ジョブ、および高機能印刷 (AFP) リソース (フォント、フォーム定義、 オーバーレイ、ページ定義、およびページ・セグメントを含む) といったオブジェクトを印刷します。 AFP リソースにアクセスできるのは、OS/400 のバージョン 3 リリース 7 (V3R7) およびそれ以降のバージョンだけです。 (V3R7 より前のバージョンを実行しているシステムで AFPResourceList をオープンしようとすると、 RequestNotSupportedException 例外が生成されます。)

印刷オブジェクト用の IBM Toolbox for Java クラスは、 基本クラス PrintObject と、 6 種類の印刷オブジェクトごとのサブクラスで編成されています。 基本クラスには、すべてのサーバー印刷オブジェクトに共通のメソッドと属性が含まれています。 サブクラスには、個々のサブタイプに固有のメソッドと属性が含まれています。

以下の目的で印刷クラスを使用してください。

- サーバー印刷オブジェクトの処理:
  - PrintObjectList クラス - サーバー印刷オブジェクトをリストしたり処理したりするのに使用します。 (印刷オブジェクトにはスプール・ファイル、出力待ち行列、プリンター、高機能印刷 (AFP) リソース、 プリンター・ファイル、および書き込み機能ジョブが含まれます。)
  - PrintObject 基本クラス - 印刷オブジェクトを用いる作業に使用します。
- PrintObject 属性の取り出し
- SpooledFileOutputStream クラスを使用した新しいサーバー・スプール・ファイルの作成 (EBCDIC ベースのプリンター・データ用に使用)
- SNA 文字ストリーム (SCS) プリンター・データ・ストリームの生成
- PrintObjectInputStream を使用したスプール・ファイルと AFP リソースの読み取り
- PrintObjectPageInputStream および PrintObjectTransformedInputStream を使用したスプール・ファイルの読み取り
- 高機能印刷 (AFP) と SNA 文字ストリーム (SCS) スプール・ファイルの表示

例

- スプール・ファイルの作成の例には、 サーバー上で入力ストリームからスプール・ファイルを作成する方法が示されています。
- SCS スプール・ファイルの作成の例には、 SCS3812Writer クラスを使用して SCS データ・ストリームを生成する方法と、 このストリームをサーバー上でスプール・ファイルに書き込む方法が示されています。
- スプール・ファイルの読み取りの例では、 既存のサーバー・スプール・ファイルを読み取る方法が示されています。
- 1 つ目の非同期リストの例には、 システム上のスプール・ファイルをすべて非同期的にリストする方法と、 そのリストの構築時に PrintObjectListListener インターフェースを使用して フィードバックを取得する方法を示します。
- 2 つ目の非同期リストの例には、 PrintObjectListListener インターフェースを使用しないで システム上のスプール・ファイルをすべて非同期的にリストする方法を示します。
- 同期リストの例には、 システム上のスプール・ファイルをすべて同期的にリストする方法を示します。

印刷オブジェクトのリスト

PrintObjectList クラスとそのサブクラスを使用すると、印刷オブジェクトのリストを処理することができます。 各サブクラスにはメソッドがあり、このメソッドを使用すると、 その特定のタイプの印刷オブジェクトに適した基準に基づいてリストをフィルター操作することができます。 たとえば、SpooledFileList では、 スプール・ファイルを作成したユーザー、スプール・ファイルが入っている出力待ち行列、 用紙タイプ、またはスプール・ファイルのユーザー・データに基づいて、 スプール・ファイルのリストをフィルター操作することができます。 フィルター基準に適合するスプール・ファイルだけがリストされます。 フィルターが設定されない場合は、各フィルターのデフォルトが使用されます。

実際にサーバーから印刷オブジェクトのリストを取り出すためには、 openSynchronously() または openAsynchronously() メソッドが使用されます。 openSynchronously() メソッドは、 サーバーからリスト内のすべてのオブジェクトが取り出されるまでは戻されません。 openAsynchronously() メソッドはすぐに戻されます。 呼び出し元では、リストが作成されるのを待つ間に、フォアグラウンドで別のジョブを行うことができます。 また、非同期にオープンしたリストを使用することにより、呼び出し元では、 オブジェクトが戻されたときにそれをユーザーに表示し始めることもできます。 ユーザーはオブジェクトが戻されたときにそれを見ることができるため、 応答時間が速いように感じるかもしません。 実際には、リスト内の各オブジェクトについて余分の処理が行われるため、 全体の応答時間は長くなる可能性があります。

リストを非同期にオープンする場合、 呼び出し元はリストの作成についてのフィードバックを取得することができます。 isCompleted() および size() などのメソッドは、リストの作成が完了したかどうかを示すか、あるいはリストの現在のサイズを戻します。 そのほかに、メソッド waitForListToComplete() および waitForItem() を使用すると、呼び出し元では、リストの完成や特定の項目を待つことができます。 このような PrintObjectList メソッドを呼び出すことに加え、 呼び出し元はリスナーとしてリストに登録することができます。 この場合、呼び出し元はリストに生じたイベントを通知されます。 イベントについて登録あるいは抹消する場合、呼び出し元は、 PrintObjectListListener() を使用し、さらに addPrintObjectListListener() を呼び出して登録するか、または removePrintObjectListListener() を呼び出して抹消します。 次の表では、PrintObjectList によって送達されるイベントが示されています。

PrintObjectList イベント	イベントが引き渡された時
listClosed	リストのクローズ時。
listCompleted	リストの完成時。

listErrorOccurred	リストの取り出し中に例外が生じた場合。
listOpened	リストのオープン時。
listObjectAdded	オブジェクトをリストに追加するとき。

リストがオープンされ、リスト内のオブジェクトが処理された後、 close() メソッドを使用してリストをクローズしてください。 これにより、オープン時にガーベッジ・コレクターに割り振られたリソースが解放されます。 リストをクローズした後、そのフィルターを変更し、もう一度リストをオープンすることができます。

印刷オブジェクトがリストされるときには、 リストされる各印刷オブジェクトについての属性がサーバーから送信され、 印刷オブジェクトと一緒に保管されます。 これらの属性は、PrintObject クラスの update() メソッドを用いて更新することができます。 サーバーから送り返される属性、リストされる印刷オブジェクトのタイプによって異なります。 印刷オブジェクトのタイプごとに属性のデフォルト・リストが存在しますが、 それは PrintObjectList 内の setAttributesToRetrieve() メソッドを使用してオーバーライドすることができます。 それぞれのタイプの印刷オブジェクトでサポートされる属性のリストについては、PrintObject 属性の取り出しのセクションを参照してください。

AFP リソースのリスト作成は、 OS/400 のバージョン 3 リリース 7 およびその後続リリースでのみ許可されます。 V3R7 より前のシステムに対して AFPResourceList をオープンすると、 RequestNotSupportedException 例外が生成されます。

例

非同期リストの例 1

非同期リストの例 2

同期リストの例

印刷オブジェクトの処理

PrintObject は抽象クラスです。(抽象クラスでは、クラスのインスタンスを作成することができません。 その代わり、そのサブクラスの 1 つのインスタンスを作成しなければなりません。) 以下の方法のいずれかにより、 サブクラスのオブジェクトを作成してください。

- オブジェクトのシステム、および識別属性を知っている場合、 パブリック・コンストラクターを呼び出してそのオブジェクトを明示的に作成することができます。
- PrintObjectList サブクラスを使用して、オブジェクトのリストを作成してから、 そのリストを通じて個々のオブジェクトを取得することができます。
- 1 つのメソッドあるいはセットのメソッドの呼び出しの結果として、 オブジェクトを作成して戻すことができます。 たとえば、 WriterJob クラス内の静的メソッド start() は、WriterJob オブジェクトを戻します。

基本クラス PrintObject とそのサブクラスは、以下のサーバー印刷オブジェクトを処理するために使用します。

- OutputQueue
- Printer
- PrinterFile
- SpooledFile
- WriterJob

PrintObject 属性の取り出し

属性 ID と、基本 PrintObject クラスからのいずれかのメソッドを使用することによって、 印刷オブジェクト属性を取り出すことができます。

- getIntegerAttribute(int attributeID) を使用して、整数タイプの属性を取り出します。
- getFloatAttribute(int attributeID) を使用して、浮動小数点タイプの属性を取り出します。
- getStringAttribute(int attributeID) を使用して、STRING タイプの属性を取り出します。

attributeID パラメーターは整数であり、取り出す属性を識別します。 すべての ID は、基本 PrintObject クラスでパブリック定数として定義されています。 PrintAttributes ファイルには、 各属性 ID のエントリーが入っています。 このエントリーには、属性とそのタイプ (整数、浮動小数点、 あるいは STRING) についての記述が組み込まれています。 このようなメソッドを使用して取り出せる属性のリストについては、 以下のリンクを選択してください。

- AFP リソースの場合、AFPResourceAttrs
- 出力待ち行列の場合、OutputQueueAttrs
- プリンターの場合、PrinterAttrs
- プリンター・ファイルの場合、PrinterFileAttrs
- スプール・ファイルの場合、SpooledFileAttrs
- 書き込みジョブの場合、WriterJobAttrs

受け入れ可能なパフォーマンスを達成するために、 これらの属性はクライアントにコピーされます。 これらの属性は、オブジェクトがリストされたとき、あるいはオブジェクトが暗黙的に作成されてから初めて属性が必要になったときのいずれかに、 コピーされます。 これによって、アプリケーションが属性を取り出すことが必要になるたびに、 オブジェクトがホストに送られることがなくなります。 さらに、Java 印刷オブジェクトのインスタンスが、 サーバー上のオブジェクトについての古い情報を含めることが可能になります。 オブジェクトのユーザーは、 オブジェクトに対して update() メソッドを呼び出すことによって、 すべての属性を最新表示することができます。さらに、アプリケーションが、 オブジェクトの属性を変更させるメソッドをオブジェクトに対して呼び出すと、 属性は自動的に更新されます。 たとえば、ある出力待ち行列の状況属性が RELEASED であり (getStringAttribute(ATTR_OUTQSTS); が "RELEASED" を戻す)、 その出力待ち行列に対して hold() メソッドが呼び出された後に、 状況属性を取得すると、HELD が戻されます。

setAttributes メソッド

setAttributes メソッドを使用すると、 スプール・ファイルおよびプリンター・ファイル・オブジェクトの属性を変更することができます。 設定できる属性のリストについては、以下のリンクを選択してください。

- プリンター・ファイルの場合、PrinterFileAttrs
- スプール・ファイルの場合、SpooledFileAttrs

setAttributes メソッドは、 PrintParameterList パラメーターを取ります。 これは、属性 ID とその値のコレクションを保持するために使用されるクラスです。 リストは最初は空であり、 呼び出し元はリストに対してさまざまな setParameter() メソッドを使用することによって、 リストに属性を追加することができます。

PrintParameterList クラス

PrintParameterList クラスを使用すると、 任意の数の属性をパラメーターとして取るメソッドに属性のグループを渡すことができます。 たとえば、SpooledFile メソッド sendTCP() を使用すると、TCP (LPR) を用いてスプール・ファイルを送信することができます。 PrintParameterList オブジェクトには、 送信コマンド用の必須パラメーター (リモート・システムおよび待ち行列など) と、 必要な任意指定パラメーター (スプール・ファイルを送信後に削除するかどうかなど) が含まれます。 このような場合、メソッドの資料には、必須および任意指定の属性のリストが示されています。 PrintParameterList setParameter() メソッドでは、設定される属性と、それらの属性に設定される値が検査されません。 PrintParameterList setParameter() メソッドには、 このメソッドに渡す値だけしか含まれません。 一般に、 PrintParameterList 内の余分な属性は無視され、 使用される属性についての無効な値はサーバー上で診断されます。

プリンター・ファイルの属性

属性の検索

該当する getIntegerAttribute()、getStringAttribute()、または getFloatAttribute() メソッドを使って、 以下のようなプリンター・ファイルの属性を検索することができます。

- ATTR_ALIGN - ページの位置合わせ
- ATTR_BKMGN_ACR - バック・マージン横方向オフセット
- ATTR_BKMGN_DWN - バック・マージン下方向オフセット
- ATTR_BACK_OVERLAY - 背面オーバーレイの統合ファイル・システム名
- ATTR_BKOVL_DWN - 背面オーバーレイ下方向オフセット
- ATTR_BKOVL_ACR - 背面オーバーレイ横方向オフセット
- ATTR_CPI - 1 インチ当たりの文字数
- ATTR_CODEDFNTLIB - コード化フォント・ライブラリー名
- ATTR_CODEPAGE - コード・ページ
- ATTR_CODEDFNT - コード化フォント名
- ATTR_CONTROLCHAR - 制御文字
- ATTR_CONVERT_LINEDATA - 行データの変換
- ATTR_COPIES - コピー枚数
- ATTR_CORNER_STAPLE - コーナー・ステープルとじ
- ATTR_DBCSDATA - ユーザー指定の DBCS データ
- ATTR_DBCSEXTENSN - DBCS 拡張文字
- ATTR_DBCSROTATE - DBCS の回転
- ATTR_DBCSCPI - インチ当たりの DBCS 数
- ATTR_DBCSSISO - DBCS の SO/SI のスペース
- ATTR_DFR_WRITE - 据え置き書き出し
- ATTR_PAGRTT - ページ回転の角度
- ATTR_EDGESTITCH_NUMSTAPLES - 平とじステープル数
- ATTR_EDGESTITCH_REF - 平とじ参照
- ATTR_EDGESTITCH_REFOFF - 平とじ参照

- ATTR_ENDPAGE - 終了ページ
- ATTR_FILESEP - ファイル区切り
- ATTR_FOLDREC - レコードの折り返し
- ATTR_FONTID - フォント ID
- ATTR_FORM_DEFINITION - 用紙定義の統合ファイル・システム名
- ATTR_FORMFEED - 用紙送り
- ATTR_FORMTYPE - 用紙タイプ
- ATTR_FTMGN_ACR - フロント・マージン横方向オフセット
- ATTR_FTMGN_DWN - フロント・マージン下方向オフセット
- ATTR_FRONT_OVERLAY - フロント・オーバーレイの統合ファイル・システム名
- ATTR_FTOVL_ACR - フロント・オーバーレイ横方向オフセット
- ATTR_FTOVL_DWN - フロント・オーバーレイ下方向オフセット
- ATTR_CHAR_ID - 図形文字セット
- ATTR_JUSTIFY - ハードウェア行末調整
- ATTR_HOLD - スプール・ファイル保留
- ATTR_LPI - 1 インチ当たりの行数
- ATTR_MAXRCDS - 最大スプール出力レコード数
- ATTR_OUTPTY - 出力優先順位
- ATTR_OUTPUT_QUEUE - 出力待ち行列の統合ファイル・システム名
- ATTR_OVERFLOW - オーバーフロー行番号
- ATTR_PAGE_DEFINITION - ページ定義の統合ファイル・システム
- ATTR_PAGELEN - ページの長さ
- ATTR_MEASMETHOD - 測定方法
- ATTR_PAGEWIDTH - ページ幅
- ATTR_MULTIUP - 面当たりページ数
- ATTR_POINTSIZE - ポイント・サイズ
- ATTR_FIDELITY - 印刷精度
- ATTR_DUPLEX - 両面印刷
- ATTR_PRTQUALITY - 印刷品質
- ATTR_PRTTEXT - 印刷テキスト

- ATTR_PRINTER - プリンター
- ATTR_PRTDEVTYPE - 印刷装置タイプ
- ATTR_RPLUNPRT - 印刷不能文字の置き換え
- ATTR_RPLCHAR - 置き換え文字
- ATTR_SADDLESTITCH_NUMSTAPLES - 中とじステープル数
- ATTR_SADDLESTITCH_REF - 中とじ参照
- ATTR_SAVE -スプール・ファイルの保管
- ATTR_SRCDRWR - ソース用紙入れ
- ATTR_SPOOL - データのスプール
- ATTR_SCHEDULE - スプール出力スケジュール
- ATTR_STARTPAGE - 開始ページ
- ATTR_DESCRIPTION - テキスト記述
- ATTR_UNITOFMEAS - 測定単位
- ATTR_USERDATA - ユーザー・データ
- ATTR_USRDEFDATA - ユーザー定義データ
- ATTR_USRDEFOPT - ユーザー定義オプション
- ATTR_USER_DEFINED_OBJECT - ユーザー定義オブジェクトの統合ファイル・システム名

属性の設定

setAttributes() メソッドを使って、 以下のようなプリンター・ファイルの属性を設定することができます。

- ATTR_ALIGN - ページの位置合わせ
- ATTR_BKMGN_ACR - バック・マージン横方向オフセット
- ATTR_BKMGN_DWN - バック・マージン下方向オフセット
- ATTR_BACK_OVERLAY - 背面オーバーレイの統合ファイル・システム名
- ATTR_BKOVL_DWN - 背面オーバーレイ下方向オフセット
- ATTR_BKOVL_ACR - 背面オーバーレイ横方向オフセット
- ATTR_CPI - 1 インチ当たりの文字数
- ATTR_CODEDFNTLIB - コード化フォント・ライブラリー名

- ATTR_CODEPAGE - コード・ページ
- ATTR_CODEDFNT - コード化フォント名
- ATTR_CONTROLCHAR - 制御文字
- ATTR_CONVERT_LINEDATA - 行データの変換
- ATTR_COPIES - コピー枚数
- ATTR_CORNER_STAPLE - コーナー・ステープルとじ
- ATTR_DBCSDATA - ユーザー指定の DBCS データ
- ATTR_DBCSEXTENSN - DBCS 拡張文字
- ATTR_DBCSROTATE - DBCS の回転
- ATTR_DBCSCPI - インチ当たりの DBCS 数
- ATTR_DBCSSISO - DBCS の SO/SI のスペース
- ATTR_DFR_WRITE - 据え置き書き出し
- ATTR_PAGRTT - ページ回転の角度
- ATTR_EDGESTITCH_NUMSTAPLES - 平とじステープル数
- ATTR_EDGESTITCH_REF - 平とじ参照
- ATTR_EDGESTITCH_REFOFF - 平とじ参照
- ATTR_ENDPAGE - 終了ページ
- ATTR_FILESEP - ファイル区切り
- ATTR_FOLDREC - レコードの折り返し
- ATTR_FONTID - フォント ID
- ATTR_FORM_DEFINITION - 用紙定義の統合ファイル・システム名
- ATTR_FORMFEED - 用紙送り
- ATTR_FORMTYPE - 用紙タイプ
- ATTR_FTMGN_ACR - フロント・マージン横方向オフセット
- ATTR_FTMGN_DWN - フロント・マージン下方向オフセット
- ATTR_FRONT_OVERLAY - フロント・オーバーレイの統合ファイル・システム名
- ATTR_FTOVL_ACR - フロント・オーバーレイ横方向オフセット
- ATTR_FTOVL_DWN - フロント・オーバーレイ下方向オフセット
- ATTR_CHAR_ID - 図形文字セット
- ATTR_JUSTIFY - ハードウェア行末調整

- ATTR_HOLD - スプール・ファイル保留
- ATTR_LPI - 1 インチ当たりの行数
- ATTR_MAXRCDS - 最大スプール出力レコード数
- ATTR_OUTPTY - 出力優先順位
- ATTR_OUTPUT_QUEUE - 出力待ち行列の統合ファイル・システム名
- ATTR_OVERFLOW - オーバーフロー行番号
- ATTR_PAGE_DEFINITION - ページ定義の統合ファイル・システム
- ATTR_PAGELEN - ページの長さ
- ATTR_MEASMETHOD - 測定方法
- ATTR_PAGEWIDTH - ページ幅
- ATTR_MULTIUP - 面当たりページ数
- ATTR_POINTSIZE - ポイント・サイズ
- ATTR_FIDELITY - 印刷精度
- ATTR_DUPLEX - 両面印刷
- ATTR_PRTQUALITY - 印刷品質
- ATTR_PRTTEXT - 印刷テキスト
- ATTR_PRINTER - プリンター
- ATTR_PRTDEVTYPE - 印刷装置タイプ
- ATTR_RPLUNPRT - 印刷不能文字の置き換え
- ATTR_RPLCHAR - 置き換え文字
- ATTR_SADDLESTITCH_NUMSTAPLES - 中とじステープル数
- ATTR_SADDLESTITCH_REF - 中とじ参照
- ATTR_SAVE -スプール・ファイルの保管
- ATTR_SRCDRWR - ソース用紙入れ
- ATTR_SPOOL - データのスプール
- ATTR_SCHEDULE - スプール出力スケジュール
- ATTR_STARTPAGE - 開始ページ
- ATTR_DESCRIPTION - テキスト記述
- ATTR_UNITOFMEAS - 測定単位
- ATTR_USERDATA - ユーザー・データ
- ATTR_USRDEFDATA - ユーザー定義データ

- ATTR_USRDEFOPT - ユーザー定義オプション
- ATTR_USER_DEFINED_OBJECT - ユーザー定義オブジェクトの統合ファイル・システム名

スプール・ファイルの属性

属性の検索

該当する getIntegerAttribute()、getStringAttribute()、または getFloatAttribute() メソッドを使って、 以下のようなスプール・ファイルの属性を検索することができます。

- ATTR_AFP - 高機能印刷
- ATTR_ALIGN - ページの位置合わせ
- ATTR_BKMGN_ACR - 背面オーバーレイ横方向オフセット
- ATTR_BKMGN_DWN - 背面オーバーレイ下方向オフセット
- ATTR_BACK_OVERLAY - 背面オーバーレイの統合ファイル・システム名
- ATTR_BKOVL_DWN - 背面オーバーレイ下方向オフセット
- ATTR_BKOVL_ACR - 背面オーバーレイ横方向オフセット
- ATTR_CPI - 1 インチ当たりの文字数
- ATTR_CODEDFNTLIB - コード化フォント・ライブラリー名
- ATTR_CODEDFNT - コード化フォント名
- ATTR_CODEPAGE - コード・ページ
- ATTR_CONTROLCHAR - 制御文字
- ATTR_COPIES - コピー枚数
- ATTR_COPIESLEFT - 作成されていない残りのコピー
- ATTR_CORNER_STAPLE - コーナー・ステープルとじ
- ATTR_CURPAGE - 現在のページ
- ATTR_DATE - オブジェクトの作成日付
- ATTR_DATE_WTR_BEGAN_FILE - 書き出しプログラムがスプール・ファイルの処理を開始した日付
- ATTR_DATE_WTR_CMPL_FILE - 書き出しプログラムがスプール・ファイルの処理を完了した日付
- ATTR_DBCSDATA - ユーザー指定の DBCS データ
- ATTR_DBCSEXTENSN - DBCS 拡張文字

- ATTR_DBCSROTATE - DBCS の回転
- ATTR_DBCSCPI - インチ当たりの DBCS 数
- ATTR_DBCSSISO - DBCS の SO/SI のスペース
- ATTR_PAGRTT - ページ回転の角度
- ATTR_EDGESTITCH_NUMSTAPLES - 平とじステープル数
- ATTR_EDGESTITCH_REF - 平とじ参照
- ATTR_EDGESTITCH_REFOFF - 平とじ参照 オフセット
- ATTR_ENDPAGE - 終了ページ
- ATTR_FILESEP - ファイル区切り
- ATTR_FOLDREC - レコードの折り返し
- ATTR_FONTID - フォント ID
- ATTR_FORM_DEFINITION - 用紙定義の統合ファイル・システム名
- ATTR_FORMFEED - 用紙送り
- ATTR_FORMTYPE - 用紙タイプ
- ATTR_FTMGN_ACR - フロント・マージン横方向オフセット
- ATTR_FTMGN_DWN - フロント・マージン下方向オフセット
- ATTR_FRONTSIDE_OVERLAY - フロント・オーバーレイの統合ファイル・システム名
- ATTR_FTOVL_ACR - フロント・オーバーレイ横方向オフセット
- ATTR_FTOVL_DWN - フロント・オーバーレイ下方向オフセット
- ATTR_CHAR_ID - 図形文字セット
- ATTR_JUSTIFY - ハードウェア行末調整
- ATTR_HOLD - スプール・ファイル保留
- ATTR_IPP_ATTR_CHARSET - IPP 属性文字セット
- ATTR_IPP_JOB_ID - IPP ジョブ ID
- ATTR_IPP_JOB_NAME - IPP ジョブ名
- ATTR_IPP_JOB_NAME_NL - IPP ジョブ名 NL
- ATTR_IPP_JOB_ORIGUSER - IPP ジョブ発信ユーザー
- ATTR_IPP_JOB_ORIGUSER_NL - IPP ジョブ発信ユーザー NL
- ATTR_IPP_PRINTER_NAME - IPP プリンター名
- ATTR_JOBNAME - ジョブ名

- ATTR_JOBNUMBER - ジョブ番号
- ATTR_JOBUSER - ジョブのユーザー
- » ATTR_JOB_SYSTEM - ジョブ・システム «
- ATTR_LASTPAGE - 印刷する最終ページ
- ATTR_LINESPACING - 行の間隔
- ATTR_LPI - 1 インチ当たりの行数
- ATTR_MAXRCDS - 最大スプール出力レコード数
- ATTR_PAGELEN - ページの長さ
- ATTR_PAGEWIDTH - ページ幅
- ATTR_MEASMETHOD - 測定方法
- ATTR_NETWORK - ネットワーク ID
- ATTR_NUMBYTES - 読み取り/書き込みのバイト数
- ATTR_OUTPUTBIN - 出力ビン
- ATTR_OUTPTY - 出力優先順位
- ATTR_OUTPUT_QUEUE - 出力待ち行列の統合ファイル・システム名
- ATTR_OVERFLOW - オーバーフロー行番号
- ATTR_MULTIUP - 面当たりページ数
- ATTR_POINTSIZE - ポイント・サイズ
- ATTR_FIDELITY - 印刷精度
- ATTR_DUPLEX - 両面印刷
- ATTR_PRTQUALITY - 印刷品質
- ATTR_PRTTEXT - 印刷テキスト
- ATTR_PRINTER - プリンター
- ATTR_PRTASSIGNED - 割り当てプリンター
- ATTR_PRTDEVTYPE - 印刷装置タイプ
- ATTR_PRINTER_FILE - プリンター・ファイルの統合ファイル・システム名
- ATTR_RECLENGTH - レコード長
- ATTR_REDUCE - 出力の削減
- ATTR_RPLUNPRT - 印刷不能文字の置き換え
- ATTR_RPLCHAR - 置き換え文字

- ATTR_RESTART - 印刷の再始動
- ATTR_SADDLESTITCH_NUMSTAPLES - 中とじステープル数
- ATTR_SADDLESTITCH_REF - 中とじ参照
- ATTR_SAVE - スプール・ファイルの保管
- ATTR_SRCDRWR - ソース用紙入れ
- ATTR_SPOOLFILE - スプール・ファイル名
- ATTR_SPLFNUM - スプール・ファイル番号
- ATTR_SPLFSTATUS - スプール・ファイル状況
- ATTR_SCHEDULE - スプール出力スケジュール
- ATTR_STARTPAGE - 開始ページ
- ATTR_SYSTEM - 作成したシステム
- ATTR_TIME - オブジェクトの作成時刻
- ATTR_TIME_WTR_BEGAN_FILE - 書き出しプログラムがスプール・ファイルの処理を開始した時刻
- ATTR_TIME_WTR_CMPL_FILE - 書き出しプログラムがスプール・ファイルの処理を完了した時刻
- ATTR_PAGES - 合計ページ数
- ATTR_UNITOFMEAS - 測定単位
- ATTR_USERCMT - ユーザー注記
- ATTR_USERDATA - ユーザー・データ
- ATTR_USRDEFDATA - ユーザー定義データ
- ATTR_USRDEFFILE - ユーザー定義ファイル
- ATTR_USRDEFOPT - ユーザー定義オプション
- ATTR_USER_DEFINED_OBJECT - ユーザー定義オブジェクトの統合ファイル・システム名

属性の設定

setAttributes() メソッドを使って、 以下のようなスプール・ファイルの属性を設定することができます。

- ATTR_ALIGN - ページの位置合わせ
- ATTR_BACK_OVERLAY - 背面オーバーレイの統合ファイル・システム

名

- ATTR_BKOVL_DWN - 背面オーバーレイ下方向オフセット
- ATTR_BKOVL_ACR - 背面オーバーレイ横方向オフセット
- ATTR_COPIES - コピー枚数
- ATTR_ENDPAGE - 終了ページ
- ATTR_FILESEP - ファイル区切り
- ATTR_FORM_DEFINITION - 用紙定義の統合ファイル・システム名
- ATTR_FORMFEED - 用紙送り
- ATTR_FORMTYPE - 用紙タイプ
- ATTR_FRONTSIDE_OVERLAY - フロント・オーバーレイの統合ファイル・システム名
- ATTR_FTOVL_ACR - フロント・オーバーレイ横方向オフセット
- ATTR_FTOVL_DWN - フロント・オーバーレイ下方向オフセット
- ATTR_OUTPTY - 出力優先順位
- ATTR_OUTPUT_QUEUE - 出力待ち行列の統合ファイル・システム名
- ATTR_MULTIUP - 面当たりページ数
- ATTR_FIDELITY - 印刷精度
- ATTR_DUPLEX - 両面印刷
- ATTR_PRTQUALITY - 印刷品質
- ATTR_PRTSEQUENCE - 印刷順序
- ATTR_PRINTER - プリンター
- ATTR_RESTART - 印刷の再始動
- ATTR_SAVE - スプール・ファイルの保管
- ATTR_SCHEDULE - スプール出力スケジュール
- ATTR_STARTPAGE - 開始ページ
- ATTR_USERDATA - ユーザー・データ
- ATTR_USRDEFOPT - ユーザー定義オプション
- ATTR_USER_DEFINED_OBJECT - ユーザー定義オブジェクトの統合ファイル・システム名

新規スプール・ファイルの作成

SpooledFileOutputStream クラスを使用すると、 新規のサーバー・スプール・ファイルを作成することができます。 このクラスは標準 JDK java.io.OutputStream クラスから派生したものであり、 構成後は、OutputStream が使用されている任意の箇所で使用することができます。

新しい SpooledFileOutputStream を作成するときには、 呼び出し元は以下のものを指定することができます。

- 使用するプリンター・ファイル
- スプール・ファイルを入れる出力待ち行列
- プリンター・ファイル内のフィールドをオーバーライドするパラメーターを含められる PrintParameterList オブジェクト

これらのパラメーターは、すべて任意指定です (呼び出し元では、 これらの一部あるいはすべてをヌルにして渡すことができます)。 プリンター・ファイルを指定しない場合、ネットワーク印刷サーバーは、 ネットワーク印刷のデフォルト・プリンター・ファイルである QPNPSPRTF を使用します。 出力待ち行列パラメーターは便宜上存在しますが、 PrintParameterList でも指定することができます。 出力待ち行列パラメーターを両方の場所に指定すると、 PrintParameterList フィールドが出力待ち行列パラメーターをオーバーライドします。 新規スプール・ファイルを作成するために PrintParameterList で設定できる属性の完全なリストについては、SpooledFileOutputStream コンストラクターの資料を参照してください。

スプール・ファイルにデータを書き込むには、 いずれかの write() メソッドを使用します。 SpooledFileOutputStream オブジェクトはデータをバッファーに入れ、 出力ストリームがクローズされるか、またはバッファーが満杯になったときに、 そのデータを送信します。 バッファリングは、以下の 2 つの理由で行われます。

- バッファリングにより、 満杯になったバッファーのデータを分析してデータ・タイプを判別するための自動データ・タイプ指定が可能になる (スプール・ファイル内のデータ・ストリーム・タイプを参照してください)。
- 個々の書き込み要求がサーバーに送信されないため、出力ストリームの処理が速くなる。

データを強制的にサーバーに書き込むには、 flush() メソッドを使用します。

新規スプール・ファイルへのデータの書き込みが終了すると、 close() メソッドが呼び出されてスプール・ファイルがクローズされます。 いったんスプール・ファイルがクローズされると、 そのファイルにそれ以上データを書き込むこと

ができなくなります。 スプール・ファイルのクローズ後に getSpooledFile() メソッドを呼び出すと、 呼び出し元では、そのスプール・ファイルを表す SpooledFile オブジェクトへの参照を取得することができます。

スプール・ファイル内のデータ・ストリーム・タイプ

スプール・ファイルに入れられるデータのタイプを設定するには、 スプール・ファイルの「Printer Data Type (プリンター・データ・タイプ)」属性を使用してください。 呼び出し元がプリンター・データ・タイプを指定しない場合、 デフォルトは自動データ・タイプ指定の使用です。 このメソッドでは、スプール・ファイル・データの最初の数千バイトを調べて、 それが SNA 文字ストリーム (SCS) と高機能印刷データ・ストリーム (AFPDS) のどちらのデータ・ストリーム・アーキテクチャーに適合するかを判別し、 それに応じて属性を設定します。 スプール・ファイル・データのバイトがどちらのアーキテクチャーとも一致しない場合、 そのデータは *USERASCII とタグ付けされます。 自動データ・タイプ指定は、ほとんどの場合うまくいきます。 呼び出し元では、自動データ・タイプ指定がうまくいかないような特殊なケースがある場合を除き、通常、この自動データ・タイプ指定を使用すべきです。 そのようなケースでは、呼び出し元は Printer Data Type 属性を特定の値 (たとえば、*SCS) に設定することができます。 呼び出し元でプリンター・ファイル内のプリンター・データを使用したい場合には、 特殊値 *PRTF を使用しなければなりません。 呼び出し元でスプール・ファイルの作成時にデフォルトのデータ・タイプをオーバーライドする場合は、 スプール・ファイルに入れられるデータがデータ・タイプ属性と一致するように、 注意を払わなければなりません。 SCS データを受信するとマークされているスプール・ファイルに非 SCS データを入れると、ホストからエラー・メッセージが出され、スプール・ファイルが失われます。

通常、この属性には次の 3 つの値があります。

- *SCS - EBCDIC で、テキスト・ベースのプリンター・データ・ストリーム。
- *AFPDS (高機能印刷データ・ストリーム) - サーバー上でサポートされる別のデータ・ストリーム。 *AFPDS には、テキスト、イメージ、およびグラフィックスが含まれます。また、*AFPDS では、 ページ・セグメント内でページ・オーバーレイや外部イメージなどの外部リソースを使用することができます。
- *USERASCII - サーバーが単に通過させることによって処理する非 SCS および非 AFPDS プリンター・データ。 *USERASCII スプール・ファイルに入れられるデータ・ストリームの例としては、 ポストスクリプトや HP-PCL データ・ストリームがあります。

例

スプール・ファイルの作成の例

SCS スプール・ファイルの作成の例

SCS データ・ストリームの生成

サーバーに接続された特定のプリンターで印刷されるスプール・ファイルを生成するには、 SNA 文字ストリーム (SCS) データ・ストリームを作成することが必要な場合があります。 (SCS はテキスト・ベースの EBCDIC データ・ストリームであり、 SCS プリンター、IPDS プリンター、あるいは PC プリンターで印刷することができます。) SCS は、エミュレーターまたはサーバー上のホスト印刷変換機能を用いて変換することによって、 印刷することができます。

そのような SCS データ・ストリームを作成するには、 SCS 書き込み機能クラスを使用することができます。 SCS 書き込み機能クラスは、 Java Unicode 文字とフォーマット・オプションを SCS データ・ストリームに変換します。 5 つの SCS 書き込み機能クラスにより、 さまざまなレベルの SCS データ・ストリームが生成されます。 呼び出し元は、 呼び出し元またはエンド・ユーザーが印刷を行う最終的なプリント出力先に適した書き込み機能を選択する必要があります。

SCS プリンター・データ・ストリームを生成するには、 以下の SCS 書き込み機能クラスを使用します。

SCS 書き込み機能 クラス	説明
SCS5256Writer	最も単純な SCS 書き込み機能クラス。 テキスト、復帰、改行、新規行、改ページ、水平方向および垂直方向の絶対位置決め、 水平方向および垂直方向の相対位置決め、そして垂直方向形式の設定をサポートします。
SCS5224Writer	5256 書き込み機能を拡張し、 1 インチ当たりの文字数 (CPI) と 1 インチ当たりの行数 (LPI) を設定するためのメソッドを追加します。
SCS5219Writer	5224 書き込み機能を拡張して、左マージン、下線、用紙タイプ (紙または封筒)、 用紙サイズ、印刷品質、コード・ページ、文字セット、給紙用の用紙入れ番号、 および出力用の用紙入れ番号のサポートを追加します。
SCS5553Writer	5219 書き込み可能を拡張し、文字回転、けい線、およびフォント・スケーリングのサポートを追加します。 5553 は、2 バイト文字セット (DBCS) データ・ストリームです。
SCS3812Writer	5219 書き込み機能を拡張し、太字、両面印刷、 テキスト方位、およびフォントのサポートを追加します。

SCS 書き込み機能を構成する際、呼び出し元では出力ストリームと、 エンコード (任意) が必要になります。 データ・ストリームは、この出力ストリームに書き込まれます。 SCS スプール・ファイルを作成するために、 呼び出し元は、まず SpooledFileOutputStream を構成し、 次にそれを使用して SCS 書き込み機能オブジェクトを構成します。 エンコード・パラメーターは、 文字を変換する宛先 EBCDIC コード化文字セット識別子 (CCSID) を提供します。

書き込み機能を構成したら、 write() メソッドを使用してテキストを出力してください。 ページ上で書き込みカーソルを配置するには、 carriageReturn()、lineFeed()、および newLine() メソッドを使用します。 現在のページを終了して新しいページを開始するには、 endPage() メソッドを使用します。

すべてのデータが書き込まれたら、 close() メソッドを使用してデータ・ストリームを終了し、 出力ストリームをクローズしてください。

スプール・ファイルと AFP リソースの読み取り

PrintObjectInputStream クラスを使用すると、 サーバーからスプール・ファイルまたは高機能印刷 (AFP) リソースの生の内容を読み取ることができます。 このクラスは、標準 JDK java.io.InputStream クラスを拡張したものであるため、InputStream が使用されている任意の箇所で使用することができます。

PrintObjectInputStream オブジェクトを取得するには、 SpooledFile クラスのインスタンスに対して getInputStream() メソッドを呼び出すか、 あるいは AFPResource クラスのインスタンスに対して getInputStream() メソッドを呼び出します。 スプール・ファイルの入力ストリームの取得は、OS/400 のバージョン 3 リリース 2 (V3R2)、 V3R7、およびそれ以降のバージョンでサポートされます。 AFP リソースの入力ストリームの取得は、V3R7 以降でサポートされます。

入力ストリームからの読み取りには、いずれかの read() メソッドを使用します。これらのメソッドはすべて、実際に読み取られたバイトの数、 あるいは -1 (バイトを読み取ることなく、ファイルの終わりに達した場合) を戻します。

スプール・ファイルまたは AFP リソースの合計バイト数を戻すには、PrintObjectInputStream の available() メソッドを使用します。
PrintObjectInputStream クラスでは入力ストリームへのマーキングがサポートされるため、 PrintObjectInputStream は markSupported() メソッドから常に TRUE を戻します。呼び出し元では、 mark() および reset() メソッドを使用して、 後続の読み取りが同じバイトを再度読み取れるように、 入力ストリーム内の現在の読み取り位置を後方へ移動することができます。 データを読み取らずに、入力ストリーム内の読み取り位置を前方へ移動するには、 skip() メソッドを使用してください。

例

スプール・ファイルの読み取りの例

PrintObjectPageInputStream および PrintObjectTransformedInputStream を使用してスプール・ファイルを読み取る

PrintObjectPageInputStream クラスを使用すれば、 サーバー AFP および SCS スプール・ファイルからデータを 1 ページずつ読み取ることができます。

PrintObjectPageInputStream オブジェクトは、 getPageInputStream() メソッドを使用して取得できます。

入力ストリームからの読み取りには、いずれかの read() メソッドを使用します。 これらのメソッドはすべて、実際に読み取られたバイトの数、 あるいは -1 (バイトを読み取ることなく、ページの終わりに達した場合) を戻します。

現行ページの合計バイト数を戻すには、PrintObjectPageInputStream の available() メソッドを使用します。 PrintObjectPageInputStream クラスでは入力ストリームへのマーキングがサポートされるため、 PrintObjectPageInputStream は markSupported() メソッドから常に TRUE を戻します。 呼び出し元では、 mark() および reset() メソッドを使用して、 後続の読み取りが同じバイトを再度読み取れるように、 入力ストリーム内の現在の読み取り位置を後方へ移動することができます。 呼び出し元は、 skip() メソッドを使用して、 データを読み取らずに、入力ストリーム内の読み取り位置を前方へ移動することができます。

しかし、スプール・ファイル・データ・ストリームの全体を変換したい場合は、 PrintObjectTransformedInputStream クラスを使用します。

プロダクト・ライセンス

ProductLicense クラスを使用すると、iSeries にインストールされているプロダクトのライセンスを要求できます。 他の iSeries ライセンス・ユーザーとの互換性を得るために、 このクラスはライセンスの要求または解放時に、iSeries プロダクト・ライセンス・サポートを使用して処理を行います。

クラスがライセンス・ポリシーを強制することはありませんが、 アプリケーションがポリシーを強制できるように十分な情報を送信します。 ライセンスが要求されると、 ProductLicense クラスは、要求の状況、 つまりライセンスが認可されたか拒否されたかを戻します。 要求が拒否された場合、 アプリケーションは、ライセンスを必要とした動作を無効にする必要があります。 これは、IBM Toolbox for Java がどの機能を無効にするべきかを認識しないからです。

iSeries ライセンス・サポートを持つ ProductLicense クラスを使用して、 アプリケーションのライセンスを強制します。

- アプリケーションのサーバー側は、プロダクトおよびライセンス条項を iSeries ライセンス・サポートに登録します。
- アプリケーションのクライアント側は、ProductLicense オブジェクトを使用して、 ライセンスを要求および解放します。

例: ProductLicense のシナリオ

たとえば、自分の顧客が、プロダクト用に 15 個のコンカレント使用のライセンスを購入したとします。 コンカレント使用とは、15 人のユーザーが同時にプロダクトを使用できるということですが、 特定の 15 人である必要はなく、組織内のいかなる 15 人であってもよいわけです。 この情報は、iSeries ライセンス・サポートに登録されています。 ユーザーが接続すると、アプリケーションは ProductLicense クラスを使用してライセンスを要求します。

- 並行ユーザーの数が 15 人よりも少ない場合、 要求は成功し、アプリケーションは実行します。
- 16 人目のユーザーが接続すると、ProductLicense 要求は失敗します。 そして、アプリケーションがエラー・メッセージを表示して終了します。

ユーザーがアプリケーションの実行を停止すると、 アプリケーションは ProductLicense クラスを介してライセンスを解放します。 これで、他の人がライセンスを使用することができます。

詳細およびコード例については、ProductLicense javadoc を参照してください。

ProgramCall クラス

ProgramCall クラスによって、 Java プログラムは iSeries プログラムを呼び出すことができます。 ProgramParameter クラスを使用して、 入力、出力、および入出力パラメーターを指定することができます。 プログラムを実行すると、 出力および入出力パラメーターには iSeries プログラムから戻されたデータが入ります。 iSeries プログラムの実行が失敗した場合、Java プログラムでは、 結果の iSeries メッセージを AS400Message オブジェクトのリストとして取り出すことができます。

必須パラメーターは、次のとおりです。

- 実行するプログラムとパラメーター
- プログラムを所有する iSeries システムを表す AS400 オブジェクト

プログラム名およびパラメーター・リストは、 setProgram() メソッドを介してコンストラクターで、 あるいは run() メソッドで設定することができます。 run() メソッドはプログラムを呼び出します。

ProgramCall オブジェクト・クラスは、AS400 オブジェクトを iSeries に接続させます。

以下の例では、ProgramCall クラスを使用する方法を示します。

```
                    // Create an AS400 object.
AS400 sys = new AS400("mySystem.myCompany.com");

                    // Create a program object. I choose
                    // to set the program to run later.
ProgramCall pgm = new ProgramCall(sys);

                    // Set the name of the program.
                    // Because the program does not take
                    // any parameters, pass null for the
                    // ProgramParameter[] argument.
pgm.setProgram(QSYSObjectPathName.toPath("MYLIB",
                                         "MYPROG",
                                         "PGM"));

                    // Run the program. My program has
                    // no parms. If it fails to run, the failure
                    // is returned as a set of messages
                    // in the message list.
```

```
if (pgm.run() != true)
{
                    // If you get here, the program
                    // failed to run. Get the list of
                    // messages to determine why the
                    // program didn't run.
   AS400Message[] messageList = pgm.getMessageList();

                    // ... Process the message list.
}

                    // Disconnect since I am done
                    // running programs
sys.disconnectService(AS400.COMMAND);
```

ProgramCall オブジェクトは、 プログラムの統合ファイル・システム・パス名を必要とします。

ProgramCall クラスを使用すると、AS400 オブジェクトが iSeries に接続されます。 接続の管理については、接続の管理を参照してください。

デフォルトの動作は、Java プログラムと iSeries プログラムが同一サーバー上にあるときでも、 iSeries プログラムが個別のサーバー・ジョブで実行するというものです。 しかし、このデフォルトの動作をオーバーライドし、 setThreadSafe() メソッドを使用して iSeries プログラムが Java のジョブで実行されるようにすることができます。

ProgramParameter オブジェクトを使用する

ProgramParameter オブジェクトを使用すると、 Java プログラムと iSeries プログラムの間でパラメーター・データを受け渡しすることができます。 setInputData() メソッドを使用して入力データを設定します。 プログラムの実行後に、 getOutputData() メソッドで出力データを検索します。 各パラメーターはバイト配列です。 Java プログラムは、バイト配列を Java 形式と iSeries 形式との間で変換する必要があります。 データ変換クラスは、データを変換するためのメソッドを提供します。 パラメーターは、リストとして ProgramCall オブジェクトに追加されます。

以下の例では、ProgramParameter オブジェクトを使用してパラメーター・データを渡す方法を示します。

```
                    // Create an AS400 object
AS400 sys = new AS400("mySystem.myCompany.com");
```

```
                    // My program has two parameters.
                    // Create a list to hold these
                    // parameters.
  ProgramParameter[] parmList = new ProgramParameter[2];

                    // First parameter is an input
                    // parameter
  byte[] key = {1, 2, 3};
  parmList[0] = new ProgramParameter(key);

                    // Second parameter is an output
                    // parameter. A four-byte number
                    // is returned.
  parmList[1] = new ProgramParameter(4);

                    // Create a program object
                    // specifying the name of the
                    // program and the parameter list.
  ProgramCall pgm = new ProgramCall(sys,
                                    "/QSYS.LIB/MYLIB.LIB/MYPROG.PGM",
                                    parmList);

                    // Run the program.
  if (pgm.run() != true)
  {

                    // If the iSeries cannot run the
                    // program, look at the message list
                    // to find out why it didn't run.
     AS400Message[] messageList = pgm.getMessageList();

  }
else
  {
                    // Else the program ran. Process the
                    // second parameter, which contains
                    // the returned data.

                    // Create a converter for this
                    // iSeries data type
     AS400Bin4 bin4Converter = new AS400Bin4();
```

```
                    // Convert from iSeries type to Java
                    // object. The number starts at the
                    // beginning of the buffer.
    byte[] data = parmList[1].getOutputData();
    int i = bin4Converter.toInt(data);
}

                    // Disconnect since I am done
                    // running programs
sys.disconnectService(AS400.COMMAND);
```

QSYSObjectPathName クラス

QSYSObjectPathName クラスを使用すると、 統合ファイル・システム内のオブジェクトを表現することができます。 このクラスは、統合ファイル・システム名を組み立てたり、統合ファイル・システム名をその構成要素に解析したりするために使用します。

いくつかの IBM Toolbox for Java クラスを使用するためには、統合ファイル・システム・パス名が必要です。 統合ファイル・システム・パス名を組み立てるには、 QSYSObjectPathName オブジェクトを使用します。

以下の例では、QSYSObjectPathName クラスを使用する方法を示します。

例 1: ProgramCall オブジェクトは、サーバー・プログラムを呼び出すために統合ファイル・システム名を必要とします。 統合ファイル・システム名を組み立てるには、 QSYSObjectPathName オブジェクトを使用します。 QSYSObjectPathName を使用してライブラリー REPORTS 内のプログラム PRINT_IT を呼び出す場合:

```
                    // Create an AS400 object.
AS400 sys = new AS400("mySystem.myCompany.com");

                    // Create a program call object.
ProgramCall pgm = new ProgramCall(sys);

                    // Create a path name object that
                    // represents program PRINT_IT in
                    // library REPORTS.
QSYSObjectPathName pgmName = new QSYSObjectPathName("REPORTS",
                                                    "PRINT_IT",
                                                    "PGM");

                    // Use the path name object to set
                    // the name on the program call
                    // object.
pgm.setProgram(pgmName.getPath());

                    // ... run the program, process the
                    // results
```

例 2: AS400 オブジェクトの名前が一度だけ使用される場合、Java プログラムでは、 toPath() メソッドを使用して名前を組み立てることができます。 このメ

ソッドのほうが、QSYSObjectPathName オブジェクトを作成するより効率的です。

```
                    // Create an AS400 object.
AS400 sys = new AS400("mySystem.myCompany.com");

                    // Create a program call object.
ProgramCall pgm = new ProgramCall(sys);

                    // Use the toPath method to create
                    // the name that represents program
                    // PRINT_IT in library REPORTS.
pgm.setProgram(QSYSObjectPathName.toPath("REPORTS",
                                         "PRINT_IT",
                                         "PGM"));

                    // ... run the program, process the
                    // results
```

例 3:この例では、Java プログラムに統合ファイル・システム・パスが与えられています。 QSYSObjectPathName クラスを使用して、この名前を構成要素に解析することができます。

```
                    // Create a path name object from
                    // the fully qualified integrated
                    // file system name.
QSYSObjectPathName ifsName = new QSYSObjectPathName(pathName);

                    // Use the path name object to get
                    // the library, name and type of
                    // server object.
String library = ifsName.getLibraryName();
String name    = ifsName.getObjectName();
String type    = ifsName.getObjectType();
```

レコード・レベルでのアクセス

レコード・レベルのアクセス・クラスには、以下のことを行う機能があります。

- 次のいずれかを指定して iSeries 物理ファイルを作成する。
  - レコード長
  - 既存のデータ記述仕様 (DDS) ソース・ファイル
  - RecordFormat オブジェクト
- iSeries 物理または論理ファイルからレコード様式を取り出すか、 または iSeries 複数様式論理ファイルからレコード様式を取り出す。

  注: ファイルのレコード様式は、完全な形では取り出されません。 取り出されるレコード様式は、 AS400File オブジェクトのレコード様式の設定時に使用されるように意図されています。 ファイルのレコードの内容を記述するのに十分な情報だけが取り出されます。 列見出しや別名などのレコード様式情報は取り出されません。
- レコード番号またはキーによって、iSeries ファイル内のレコードに順次にアクセスする。
- iSeries ファイルにレコードを書き込む。
- レコード番号またはキーによって、iSeries ファイル内のレコードに順次に更新する。
- レコード番号またはキーによって、iSeries ファイル内のレコードに順次に削除する。
- さまざまなタイプのアクセスのために iSeries ファイルをロックする。
- Java プログラムが以下のことを行えるように、コミットメント制御を使用する。
  - 接続についてのコミットメント制御の開始
  - ファイルごとに異なるコミットメント制御ロック・レベルの指定
  - トランザクションのコミットおよびロールバック
- iSeries ファイルを削除する。
- iSeries ファイルからメンバーを削除する。

注: レコード・レベルのアクセス・クラスは、 論理結合ファイルまたはヌル・キー・フィールドをサポートしません。

以下のクラスが実行する機能は、次のとおりです。

- AS400File クラスは、レコード・レベルのアクセス・クラス用の抽象基本クラスです。 このクラスは、順次レコード・アクセス、ファイルとメン

バーの作成および削除、 コミットメント制御アクティビティーのためのメソッドを提供します。
- KeyedFile クラスは、 キーによってアクセスされる iSeries ファイルを表します。
- SequentialFile クラスは、 レコード番号によってアクセスされる iSeries ファイルを表します。
- AS400FileRecordDescription クラスは、 iSeries ファイルのレコード様式を取り出すためのメソッドを提供します。

レコード・レベルのアクセス・クラスには、 データベース・ファイルが含まれているシステムを示す AS400 オブジェクトが必要です。 レコード・レベルのアクセス・クラスを使用することにより、AS400 オブジェクトが iSeries に接続されます。 接続の管理については、接続の管理を参照してください。

レコード・レベルのアクセス・クラスには、データベース・ファイルの統合ファイル・システム・パス名が必要です。 詳細については、統合ファイル・システム・パス名を参照してください。

レコード・レベルのアクセス・クラスは、以下のものを使用します。
- データベース・ファイルのレコードを記述する RecordFormat クラス
- データベース・ファイルのレコードへのアクセスを提供する Record クラス
- レコードを行データ形式で書き込む LineDataRecordWriter クラス

これらのクラスについては、データ変換のセクションで説明します。

例

- 順次アクセスの例では、 iSeries ファイルに順次にアクセスする方法を示します。
- ファイル読み取りの例では、 レコード・レベルのアクセス・クラスを使用して iSeries ファイルを読み取る方法を示します。
- キー付きファイルの例では、 レコード・レベルのアクセス・クラスを使用して iSeries ファイルからキーによってレコードを読み取る方法を示します。

AS400File

AS400File クラスは、 次のことを行うためのメソッドを提供します。

- サーバー物理ファイルとメンバーの作成および削除
- サーバー・ファイル内のレコードの読み取りおよび書き込み
- 各種のアクセスに対するファイルのロック
- パフォーマンスを向上させるためのレコード・ブロック化の使用
- オープンしたサーバー・ファイル内でのカーソル位置の設定
- コミットメント制御アクティビティーの管理

KeyedFile

KeyedFile クラスを使用すると、 Java プログラムでサーバー上のファイルへのキー順アクセスを行うことができます。 キー順アクセスとは、 Java プログラムでキーを指定することによってファイルのレコードにアクセスできることを意味します。 キーによってカーソルの位置付け、レコードの読み取り、更新、 および削除を行うためのメソッドが存在します。

カーソルを位置付けるには、以下のメソッドを使用してください。

- positionCursor(Object[]) - 指定されたキーを持つ最初のレコードにカーソルを設定します。
- positionCursorAfter(Object[]) - 指定されたキーを持つ最初のレコードの次のレコードにカーソルを設定します。
- positionCursorBefore(Object[]) - 指定されたキーを持つ最初のレコードの前のレコードにカーソルを設定します。

レコードを削除するには、次のメソッドを使用してください。

- deleteRecord(Object[]) - 指定されたキーを持つ最初のレコードを削除します。

読み取りメソッドは、次のとおりです。

- read(Object[]) - 指定されたキーを持つ最初のレコードを読み取ります。
- readAfter(Object[]) - 指定されたキーを持つ最初のレコードの次のレコードを読み取ります。
- readBefore(Object[]) - 指定されたキーを持つ最初のレコードの前のレコードを読み取ります。
- readNextEqual() - キーが指定のキーと一致する次のレコードを読み取ります。 検索は、現行カーソル位置の後ろから始まります。
- readPreviousEqual() - キーが指定のキーと一致する前のレコードを読み取ります。 検索は、現行カーソル位置の前から始まります。

レコードを更新するには、次のメソッドを使用してください。

- update(Object[]) - 指定されたキーを持つレコードを更新します。

キーによる位置付け、読み取り、 および更新時の検索基準を指定するためのメソッドも用意されています。 有効な検索基準値は、以下のとおりです。

- 等しい (Equal) - キーが指定のキーと一致する最初のレコードを検出します。
- より小 (Less than) - キーがファイルのキー順で指定のキーの前にくる最後のレコードを検出します。
- より小か等しい (Less than or equal) - キーが指定のキーと一致する最初のレコードを検出します。 指定されたキーと一致するレコードがない場合は、 キーがファイルのキー順で指定のキーの前にくる最後のレコードを検出します。
- より大 (Greater than) - キーがファイルのキー順で指定のキーの後にくる最初のレコードを検出します。
- より大か等しい (Greater than or equal) - キーが指定のキーと一致する最初のレコードを検出します。 指定されたキーと一致するレコードがない場合は、 キーがファイルのキー順で指定のキーの後にくる最初のレコードを検出します。

KeyedFile は、AS400File のサブクラスです。 つまり、AS400File 内のすべてのメソッドが KeyedFile で使用可能です。

キーの指定

KeyedFile オブジェクトのキーは、タイプと順序が、 ファイルの RecordFormat オブジェクトによって指定されたキー・フィールドのタイプと順序に対応する Java オブジェクトの配列によって表現されます。

以下の例では、KeyedFile オブジェクトのキーを指定する方法を示します。

```
            // Specify the key for a file whose key fields, in order,
            // are:
            //    CUSTNAME    CHAR(10)
            //    CUSTNUM     BINARY(9)
            //    CUSTADDR    CHAR(100)VARLEN()
            // Note that the last field is a variable-length field.
    Object[] theKey = new Object[3];
    theKey[0] = "John Doe";
    theKey[1] = new Integer(445123);
    theKey[2] = "2227 John Doe Lane, ANYTOWN, NY 11199";
```

KeyedFile オブジェクトは、完全キーと同様に部分キーを受け入れます。 ただし、指定されるキー・フィールド値は正しい順序になっていなければなりません。

たとえば、次のようにします。

```
            // Specify a partial key for a file whose key fields,
            // in order, are:
            //    CUSTNAME    CHAR(10)
            //    CUSTNUM     BINARY(9)
            //    CUSTADDR    CHAR(100)VARLEN()
    Object[] partialKey = new Object[2];
    partialKey[0] = "John Doe";
    partialKey[1] = new Integer(445123);

            // Example of an INVALID partial key
    Object[] INVALIDPartialKey = new Object[2];
    INVALIDPartialKey[0] = new Integer(445123);
    INVALIDPartialKey[1] = "2227 John Doe Lane, ANYTOWN, NY 11199";
```

ヌル・キーとヌル・キー・フィールドは、サポートされていません。

レコードのキー・フィールドの値は、 getKeyFields() メソッドによって、 ファイルの Record オブジェクトから取得することができます。

以下の例では、キーによってファイルから読み取る方法を示します。

```
            // Create an AS400 object, the file exists on this
            // server.
    AS400 sys = new AS400("mySystem.myCompany.com");

            // Create a file object that represents the file
    KeyedFile myFile = new KeyedFile(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE/%FILE%.MBR");
```

```
        // Assume that the AS400FileRecordDescription class
        // was used to generate the code for a subclass of
        // RecordFormat that represents the record format
        // of file MYFILE in library MYLIB.  The code was
        // compiled and is available for use by the Java program.
RecordFormat recordFormat = new MYKEYEDFILEFormat();

        // Set the record format for myFile.  This must
        // be done prior to invoking open()
myFile.setRecordFormat(recordFormat);

        // Open the file.
myFile.open(AS400File.READ_WRITE, 0, AS400File.COMMIT_LOCK_LEVEL_NONE);

        // The record format for the file contains
        // four key fields, CUSTNUM, CUSTNAME, PARTNUM
        // and ORDNUM in that order.
        // The partialKey will contain 2 key field
        // values.  Because the key field values must be
        // in order, the partialKey will consist of values for
        // CUSTNUM and CUSTNAME.
Object[] partialKey = new Object[2];
partialKey[0] = new Integer(1);
partialKey[1] = "John Doe";

        // Read the first record matching partialKey
Record keyedRecord = myFile.read(partialKey);

        // If the record was not found, null is returned.
if (keyedRecord != null)
{ // Found the record for John Doe, print out the info.
  System.out.println("Information for customer " + (String)partialKey[1] + ":");
  System.out.println(keyedRecord);
}

                ....

        // Close the file since I am done using it
myFile.close();

        // Disconnect since I am done using record-level access
sys.disconnectService(AS400.RECORDACCESS);
```

SequentialFile

SequentialFile クラスを使用すると、 Java プログラムでレコード番号によってサーバー上のファイルにアクセスすることができます。 レコード番号によってカーソルの位置付け、レコードの読み取り、更新、 および削除を行うためのメソッドが存在します。

カーソルを位置付けるには、以下のメソッドを使用してください。

- positionCursor(int) - 指定されたレコード番号を持つレコードにカーソルを設定します。
- positionCursorAfter(int) - 指定されたレコード番号の次のレコードにカーソルを設定します。
- positionCursorBefore(int) - 指定されたレコード番号の前のレコードにカーソルを設定します。

レコードを削除するには、次のメソッドを使用してください。

- deleteRecord(int) - 指定されたレコード番号を持つレコードを削除します。

レコードを読み取るには、以下のメソッドを使用してください。

- read(int) - 指定されたレコード番号を持つレコードを読み取ります。
- readAfter(int) - 指定されたレコード番号の次のレコードを読み取ります。
- readBefore(int) - 指定されたレコード番号の前のレコードを読み取ります。

レコードを更新するには、次のメソッドを使用してください。

- update(int) - 指定されたレコード番号を持つレコードを更新します。

SequentialFile は、AS400File のサブクラスです。 つまり、AS400File 内のすべてのメソッドが SequentialFile で使用可能です。

以下の例では、SequentialFile クラスを使用する方法を示します。

```
                    // Create an AS400 object, the file exists on this
                    // server.
AS400 sys = new AS400("mySystem.myCompany.com");

                    // Create a file object that represents the file
SequentialFile myFile = new SequentialFile(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE/%FILE%.MBR");

                    // Assume that the AS400FileRecordDescription class
                    // was used to generate the code for a subclass of
                    // RecordFormat that represents the record format
                    // of file MYFILE in library MYLIB.  The code was
                    // compiled and is available for use by the Java program.
RecordFormat recordFormat = new MYFILEFormat();

                    // Set the record format for myFile.  This must
                    // be done prior to invoking open()
myFile.setRecordFormat(recordFormat);

                    // Open the file.
myFile.open(AS400File.READ_WRITE, 0, AS400File.COMMIT_LOCK_LEVEL_NONE);

                    // Delete record number 2.
myFile.delete(2);

                    // Read record number 5 and update it
Record updateRec = myFile.read(5);
updateRec.setField("CUSTNAME", newName);

                    // Use the base class' update() method since I am
                    // already positioned on the record.
```

```
myFile.update(updateRec);

                  // Update record number 7
updateRec.setField("CUSTNAME", nextNewName);
updateRec.setField("CUSTNUM", new Integer(7));
myFile.update(7, updateRec);

                ....

                  // Close the file since I am done using it
myFile.close();

                  // Disconnect since I am done using record-level access
sys.disconnectService(AS400.RECORDACCESS);
```

AS400FileRecordDescription

AS400FileRecordDescription クラスは、 サーバー上のファイルのレコード様式を取り出すためのメソッドを提供します。 このクラスは、RecordFormat のサブクラス用の Java ソース・コードを作成するためのメソッドと、 ユーザー指定の物理ファイルまたはサーバー上の論理ファイルのレコード様式を記述する RecordFormat オブジェクトを戻すためのメソッドを提供します。 これらのメソッドの出力は、レコード様式の設定時に AS400File オブジェクトへの入力として使用できます。

サーバーにファイルがすでに存在する場合には、 RecordFormat オブジェクトを生成するために必ず AS400FileRecordDescription クラスを使用することをお勧めします。

注: AS400FileRecordDescription クラスでは、 ファイルのレコード様式全体は取り出されません。 ファイルを構成するレコードの内容を記述するのに十分な情報だけが取り出されます。 列見出し、別名、および参照フィールドなどの情報は取り出されません。 したがって、取り出されたレコード様式は、 その取り出し元のファイルとレコード様式が一致するファイルを作成するために必ずしも使用できるとは限りません。

サーバー上のファイルのレコード様式を表す RecordFormat のサブクラス用の Java ソース・コードの作成

createRecordFormatSource() メソッドは、 RecordFormat クラスのサブクラス用の Java ソース・ファイルを作成します。 ファイルは、コンパイルして、 AS400File.setRecordFormat() メソッドへの入力として、 アプリケーションまたはアプレットで使用できます。

createRecordFormatSource() メソッドは、開発時ツールとして、 既存のサーバー上のファイルのレコード様式を取り出すために使用します。 このメソッドを使用すると、RecordFormat クラスのサブクラス用のソースを一度作成し、 必要に応じて変更を加え、コンパイルして、 サーバー上の同じファイルにアクセスする多数の Java プログラムで使用することができます。 このメソッドはローカル・システム上でファイルを作成するため、 Java アプリケーションによってのみ使用できます。 ただし、出力 (Java ソース・コード) は、コンパイルした後、 Java アプリケーションでもアプレットでも同様に使用できます。

注: このメソッドは、作成される Java ソース・ファイルと同じ名前のファイルをオーバーライドします。

例 1: 以下の例では、createRecordFormatSource() メソッドを使用する方法を示します。

```
                    // Create an AS400 object, the file exists on this
                    // server.
    AS400 sys = new AS400("mySystem.myCompany.com");

                    // Create an AS400FileRecordDescription object that represents the file
    AS400FileRecordDescription myFile = new AS400FileRecordDescription(sys,
"/QSYS.LIB/MYLIB.LIB/MYFILE.FILE");

                    // Create the Java source file in the current working directory.
                    // Specify "package com.myCompany.myProduct;" for the
                    // package statement in the source since I will ship the class
                    // as part of my product.
    myFile.createRecordFormatSource(null, "com.myCompany.myProduct");

                    // Assuming that the format name for file MYFILE is FILE1, the
                    // file FILE1Format.java will be created in the current working directory.
                    // It will overwrite any file by the same name.  The name of the class
                    // will be FILE1Format.  The class will extend from RecordFormat.
```

例 2: 上で作成したファイル FILE1Format.java をコンパイルし、 次のように使用します。

```
                    // Create an AS400 object, the file exists on this
                    // server.
    AS400 sys = new AS400("mySystem.myCompany.com");
```

```
                    // Create an AS400File object that represents the file
    SequentialFile myFile = new SequentialFile(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE");

                    // Set the record format
                    // This assumes that import.com.myCompany.myProduct.FILE1Format;
                    // has been done.

    myFile.setRecordFormat(new FILE1Format());

                    // Open the file and read from it
                    ....

                    // Close the file since I am done using it
    myFile.close();

                    // Disconnect since I am done using record-level access
    sys.disconnectService(AS400.RECORDACCESS);
```

サーバー上のファイルのレコード様式を表す RecordFormat オブジェクトの作成

retrieveRecordFormat() メソッドは、 サーバー上の既存のファイルのレコード様式を表す RecordFormat オブジェクトの配列を戻します。 通常、配列では 1 つの RecordFormat オブジェクトだけが戻されます。 レコード様式が取り出されるファイルが複数様式論理ファイルである場合は、 複数の RecordFormat オブジェクトが戻されます。 このメソッドは、実行時にサーバー上の既存のファイルのレコード様式を動的に取り出すために使用します。 その後、RecordFormat オブジェクトは、 AS400File.setRecordFormat() メソッドへの入力として使用できます。

以下の例では、retrieveRecordFormat() メソッドを使用する方法を示します。

```
                    // Create an AS400 object, the file exists on this
                    // server.
    AS400 sys = new AS400("mySystem.myCompany.com");

                    // Create an AS400FileRecordDescription object that represents the file
    AS400FileRecordDescription myFile = new AS400FileRecordDescription(sys,
"/QSYS.LIB/MYLIB.LIB/MYFILE.FILE");

                    // Retrieve the record format for the file
    RecordFormat[] format = myFile.retrieveRecordFormat();

                    // Create an AS400File object that represents the file
    SequentialFile myFile = new SequentialFile(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE");

                    // Set the record format
    myFile.setRecordFormat(format[0]);

                    // Open the file and read from it
                    ....

                    // Close the file since I am done using it
    myFile.close();

                    // Disconnect since I am done using record-level access
    sys.disconnectService(AS400.RECORDACCESS);
```

サービス・プログラム呼び出し

ServiceProgramCall クラスを使用すると、 iSeries サービス・プログラムを呼び出すことができます。 ServiceProgramCall は、 iSeries プログラムを呼び出すのに使用する ProgramCall のサブクラスです。 iSeries プログラムを呼び出す場合は、ProgramCall クラスを使用します。

ServiceProgramCall クラスを使用すると、 iSeries サービス・プログラムを呼び出し、入力パラメーターによって iSeries サービス・プログラムにデータを渡し、 iSeries サービス・プログラムが出力パラメーターによって戻すデータにアクセスすることができます。 ServiceProgramCall を使用すると、iSeries オブジェクトが iSeries に接続されます。 接続の管理については、接続の管理を参照してください。

デフォルトの動作は、 Java プログラムとサービス・プログラムが同じサーバー上にあるときでも、 サービス・プログラムが個別のサーバー・ジョブで実行するというものです。 デフォルトの動作を変更して、 サービス・プログラムが、(ProgramCall から) 継承した setThreadSafe() メソッドを使用して、 Java ジョブで実行されるようにすることができます。

ServiceProgramCall クラスを使用する

ServiceProgramCall クラスを使用するためには、以下の要件が満たされるようにしなければなりません。

- サービス・プログラムは、OS/400 V4R4 以上を実行している iSeries または AS/400e 上になければならない
- サービス・プログラムには 7 つまでしかパラメーターを渡すことができない
- サービス・プログラムの戻り値はボイドまたは数値である

ProgramParameter オブジェクトを使用して作業する

ProgramParameter クラスは、 ServiceProgramCall クラスを使用して、iSeries サービス・プログラムとの間でパラメーター・データをやり取りします。 入力データは、setInputData() を使用して iSeries サービス・プログラムに渡します。

戻される出力データの量は、 setOutputDataLength() を使用して要求します。 サービス・プログラムの実行が終了した後で出力データを取り出すには、getOutputData() を使用します。 ServiceProgramCall は、データそのものに加えて、 パラメーター・データをサービス・プログラムに渡す方法を認識している必要があります。 この情報を提供するには、 ProgramParameter の

setParameterType() メソッドを使用します。 このタイプは、パラメーターが値によって渡されるのか、参照によって渡されるのかを示します。 どちらの場合でも、データはクライアントからサーバーに送信されます。 データが iSeries に渡されると、サーバーはこのパラメーター・タイプを使用して、 サービス・プログラムを正確に呼び出します。

すべてのパラメーターは、バイト配列の形式になります。 したがって、iSeries 形式と Java 形式の間で変換を行うには、 データ変換および記述クラスを使用します。

SystemStatus クラス

SystemStatus クラスを使用すると、 システム状況の検索と、システム・プール情報の検索および変更を行うことができます。 SystemStatus オブジェクトでは、以下のものを含むシステム状況情報を取り出すことができます。

- getUsersCurrentSignedOn(): システムに現在サインオンしているユーザーの数を戻します。
- getUsersTemporarilySignedOff(): 切断されている対話式ジョブの数を戻します。
- getDateAndTimeStatusGathered(): システム状況情報が収集された日時を戻します。
- getJobsInSystem(): 現在実行中のユーザーおよびシステム・ジョブの合計数を戻します。
- getBatchJobsRunning(): システムで現在実行中のバッチ・ジョブの数を戻します。
- getBatchJobsEnding(): 終了処理中のバッチ・ジョブの数を戻します。
- getSystemPools(): 各システム・プールの SystemPool オブジェクトを含むリストを戻します。

SystemStatus を使用して、SystemStatus クラス内のメソッドのほかに、SystemPool にアクセスすることもできます。 SystemPool を使用すると、システム・プールに関する情報を取得および変更することができます。

例

この例では、SystemStatus クラスと一緒にキャッシュを使用する方法を示します。

```
AS400 system = new AS400("MyAS400");
SystemStatus status = new SystemStatus(system);

// Turn on caching. It is off by default.
status.setCaching(true);

// This will retrieve the value from the system.
// Every subsequent call will use the cached value
// instead of retrieving it from the system.
int jobs = status.getJobsInSystem();
```

```
// ... Perform other operations here ...


// This determines if caching is still enabled.
if (status.isCaching())
{
  // This will retrieve the value from the cache.
jobs = status.getJobsInSystem();
}

// Go to the system next time, regardless if caching is enabled.
status.refreshCache();

// This will retrieve the value from the system.
jobs = status.getJobsInSystem();

// Turn off caching. Every subsequent call will go to the system.
status.setCaching(false);

// This will retrieve the value from the system.
jobs = status.getJobsInSystem();
```

システム値

システム値クラスを使用すると、 Java プログラムでは、システム値とネットワーク属性の検索および変更を行うことができます。 さらに、必要なシステム値を含む、独自のグループを定義することもできます。

SystemValue オブジェクトには、主として、次の情報が含まれます。

- 名前
- 記述
- リリース
- 値

SystemValue クラスでは、単一のシステム値の検索には getValue() メソッドを使 用し、 システム値の変更には setValue() メソッドを使用します。

さらに、特定のシステム値に関するグループ情報を取り出すこともできます。

- システム値が属するシステム定義グループを検索するには、 getGroup() メソッドを使用します。
- SystemValue オブジェクトが属するユーザー定義グループ (もしあれば) を取り出すには、 getGroupName() および getGroupDescription() メソッドを使用します。

あるシステム値の値が初めて検索されるときには、必ず、 iSeries から値が検索され、キャッシュに入れられます。 後続の検索では、キャッシュに入れられた値が戻されます。 キャッシュに入れられた値ではなく、iSeries の現行値が必要な場合は、 clear() を実行して、 現行のキャッシュをクリアしなければなりません。

システム値リスト

SystemValueList は、 指定された iSeries システム上のシステム値のリストを表します。 リストは、いくつかのシステム定義グループに分割され、 これにより、Java プログラムでは、一度に 1 つのシステム値の部分にアクセスすることができます。

システム値グループ

SystemValueGroup は、 システム値およびネットワーク属性のユーザー定義コレクションを表します。 これは、コンテナーというよりは、むしろ、 システム値の固有のコレクションを生成および保守するためのファクトリーです。

SystemValueGroup を作成するには、 システム定義グループの 1 つ (SystemValueList クラスの定数の 1 つ) を指定するか、 またはシステム値名の配列を指定します。

add() メソッドを使用すると、 システム値の名前を個別に追加して、グループに組み込むことができます。 また、remove() メソッドを使用して、 それを除去することもできます。

SystemValueGroup に必要なシステム値名を挿入した後で、 グループから実際の SystemValue オブジェクトを取得するには、 getSystemValues() メソッドを呼び出します。 このように、SystemValueGroup オブジェクトは、システム値名のセットを受け取り、 SystemValue オブジェクト (すべて SystemValueGroup のシステム、グループ名、 およびグループ記述を持つ) のベクトルを生成します。

あるベクトルの SystemValue オブジェクトをすべて一度に最新表示するには、 refresh() メソッドを使用します。

SystemValue および SystemValueList クラスの使用例

以下の例では、システム値を作成し、取り出す方法を示します。

```
//Create an AS400 object
AS400 sys = new AS400("mySystem.myCompany.com");

//Create a system value representing the current second on the system.
```

```
SystemValue sysval = new SystemValue(sys, "QSECOND");

//Retrieve the value.
String second = (String)sysval.getValue();

//At this point QSECOND is cached. Clear the cache to retrieve the most
//up-to-date value from the system.
sysval.clear();
second = (String)sysval.getValue();

//Create a system value list.
SystemValueList list = new SystemValueList(sys);

//Retrieve all the of the date/time system values.
Vector vec = list.getGroup(SystemValueList.GROUP_DATTIM);

//Disconnect from the system.
sys.disconnectAllServices();
```

SystemValueGroup クラスの使用例

以下の例では、 システム値名のグループを作成し、システム値名を操作する方法を示します。

```
//Create an AS400 object
AS400 sys = new AS400("mySystem.myCompany.com");

//Create a system value group initially representing all of the network attributes on the system.
String name = "My Group";
String description = "This is one of my system values.";
SystemValueGroup svGroup = new SystemValueGroup(sys, name, description, SystemValueList.GROUP_NET);

//Add some more system value names to the group and remove some we do not want.
svGroup.add("QDATE");
svGroup.add("QTIME");
svGroup.remove("NETSERVER");
svGroup.remove("SYSNAME");

//Obtain the actual SystemValue objects. They are returned inside a Vector.
Vector sysvals = svGroup.getSystemValues();

//You will notice that this is one of my system values.
SystemValue mySystemValue = (SystemValue)sysvals.elementAt(0);
System.out.println(mySystemValue.getName()+" - "+mySystemValue.getGroupDescription());

//We can add another SystemValue object from another system into the group.
AS400 sys2 = new AS400("otherSystem.myCompany.com");
SystemValue sv = new SystemValue(sys2, "QDATE");
sysvals.addElement(sv);

//Now refresh the entire group of system values all at once.
//It does not matter if some system values are from different iSeries systems.
//It does not matter if some system values were generated using SystemValueGroup and some were not.
SystemValueGroup.refresh(sysvals);

//Disconnect from the systems.
sys.disconnectAllServices();
sys2.disconnectAllServices();
```

トレース

トレース・オブジェクトを使用すると、 Java プログラムでは、トレース・ポイントおよび診断メッセージをログに記録することができます。 この情報は、問題を再現して診断する際に役立ちます。

注: さらに、 トレース・システムのプロパティーを使用してトレースを設定できます。

トレース・クラスでは、以下のカテゴリーの情報がログに記録されます。

情報カテゴリー	説明
変換	Unicode とコード・ページの間の文字セット変換をログに記録します。 このカテゴリーは、IBM Toolbox for Java クラスでのみ使用されます。
データ・ストリーム	iSeries と Java プログラムの間でやり取りされるデータをログに記録します。 このカテゴリーは、IBM Toolbox for Java クラスでのみ使用されます。
診断	状態情報をログに記録します。
エラー	例外を引き起こす追加のエラーをログに記録します。
通知	プログラムのフローをトレースします。
»PCML	このカテゴリーは、サーバーとの間でやり取りされるデータを PCML が解釈する方法を決定するために使用されます。 «
Proxy	このカテゴリーは、クライアントと Proxy サーバーの間でやり取りされるデータをログに記録するために、 IBM Toolbox for Java クラスによって使用されます。
警告	プログラムが回復できたエラーについての情報をログに記録します。
すべて	このカテゴリーは、 上記のすべてのカテゴリーのトレースを一度に使用可能または使用不可にするために使用されます。 トレース情報をこのカテゴリーに直接記録することはできません。

IBM Toolbox for Java クラスでも、トレース・カテゴリーが使用されます。 Java プログラムでロギングを使用可能にすると、 アプリケーションによって記録される情報とともに、 IBM Toolbox for Java の情報が組み込まれます。

トレースは、 単一のカテゴリーで使用するようにも、 複数のカテゴリーで使用するようにも設定できます。 カテゴリーを選択した後、トレースのオン/オフを切り替えるには、 setTraceOn メソッドを使用します。 データは、log メソッドを使用して、 ログに書き込まれます。

異なるコンポーネントのトレース・データを、別々のログに送ることができます。

デフォルトでは、トレース・データはデフォルトのログに書き込まれます。 アプリケーション固有のトレース・データを別個のログまたは標準出力に書き込むには、 コンポーネント・トレースを使用します。 コンポーネント・トレースを使用すると、 特定のアプリケーションのトレース・データを他のデータから簡単に分離することができます。

過度なロギングは、パフォーマンスに影響を与える可能性があります。 トレースの現在の状態を調べるには、 isTraceOn メソッドを使用します。ご使用の Java プログラムでこのメソッドを使用し、 ログ・メソッドを呼び出す前にトレース・レコードを作成するかどうかを判別することができます。 ロギングがオフのときにログ・メソッドを呼び出すのはエラーにはなりませんが、 時間がかかります。

デフォルトは、ログ情報を標準出力に書き出すことです。 ログをファイルに宛先変更するには、 Java アプリケーションから setFileName() メソッドを呼び出します。 一般に、ほとんどのブラウザーは、 ローカル・ファイル・システムへの書き込みアクセスをアプレットに付与しないため、 これが有効なのは Java アプリケーションだけです。

ロギングは、デフォルトではオフです。 ユーザーがロギングを簡単に使用可能にできるように、 Java プログラムでは、ロギングをオンにするための手段を提供するべきです。 たとえば、アプリケーションで、 ログに記録されるデータのカテゴリーを示すコマンド行パラメーターの解析を行うことができます。 ユーザーは、ログ情報が必要なときにこのパラメーターを設定することができます。

以下の例では、トレース・クラスを使用する方法を示します。

例 1: この例では、setTraceOn method を使用する方法と、 ログ・メソッドを使用してログにデータを書き込む方法を示します。

```
// Enable diagnostic, information, and warning logging.
Trace.setTraceDiagnosticOn(true);
Trace.setTraceInformationOn(true);
Trace.setTraceWarningOn(true);

// Turn tracing on.
Trace.setTraceOn(true);

// ... At this point in the Java program, write to the log.
Trace.log(Trace.INFORMATION, "Just entered class xxx, method xxx");

// Turning tracing off.
Trace.setTraceOn(false);
```

例 2: この例では、トレースを使用する方法を示します。トレースを使用するコードを書き込む場合、 望ましいのは Method 2 です。

```
// Method 1 - build a trace record
// then call the log method and let the trace class determine if the
// data should be logged. This will work but will be slower than the
// following code.
String traceData = new String("Just entered class xxx, data = ");
traceData = traceData + data + "state = " + state;
Trace.log(Trace.INFORMATION, traceData);

// Method 2 - check the log status before building the information to
// log. This is faster when tracing is not active.
if (Trace.isTraceOn() && Trace.isTraceInformationOn())
{
   String traceData = new String("just entered class xxx, data = ");
   traceData = traceData + data + "state = " + state;
   Trace.log(Trace.INFORMATION, traceData);
}
```

例 3: この例では、 コンポーネント・トレースを使用する方法を示します。

```
// Create a component string. It is more efficient to create an
// object than many String literals.
String myComponent1 = "com.myCompany.xyzComponent";
String myComponent2 = "com.myCompany.abcComponent";

// Send Toolbox and the component trace data each to separate files.
// The Toolbox trace will contain all trace information, while each
// component log file will only contain trace information specific to
// that component.  If a Trace file is not specified, all trace data
// will go to standard out with the component specified in front of
// each trace message.

// Trace.setFileName("c://bit.bucket");
// Trace.setFileName(myComponent1, "c://Component1.log");
// Trace.setFileName(myComponent2, "c://Component2.log");

Trace.setTraceOn(true);              // Turn trace on.
Trace.setTraceInformationOn(true);  // Enable information messages.

// Log component specific trace data or general toolbox
// trace data.

Trace.setFileName("c://bit.bucket");
Trace.setFileName(myComponent1, "c://Component1.log");
```

トレース・ファイルを指定しない場合、例の結果は次のようになります。

```
Toolbox for Java - Version 5 Release 1 Modification level 0
[com.myCompany.xyzComponent] Tue Oct 24 16:02:44 CDT 2000 I am here
[com.myCompany.abcComponent] Tue Oct 24 16:02:44 CDT 2000 I am there
Tue Oct 24 16:02:44 CDT 2000 I am everywhere
```

ユーザーおよびグループ

ユーザーおよびグループのクラスを使用すれば、 Java プログラムを使って、 iSeries システム上のユーザーとグループのリスト、 および各ユーザーについての情報を取得できます。

注: Toolbox for Java は、 さまざまな iSeries オブジェクトおよびリストを扱うための汎用フレームワークおよび一貫性のあるプログラミング・インターフェースを提供するリソース・クラスも備えています。 アクセス・パッケージおよびリソース・パッケージのクラスに関する記述を読んでから、 ご使用のアプリケーションにとって最善のオブジェクトを選択できます。 ユーザーを扱うリソース・クラスは、 RUser および RUserList です。

検索できるユーザー情報としては、 前回のサインオン日付、状況、パスワードが最後に変更された日付、パスワード失効日付、 およびユーザー・クラスなどがあります。 User オブジェクトにアクセスする場合、 setSystem() メソッドを使用してシステム名を設定し、 setName() メソッドを使用してユーザー名を設定します。 これらのステップが済んだら、 loadUserInformation() メソッドを使って iSeries から情報を取得します。

UserGroup オブジェクトは、 ユーザー・プロファイルがグループ・プロファイルである特殊ユーザーを表します。 getMembers() メソッドを使用すると、グループのメンバーであるユーザーのリストを戻すことができます。

Java プログラムは、この列挙を使用してリストを繰り返すことができます。 列挙内の要素はすべて、 User オブジェクトです。 以下に例を示します。

```
// Create an AS400 object.
AS400 system = new AS400 ("mySystem.myCompany.com");

// Create the UserList object.
UserList userList = new UserList (system);

// Get the list of all users and groups.
Enumeration enum = userList.getUsers ();

// Iterate through the list.
while (enum.hasMoreElements ())
{
    User u = (User) enum.nextElement ();
    System.out.println  (u);
}
```

ユーザーおよびグループに関する情報の検索

UserList を使って、以下の情報に関するリストを取得します。

- すべてのユーザーおよびグループ
- グループのみ
- グループのメンバーであるすべてのユーザー
- グループのメンバーでないすべてのユーザー

UserList オブジェクトで必ず設定しなければならない唯一のプロパティーは、ユーザー・リストが検索されるシステムを表す AS400 オブジェクトです。

デフォルトでは、すべてのユーザーが戻されます。 setUserInfo() と setGroupInfo() を組み合わせて使用すれば、 どのユーザーを戻すべきかを正確に指定できます。

例: UserList を使用して特定のグループ内のすべてのユーザーをリストする。

UserSpace クラス

UserSpace クラスは、サーバー上のユーザー・スペースを表します。 必須パラメーターは、ユーザー・スペースの名前、 およびそのユーザー・スペースがあるサーバーを表す AS400 オブジェクトです。 ユーザー・スペース・クラスのメソッドは、以下のことを行います。

- ユーザー・スペースを作成します。
- ユーザー・スペースを削除します。
- ユーザー・スペースから読み取りを行います。
- ユーザー・スペースに書き込みを行います。
- ユーザー・スペースの属性を取得します。 Java プログラムでは、ユーザー・スペースの初期値、長さ値、 および自動拡張可能属性を取得できます。
- ユーザー・スペースの属性を設定します。 Java プログラムでは、ユーザー・スペースの初期値、長さ値、 および自動拡張可能属性を取得できます。

UserSpace オブジェクトには、 プログラムの統合ファイル・システム・パス名が必要です。 詳細については、統合ファイル・システム・パス名を参照してください。

UserSpace クラスを使用すると、AS400 オブジェクトはサーバーに接続します。 接続の管理については、接続の管理を参照してください。

以下の例では、ユーザー・スペースを作成し、そのユーザー・スペースにデータを書き込みます。

```
                    // Create an AS400 object.
AS400 sys = new AS400("mySystem.myCompany.com");

                    // Create a user space object.
UserSpace US = new UserSpace(sys,
          "/QSYS.LIB/MYLIB.LIB/MYSPACE.USRSPC");

                    // Use the create method to create the user space on
                    // the server.
US.create(10240,                          // The initial size is 10K
          true,                           // Replace if the user space already exists
          " ",                            // No extended attribute
          (byte) 0x00,                    // The initial value is a null
          "Created by a Java program",    // The description of the user space
          "*USE");                        // Public has use authority to the user space


                    // Use the write method to write bytes to the user space.
US.write("Write this string to the user space.", 0);
```

HTML クラス

IBM Toolbox for Java HTML クラスは、以下の事柄に役立ちます。

- HTML ページの形式およびテーブルのセットアップ
- テキストの位置合わせ
- さまざまな HTML タグの処理
- 言語およびテキストの方向の変更
- 順序リストおよび順不同リストの作成
- ファイル・リストおよび HTML 階層ツリー (およびその中の要素) の作成
- HTML クラスで未定義のタグ属性 (たとえば、bgcolor やスタイル属性など) の追加

HTML クラスは、HTMLTagElement インターフェースを実装します。 各クラスは特定の要素タイプの HTML タグを生成します。 タグは getTag() メソッドを使用して検索でき、 どんな HTML 文書にも組み込めます。 HTML クラスで生成したタグは、 HTML 3.2 仕様に準拠しています。

HTML クラスを servlet クラスとともに使用すれば、 iSeries サーバーからのデータを取得できます。 ただし、テーブルまたはフォームのデータを提供すれば、 単独で使用することもできます。

以下の HTML クラスは、 HTML フォーム、テーブル、および他の要素の作成を容易にします。

- BidiOrdering クラスによって、 言語およびテキストの方向を変更できます。
- DirFilter クラスによって、File オブジェクトがディレクトリーであるかどうかを判別できます。
- HTMLAlign クラスによって、 HTML 出力のブロックを位置合わせできます。
- HTMLFileFilter クラスによって、File オブジェクトがファイルであるかどうかを判別できます。
- HTMLForm クラスを使うと、 CGI スクリプトよりも簡単にフォームを作成できます。
- HTMLHeading クラスによって、 HTML ページの見出しタグを作成できます。
- HTMLHyperlink クラスを使うと、 HTML ページ内にリンクを作成できます。

- » HTMLImage クラスによって、 HTML ページのイメージ・タグを作成できます。 «
- HTMLList クラスは、 HTML ページのリストを作成するために役立ちます。
- HTMLMeta クラスによって、 HTML ページのメタ・タグを作成できます。
- HTMLParameter クラスは、 HTMLServlet が使用可能なパラメーターを指定します。
- HTMLServlet クラスによって、 サーバー・サイド・インクルードを作成できます。
- HTMLTable クラスを使うと、 HTML ページ内のテーブルを容易に作成できるようになります。
- HTMLText クラスによって、 HTML ページのフォント・プロパティーにアクセスできます。
- HTMLTree クラスによって、 HTML 要素の HTML 階層ツリーを表示できます。
- URLEncoder クラスは、URL ストリングで使用する区切り文字をエンコードします。
- URLParser クラスによって、 URL ストリングを構文解析して、URI、プロパティー、および参照を調べることができます。

注: jt400Servlet.jar ファイルには、 HTML クラスと servlet クラスの両方が入っています。 com.ibm.as400.util.html パッケージでこれらのクラスを使用したい場合には、 CLASSPATH を更新して jt400Servlet.jar ファイルを指定する必要があります。

BidiOrdering クラス

BidiOrdering クラスは、 言語およびテキストの方向を変更する HTML タグを表します。 HTML <BDO> ストリングには、言語用とテキストの方向用の 2 つの属性が必要です。

BidiOrdering クラスを使用して、以下を行うことができます。

- 言語属性を取得したり設定する
- テキストの方向を取得したり設定する

<BDO> HTML タグの使用方法についての詳細は、W3C の Web サイトを参照してください。

例: BidiOrdering を使用する

以下の例では、BidiOrdering オブジェクトを作成してその言語および方法を設定します。

```
// Create a BidiOrdering object and set the language and direction.
BidiOrdering bdo = new BidiOrdering();
bdo.setDirection(HTMLConstants.RTL);
bdo.setLanguage("AR");


// Create some text.
HTMLText text = new HTMLText("Some Arabic Text.");
text.setBold(true);

// Add the text to the BidiOrdering and get the HTML tag.
bdo.addItem(text);
bdo.getTag();
```

印刷ステートメントでは、以下のタグが作成されます。

```
<bdo lang="AR" dir="rtl">
  <b>Some Arabic Text.</b>
</bdo>
```

HTML ページでこのタグを使用すると、 <BDO> タグに対応しているブラウザーではこの例が以下のように表示されます。

.txeT cibarA emoS

HTMLAlign クラス

HTMLAlign クラスを使用して、 段落や見出しなどの項目を個別に位置合わせする代わりに、 HTML 文書の複数のセクションを位置合わせすることができます。

HTMLAlign クラスは、<DIV> タグおよびそれに関連した位置合わせ属性を表しています。 位置合わせとして、右そろえ、左そろえ、および中央そろえを使用できます。

このクラスを使用して、以下を含むさまざまなアクションを実行することができます。

- 位置合わせしたいタグのリストに項目を追加したり、 リストから除去したりする。
- 位置合わせを取得したり、設定したりする。
- テキスト変換処理の方向を取得したり、 設定したりする。
- 入力要素の言語を取得したり、 設定したりする。
- HTMLAlign オブジェクトのストリング表記を取得 (Get a String representation) する。

例: HTMLAlign オブジェクトを作成する

以下の例では、リストを順不同で作成してから、 HTMLAlign オブジェクトを作成してリスト全体を位置合わせします。

```
// Create an unordered list.
UnorderedList uList = new UnorderedList();
uList.setType(HTMLConstants.DISC);
UnorderedListItem uListItem1 = new UnorderedListItem();
uListItem1.setItemData(new HTMLText("Centered unordered list"));
uList.addListItem(uListItem1);
UnorderedListItem uListItem2 = new UnorderedListItem();
uListItem2.setItemData(new HTMLText("Another item"));
uList.addListItem(uListItem2);

// Align the list.
HTMLAlign align = new HTMLAlign(uList, HTMLConstants.CENTER);
System.out.println(align);
```

上記の例では、以下のタグが生成されます。

```
<div align="center">
<ul type="disc">
  <li>Centered unordered list</li>
  <li>Another item</li>
</ul>
```

HTML ページでこのタグを使用すると以下のように表示されます。

<ul type="disc">
  Centered unordered list
  Another item

HTML フォーム・クラス

HTMLForm クラスは HTML フォームを表します。 このクラスでは、以下のことを行えます。

- フォームにボタン、ハイパーリンク、または HTML テーブルなどの要素を追加する
- フォームから要素を除去する
- 他のフォーム属性 (たとえば、フォームの内容をサーバー、 隠しパラメーター・リスト、 またはアクション URL アドレスに送るメソッドなど) を設定する。

HTMLForm オブジェクトのコンストラクターは、URL アドレスを受け取ります。 このアドレスはアクション URL と呼ばれています。 これは、フォーム入力を処理するサーバー上のアプリケーションの位置を表します。 アクション URL はコンストラクター上で指定するか、 setURL() メソッドを設定することによって指定できます。 フォーム属性は、さまざまな set メソッドを使用して設定し、 さまざまな get メソッドを使用して検索することができます。

どのような HTML タグも、 addElement() を使用して HTMLForm オブジェクトに追加し、 removeElement() を使用して除去することができます。 HTMLForms では、以下の HTML タグ要素クラスを使用してください。

- FormInput classes: HTML フォームの入力要素を表す
- LayoutFormPanel クラス: HTML フォームのフォーム要素のレイアウトを表す
- TextAreaFormElement: HTML フォームのテキスト域要素を表す
- LabelFormElement: HTML フォーム要素のラベルを表す
- SelectFormElement: HTML フォームの選択入力タイプを表す
- SelectOption: HTML フォーム内の SelectFormElement オブジェクトのオプションを表す
- RadioFormInputGroup: ラジオ入力オブジェクトのグループを表す。 ユーザーはそのグループから 1 つのオブジェクトを選択できます。

もちろん、以下のものも含め、他のタグ要素もフォームに追加できます。

- HTMLText
- HTMLHyperlink
- HTMLTable

HTMLForm クラスの使用についての詳細は、この例と、 その結果生成された

出力を参照してください。

FormInput クラス

FormInput クラスを使用して、以下を行うことができます。

- 入力要素の名前を取得したり、設定する。
- 入力要素のサイズを取得したり、 設定する。
- 入力要素の初期値を取得したり、 設定する。

FormInput クラスは、以下のリストのクラスによって拡張されます。 これらのクラスにより、特定のタイプのフォーム入力要素を作成したり、 さまざまな属性を取得および設定したり、入力要素の HTML タグを検索したりできます。

- ButtonFormInput: HTML フォームのボタン要素を表す
- FileFormInput: HTML フォームのファイル入力タイプを表す
- HiddenFormInput: HTML フォームの隠し入力タイプを表す
- ImageFormInput: HTML フォームのイメージ入力タイプを表す
- ResetFormInput: HTML フォームのリセット・ボタン入力を表す
- SubmitFormInput: HTML フォームの実行依頼ボタン入力を表す
- TextFormInput: HTML フォームの単一行のテキスト入力を表す。 行ごとの最大文字数を定義できます。 パスワード入力タイプでは、PasswordFormInput を使用します。 これは TextFormInput を拡張するもので、 HTML フォームのパスワード入力タイプを表します。
- ToggleFormInput: HTML フォームのトグル入力タイプを表す。 ユーザーは、テキスト・ラベルを設定または取得でき、 トグルをチェックするタイプにするか、または選択するタイプにするかを指定できます。 トグル入力タイプには、以下の 2 つのタイプの中の 1 つを使用できます。
  - RadioFormInput: HTML フォームのラジオ・ボタン入力タイプを表す。 ラジオ・ボタンは、 RadioFormInputGroup クラスによりグループとして置くことができます。 これにより、ラジオ・ボタンのグループが作成され、 ユーザーは表示されている選択項目の中から 1 つのボタンだけを選択します。
  - CheckboxFormInput: HTML フォームのチェック・ボックス入力タイプを表す。 ユーザーは表示される選択項目から複数のものを選択できます。 チェック・ボックスはチェックされた状態またはチェックされていない状態のいずれかで初期化されます。

ButtonFormInput クラス

ButtonFormInput クラスは、HTML フォームのボタン要素を表します。

以下の例は、ButtonFormInput オブジェクトを作成する方法を示しています。

```
ButtonFormInput button = new ButtonFormInput("button1", "Press Me", "test()");
System.out.println(button.getTag());
```

この例では、以下のようなタグが作成されます。

```
<input type="button" name="button1" value="Press Me" onclick="test()" />
```

FileFormInput クラス

FileFormInput クラスは、 HTML フォームのファイル入力タイプを表します。

以下のコーディング例では、 新しい FileFormInput オブジェクトの作成方法を示します。

```
FileFormInput file = new FileFormInput("myFile");
System.out.println(file.getTag());
```

上記のコードは、次のような出力を出します。

```
<input type="file" name="myFile" />
```

HiddenFormInput クラス

HiddenFormInput クラスは、 HTML フォームの隠し入力タイプを表します。

以下のコーディング例では、 HiddenFormInput オブジェクトの作成方法を示します。

```
HiddenFormInput hidden = new HiddenFormInput("account", "123456");
System.out.println(hidden.getTag()):
```

上記のコードは、以下のタグを生成します。

<input type="hidden" name="account" value="123456" />

HTML ページ内では、HiddenInputType は表示されません。 情報 (この例ではアカウント番号) がサーバーに送り返されるだけです。

ImageFormInput クラス

ImageFormInput クラスは、 HTML フォームのイメージ入力タイプを表します。

ImageFormInput クラスの属性の多くは、以下のメソッドを使用して検索および更新できます。

- ソースを取得または設定する
- 位置合わせを取得または設定する
- 高さを取得または設定する
- 幅を取得または設定する

以下のコード例では、 ImageFormInput オブジェクトの作成方法を示します。

```
ImageFormInput image = new ImageFormInput("myPicture", "myPicture.gif");
image.setAlignment(HTMLConstants.TOP);
image.setHeight(81);
image.setWidth(100);
```

上記のコード例は、以下のタグを生成します。

```
<input type="image" name="MyPicture" src="myPicture.gif" align="top" height="81" width="100" />
```

ResetFormInput クラス

ResetFormInput クラスは、 HTML フォームにおけるリセット・ボタンの入力タイプを表します。

以下のコード例は、ResetFormInput オブジェクトの作成方法を示します。

```
ResetFormInput reset = new ResetFormInput();
reset.setValue("Reset");
System.out.println(reset.getTag());
```

上記のコード例は、以下の HTML タグを生成します。

```
<input type="reset" value="Reset" />
```

SubmitFormInput クラス

SubmitFormInput クラスは、 HTML フォームにおける実行依頼ボタンの入力タイプを表します。

以下のコード例では、SubmitFormInput オブジェクトの作成方法を示します。

```
SubmitFormInput submit = new SubmitFormInput();
submit.setValue("Send");
System.out.println(submit.getTag());
```

上記のコード例は、以下の出力を生成します。

```
<input type="submit" value="Send" />
```

TextFormInput クラス

TextFormInput クラスは、HTML フォームでの単一行テキスト入力タイプを表します。 TextFormInput クラスが提供するメソッドを使用すると、 テキスト・フィールドでユーザーが入力できる文字の最大数を取得したり設定することができます。

以下の例は、新しい TextFormInput オブジェクトを作成する方法を示します。

```
TextFormInput text = new TextFormInput("userID");
text.setSize(40);
System.out.println(text.getTag());
```

上記のコード例は、以下のタグを生成します。

```
<input type="text" name="userID" size="40" />
```

PasswordFormInput クラス

PasswordFormInput クラスは、 HTML フォームにおけるパスワード入力フィールドのタイプを表します。

以下のコード例では、新しい PasswordFormInput オブジェクトの作成方法を示します。

```
PasswordFormInput pwd = new PasswordFormInput("password");
pwd.setSize(12);
System.out.println(pwd.getTag());
```

上記のコード例は、以下のタグを生成します。

```
<input type="password" name="password" size="12" />
```

RadioFormInput クラス

RadioFormInput クラスは、 HTML フォームにおけるラジオ・ボタンの入力タイプを表します。 ラジオ・ボタンは、構成時に選択済みとして初期設定することができます。

同じコントロール名を持つラジオ・ボタンのセットは、ラジオ・ボタン・グループを構成します。 RadioFormInputGroup クラスは、 ラジオ・ボタン・グループを作成します。同時に選択可能なラジオ・ボタンは 1 つだけです。 また、グループの構成時に特定のボタンを選択済みとして初期設定することもできます。

以下のコード例は、RadioFormInput オブジェクトの作成方法を示します。

```
RadioFormInput radio = new RadioFormInput("age", "twentysomething", "Age 20 - 29", true);
System.out.println(radio.getTag());
```

上記のコード例は、以下のタグを生成します。

```
<input type="radio" name="age" value="twentysomething" checked="checked" />
```

CheckboxFormInput クラス

CheckboxFormInput クラスは、HTML フォーム内のチェック・ボックス入力タイプを表します。ユーザーは、フォーム内でチェック・ボックスとして表示されている選択項目から複数個選択できます。

以下の例は、新しい CheckboxFormInput オブジェクトを作成する方法を示しています。

```
CheckboxFormInput checkbox = new CheckboxFormInput("uscitizen", "yes", "textLabel", true);
System.out.println(checkbox.getTag());
```

上記のコードは、以下のような出力を生成します。

```
<input type="checkbox" name="uscitizen" value="yes" checked="checked" /> textLabel
```

LayoutFormPanel クラス

LayoutFormPanel クラスは、 HTML フォームのフォーム要素のレイアウトを表します。LayoutFormPanel が提供するメソッドを使用すれば、 パネルから要素を追加および除去したり、レイアウトにある要素の数を取得したりできます。以下の 2 つのレイアウトから 1 つを使用できます。

- GridLayoutFormPanel: HTML フォームのフォーム要素の格子レイアウトを表す。
- LineLayoutFormPanel: HTML フォームのフォーム要素のライン・レイアウトを表す。

GridLayoutFormPanel

GridLayoutFormPanel クラスは、 フォーム要素の格子レイアウトを表します。 このレイアウトは、格子の列の数を指定する必要がある HTML フォームで使用します。

以下の例では、2 つの列を持つ GridLayoutFormPanel オブジェクトを作成します。

```
        // Create a text form input element for the system.
  LabelFormElement sysPrompt = new LabelFormElement("System:");
  TextFormInput system = new TextFormInput("System");


        // Create a text form input element for the userId.
  LabelFormElement userPrompt = new LabelFormElement("User:");
  TextFormInput user = new TextFormInput("User");


        // Create a password form input element for the password.
  LabelFormElement passwordPrompt = new LabelFormElement("Password:");
  PasswordFormInput password = new PasswordFormInput("Password");


        // Create the GridLayoutFormPanel object with two columns and add the form elements.
  GridLayoutFormPanel panel = new GridLayoutFormPanel(2);
  panel.addElement(sysPrompt);
  panel.addElement(system);
  panel.addElement(userPrompt);
  panel.addElement(user);
  panel.addElement(passwordPrompt);
  panel.addElement(password);


        // Create the submit button to the form.
  SubmitFormInput logonButton = new SubmitFormInput("logon", "Logon");


        // Create HTMLForm object and add the panel to it.
  HTMLForm form = new HTMLForm(servletURI);
  form.addElement(panel);
  form.addElement(logonButton);
```

この例では、以下の HTML コードが作成されます。

```
<form action=servletURI method="get">
<table border="0">
<tr>
<td>System:</td>
<td><input type="text" name="System" /></td>
</tr>
<tr>
<td>User:</td>
```

```
<td><input type="text" name="User" /></td>
</tr>
<tr>
<td>Password:</td>
<td><input type="password" name="Password" /></td>
</tr>
</table>
<input type="submit" name="logon" value="Logon" />
</form>
```

LineLayoutFormPanel クラス

LineLayoutFormPanel クラスは、 HTML フォームのフォーム要素のライン・レイアウトを表します。フォーム要素はパネル内に単一行で配列されます。

この例は LineLayoutFormPanel オブジェクトを作成して、 2 つのフォーム要素を追加します。

```
    CheckboxFormInput privacyCheckbox = new CheckboxFormInput("confidential", "yes", "Confidential",
true);
    CheckboxFormInput mailCheckbox = new CheckboxFormInput("mailingList", "yes", "Join our mailing
list", false);
    LineLayoutFormPanel panel = new LineLayoutFormPanel();
    panel.addElement(privacyCheckbox);
    panel.addElement(mailCheckbox);
    String tag = panel.getTag();
```

上記のコード例は、以下の HTML コードを生成します。

```
    <input type="checkbox" name="confidential" value="yes"
    checked="checked" /> Confidential <input type="checkbox"
    name="mailingList" value="yes" /> Join our mailing list <br/>
```

TextAreaFormElement クラス

TextAreaFormElement クラスは、HTML フォームでのテキスト・エリア要素を表します。 行および列の数を設定することにより、テキスト・エリアのサイズを判別します。 getRows() および getColumns() メソッドを使用すれば、 テキスト・エリア要素が設定されているサイズを判別することができます。

テキスト・エリア内に最初に表示されるテキストは、 setText() メソッドを使用して設定します。 getText() メソッドを使用すれば、 最初のテキストがどのように設定されているかを調べることができます。

以下の例では、TextAreaFormElement を作成する方法を示します。

```
TextAreaFormElement textArea = new TextAreaFormElement("foo", 3, 40);
textArea.setText("Default TEXTAREA value goes here");
System.out.println(textArea.getTag());
```

上記のコード例は、以下の HTML コードを生成します。

```
<form>
<textarea name="foo" rows="3" cols="40">
Default TEXTAREA value goes here
</textarea>
</form>
```

LabelFormElement クラス

LabelFormElement クラスは、HTML フォーム要素のラベルを表します。 LabelFormElement クラスを使用すれば、 テキスト域またはパスワード・フォーム入力などの HTML フォーム要素をラベル付けできます。 ラベルとは、 setLabel() メソッドを使用して設定する 1 行のテキストです。 このテキストはユーザー入力に影響されず、ユーザーがフォームをより容易に見分けるのに役立ちます。

以下のコード例では、 LabelFormElement オブジェクトの作成方法を示します。

```
LabelFormElement label = new LabelFormElement("Account Balance");
System.out.println(label.getTag());
```

この例は、以下のような出力を生成します。

```
Account Balance
```

SelectFormElement クラス

SelectFormElement クラスは、 HTML フォームにおける選択の入力タイプを表します。 選択要素内のさまざまなオプションを追加および除去することができます。

SelectFormElement には、選択要素の属性を表示および変更するために使用できるメソッドがあります。

- ユーザーが複数のオプションを選択できるかどうかを設定するには、setMultiple() を使用します。
- オプション・レイアウトにある要素の数を判別するには、getOptionCount() を使用します。
- 選択要素内で可視となるオプションの数を設定するには、setSize() を使用し、 可視オプションの数を判別するには、getSize() を使用します。

以下の例では、3 つのオプションを指定して SelectFormElement オブジェクトを作成します。 *list* という名前の SelectFormElement オブジェクトを強調表示しています。 追加されている最初の 2 つのオプションは、オプション・テキスト、名前、および選択属性を指定しています。 追加されている 3 番目のオプションは、SelectOption オブジェクトによって定義されています。

```
SelectFormElement list = new SelectFormElement("list1");
SelectOption option1 = list.addOption("Option1", "opt1");
SelectOption option2 = list.addOption("Option2", "opt2", false);
SelectOption option3 = new SelectOption("Option3", "opt3", true);
list.addOption(option3);
System.out.println(list.getTag());
```

上記のコード例は、以下の HTML コードを生成します。

```
<select name="list1">
<option value="opt1">Option1</option>
<option value="opt2">Option2</option>
<option value="opt3" selected="selected">Option3</option>
</select>
```

SelectOption クラス

SelectOption クラスは、 HTML オプション・フォーム要素内のオプションを表します。 オプション・フォーム要素は、選択フォーム内で使用します。

SelectOption 内で属性を検索および設定するのに使用できるメソッドが用意されています。 たとえば、オプションのデフォルトが選択済みの状態になるかどうかを設定することができます。 また、フォームが実行依頼されたときに使用される入力値を設定することもできます。

以下の例では、選択フォーム内の 3 つの SelectOption オプションを作成します。 以下のそれぞれの SelectOption オブジェクトを強調表示します。 これらは、*option1*、*option2* および *option3* という名前です。 *option3* オブジェクトは、最初から選択された状態になります。

```
SelectFormElement list = new SelectFormElement("list1");
SelectOption option1 = list.addOption("Option1", "opt1");
SelectOption option2 = list.addOption("Option2", "opt2", false);
SelectOption option3 = new SelectOption("Option3", "opt3", true);
list.addOption(option3);
System.out.println(list.getTag());
```

上記のコード例は、以下の HTML タグを生成します。

```
<select name="list1">
<option value="opt1">Option1</option>
<option value="opt2">Option2</option>
<option value="opt3" selected="selected">Option3</option>
</select>
```

RadioFormInputGroup クラス

RadioFormInputGroup クラスは、 RadioFormInput オブジェクトのクラスを表します。 ユーザーは、RadioFormInputGroup から RadioFormInput オブジェクトを 1 つだけ選択できます。

RadioFormInputGroup クラスのメソッドを使用すると、ラジオ・ボタンのグループのさまざまな属性を処理することができます。 これらのメソッドを使用して、以下のことを行うことができます。

- ラジオ・ボタンの追加
- ラジオ・ボタンの除去
- ラジオ・ボタン・グループの名前の取得または設定

以下の例では、ラジオ・ボタン・グループを作成します。

```
// Create some radio buttons.
RadioFormInput radio0 = new RadioFormInput("age", "kid", "0-12", true);
RadioFormInput radio1 = new RadioFormInput("age", "teen", "13-19", false);
RadioFormInput radio2 = new RadioFormInput("age", "twentysomething", "20-29", false);
RadioFormInput radio3 = new RadioFormInput("age", "thirtysomething", "30-39", false);
// Create a radio button group and add the radio buttons.
RadioFormInputGroup ageGroup = new RadioFormInputGroup("age");
ageGroup.add(radio0);
ageGroup.add(radio1);
ageGroup.add(radio2);
ageGroup.add(radio3);
System.out.println(ageGroup.getTag());
```

上記のコード例は、以下の HTML コードを生成します。

```
<input type="radio" name="age" value="kid" checked="checked" /> 0-12
<input type="radio" name="age" value="teen" /> 13-19
<input type="radio" name="age" value="twentysomething" /> 20-29
<input type="radio" name="age" value="thirtysomething" /> 30-39
```

HTMLHeading クラス

HTMLHeading クラスは HTML ヘッダーを表します。 各ヘッダーには、固有の位置合わせ、および 1 (最大フォント、最重要度) から 6 までのレベルを設定できます。

HTMLHeading クラスのメソッドには、次のものが含まれます。

- ヘッダーのテキストを取得したり、設定する。
- ヘッダーのレベルを取得したり、設定する。
- ヘッダーの位置合わせを取得したり、設定する。
- テキスト変換処理の方向を取得したり、 設定したりする。
- 入力要素の言語を取得したり、 設定したりする。
- HTMLHeader オブジェクトのストリング表記を取得 (Get a String representation) する。

例: HTMLHeading オブジェクトを作成する

以下の例では、3 つの HTMLHeading オブジェクトを作成します。

```
// Create and display three HTMLHeading objects.
HTMLHeading h1 = new HTMLHeading(1, "Heading", HTMLConstants.LEFT);
HTMLHeading h2 = new HTMLHeading(2, "Subheading", HTMLConstants.CENTER);
HTMLHeading h3 = new HTMLHeading(3, "Item", HTMLConstants.RIGHT);
System.out.print(h1 + "/r/n" + h2 + "/r/n" + h3);
```

上記の例では、以下のタグが作成されます。

```
<h1 align="left">Heading</h1>
<h2 align="center">Subheading</h2>
<h3 align="right">Item</h3>
```

HTMLHyperlink クラス

HTMLHyperlink クラスは、 HTML ハイパーリンク・タグを表します。 HTMLHyperlink クラスを使えば、 HTML ページ内にリンクを作成できます。 このクラスを使用すれば、 以下のようにハイパーリンクの多くの属性を取得したり設定したりできます。

- リンクの URI を取得または設定する
- リンクのタイトルを取得または設定する
- リンクのターゲット・フレームを取得または設定する

HTMLHyperlink クラスは定義されたプロパティーとともに完全なハイパーリンクを印刷するので、 HTML ページ内でその出力を使用できます。

以下に示すのは、HTMLHyperlink の例です。

```
// Create an HTML hyperlink to the IBM Toolbox for Java home page.
HTMLHyperlink toolbox = new HTMLHyperlink("http://www.ibm.com/as400/toolbox", "IBM Toolbox for Java home page");

// Display the toolbox link tag.
System.out.println(toolbox.toString());
```

上記のコードは、以下のタグを生成します。
IBM Toolbox for Java home page

HTML ページでこのタグを使用すると以下のように表示されます。

IBM Toolbox for Java home page

»HTMLImage クラス

HTMLImage クラスによって、 HTML ページのイメージ・タグを作成できます。 HTMLImage クラスは、以下のものを含め、 イメージ属性を取得および設定するのに使用できるメソッドを提供します。

- イメージの高さを取得または設定する
- イメージの幅を取得または設定する
- イメージの名前を取得または設定する
- イメージの代替テキストを取得または設定する
- イメージの周囲の水平方向スペースを取得または設定する
- イメージの周囲の垂直方向スペースを取得または設定する
- イメージへの絶対または相対参照を取得または設定する
- HTMLImage オブジェクトのストリング表記を取得する

以下の例は、HTMLImage オブジェクトを作成する 1 つの方法を示しています。

```
// Create an HTMLImage.
HTMLImage image = new HTMLImage("http://myWebSite/picture.gif",
                                "Alternate text for this graphic");
image.setHeight(94);
image.setWidth(105);
System.out.println(image);
```

印刷ステートメントでは、以下のような単一行のタグが作成されます。 テキストの折り返しは、表示する目的のためだけです。

```
<img src="http://myWebSite/picture.gif"
     alt="Alternate text for this graphic"
     height="94" width="105" />
```

«

HTMLList クラス

HTMLList クラスによって、HTML ページ内にリストを簡単に作成できます。これらのクラスは、リストとリスト内の項目に関するさまざまな属性を取得および設定するメソッドを提供しています。

特に、親クラス HTMLList は、 可能な限り小さな垂直スペースに項目を表示するコンパクト・リスト (compact list) を生成するメソッドを提供しています。

- HTMLList のメソッドには、 次のものが含まれます。
  - リストを短縮する (Compact)
  - 項目をリストに追加したり、 リストから除去する
  - リストを追加したり、 リストから除去する (リストのネストを可能にする)
- HTMLListItem のメソッドには、 次のものが含まれます。
  - 項目の内容を取得したり、 設定する
  - テキスト変換処理の方向を取得したり、 設定したりする。
  - 入力要素の言語を取得したり、 設定したりする。

HTMLList および HTMLListItem のサブクラスを使用して、以下の HTML リストを作成します。

- OrderedList および OrderedListItem
- UnorderedList および UnorderedListItem

コーディング断片については、以下の例を参照してください。

- 例: 順序リストを作成する
- 例: 順不同リストを作成する
- 例: ネストしたリストを作成する

OrderedList および OrderedListItem

OrderedList クラスと OrderedListItem クラスを使用して、HTML ページ内に順序リストを作成します。

- OrderedList のメソッドには、次のものが含まれます。
  - リストの最初の項目の開始番号を取得したり、 設定する。
  - 項目番号のタイプ (またはスタイル) を取得したり、 設定する。
- OrderedListItem のメソッドには、次のものが含まれます。
  - 項目の数を取得したり、 設定する

    - 項目番号のタイプ (またはスタイル) を取得したり、 設定する

OrderedListItem 内のメソッドを使用すると、 リスト内の特定の項目の番号付けおよびタイプを変更することができます。

順序リストを作成するための例を参照してください。

UnorderedList および UnorderedListItem

UnorderedList クラスと UnorderedListItem クラスを使用して、 HTML ページ内に順不同リストを作成します。

- UnorderedList のメソッドには、次のものが含まれます。
    - 項目のタイプ (またはスタイル) を取得したり、 設定する
- UnorderedListItem のメソッドには、次のものが含まれます。
    - 項目のタイプ (またはスタイル) を取得したり、 設定する

順不同リストを作成するための例を参照してください。

例

以下の例では、HTMLList クラスを使用して 順序リスト、順不同リスト、およびネストしたリストを作成する方法を示します。

例: 順序リストを作成する

以下の例では、順序リストを作成します。

```
     // Create an OrderedList.
OrderedList oList = new OrderedList(HTMLConstants.SMALL_ROMAN);
     // Create the OrderedListItems.
OrderedListItem listItem1 = new OrderedListItem();
OrderedListItem listItem2 = new OrderedListItem();
     // Set the data in the OrderedListItems.
listItem1.setItemData(new HTMLText("First item"));
listItem2.setItemData(new HTMLText("Second item"));
     // Add the list items to the OrderedList.
oList.addListItem(listItem1);
oList.addListItem(listItem2);
System.out.println(oList.getTag());
```

上記の例では、以下のタグが作成されます。

```
<ol type="i">
<li>First item</li>
<li>Second item</li>
</ol>
```

HTML ページでこれらのタグを使用すると、以下のように表示されます。

i. First item
ii. Second item

例: 順不同リストを作成する

以下の例では、順不同リストを作成します。

```
     // Create an UnorderedList.
UnorderedList uList = new UnorderedList(HTMLConstants.SQUARE);
     // Create the UnorderedListItems.
UnorderedListItem listItem1 = new UnorderedListItem();
UnorderedListItem listItem2 = new UnorderedListItem();
     // Set the data in the UnorderedListItems.
listItem1.setItemData(new HTMLText("First item"));
listItem2.setItemData(new HTMLText("Second item"));
     // Add the list items to the UnorderedList.
uList.addListItem(listItem1);
uList.addListItem(listItem2);
System.out.println(uList.getTag());
```

上記の例では、以下のタグが作成されます。

```
<ul type="square">
<li>First item</li>
<li>Second item</li>
</ul>
```

HTML ページでこれらのタグを使用すると、以下のように表示されます。

- First item
- Second item

例: ネストしたリストを作成する

以下の例では、ネストしたリストを作成します。

```
     // Create an UnorderedList.
```

```
UnorderedList uList = new UnorderedList(HTMLConstants.SQUARE);
     // Create and set the data for UnorderedListItems.
UnorderedListItem listItem1 = new UnorderedListItem();
UnorderedListItem listItem2 = new UnorderedListItem();
listItem1.setItemData(new HTMLText("First item"));
listItem2.setItemData(new HTMLText("Second item"));
     // Add the list items to the UnorderedList.
uList.addListItem(listItem1);
uList.addListItem(listItem2);

     // Create an OrderedList.
OrderedList oList = new OrderedList(HTMLConstants.SMALL_ROMAN);
     // Create the OrderedListItems.
OrderedListItem listItem1 = new OrderedListItem();
OrderedListItem listItem2 = new OrderedListItem();
OrderedListItem listItem3 = new OrderedListItem();
     // Set the data in the OrderedListItems.
listItem1.setItemData(new HTMLText("First item"));
listItem2.setItemData(new HTMLText("Second item"));
listItem3.setItemData(new HTMLText("Third item"));
     // Add the list items to the OrderedList.
oList.addListItem(listItem1);
oList.addListItem(listItem2);
     // Add (nest) the unordered list to OrderedListItem2
oList.addList(uList);
     // Add another OrderedListItem to the OrderedList
     // after the nested UnorderedList.
oList.addListItem(listItem3);
System.out.println(oList.getTag());
```

上記の例では、以下のタグが作成されます。

```
<ol type="i">
<li>First item</li>
<li>Second item</li>
<ul type="square">
<li>First item</li>
<li>Second item</li>
</ul>
<li>Third item</li>
</ol>
```

HTMLMeta クラス

HTMLMeta クラスは、 HTMLHead タグ内で使用されるメタ情報を表します。 META タグ内の属性は、HTML 文書内の情報の識別、索引付け、および定義を行うときに使用されます。

META タグの属性には、次のものが含まれます。

- NAME - META タグの内容に関連した名前
- CONTENT - NAME 属性に関連した値
- HTTP-EQUIV - 応答メッセージ・ヘッダーについて HTTP サーバーによって収集された情報
- LANG - 言語
- URL - ユーザーを現行ページから他のページに宛先変更するために使用

たとえば、検索エンジンがページの内容を判別しやすくするために、次の META タグを使用できます。

```
<META name="keywords" lang="en-us" content="games, cards, bridge">
```

HTMLMeta を使用して、あるページから別のページにユーザーを宛先変更することもできます。

HTMLMeta クラスのメソッドには、次のものが含まれます。

- NAME 属性の取得および設定
- CONTENT 属性の取得および設定
- HTTP-EQUIV 属性の 取得および設定
- LANG 属性の取得および設定
- URL 属性の取得および設定

例: META タグを作成する

以下の例では、2 つの META タグを作成します。

```
// Create a META tag to help search engines determine page content.
HTMLMeta meta1 = new HTMLMeta();
meta1.setName("keywords");
meta1.setLang("en-us");
meta1.setContent("games, cards, bridge");
// Create a META tag used by caches to determine when to refresh the page.
HTMLMeta meta2 = new HTMLMeta("Expires", "Mon, 01 Jun 2000 12:00:00 GMT");
System.out.print(meta1 + "/r/n" + meta2);
```

上記の例では、以下のタグが作成されます。

```
<meta name="keywords" content="games, cards, bridge">
<meta http-equiv="Expires" content="Mon, 01 Jun 2000 12:00:00 GMT">
```

HTMLParameter クラス

HTMLParameter クラスは、 HTMLServlet クラスと一緒に使用できるパラメーターを表します。 それぞれのパラメーターには固有の名前と値があります。

HTMLParameter クラスのメソッドには、次のものが含まれます。

- パラメーターの名前の取得および設定
- パラメーターの値の取得および設定

例: HTMLParameter タグを作成する

以下の例では、HTMLParameter タグを作成します。

```
     // Create an HTMLServletParameter.
HTMLParameter parm = new HTMLParameter ("age", "21");
System.out.println(parm);
```

上記の例では、以下のタグが生成されます。

```
<param name="age" value="21">
```

HTMLServlet クラス

HTMLServlet クラスは、 サーバー・サイド・インクルードを表します。 サーブレット・オブジェクトは、サーブレットの名前、およびオプションでそのロケーションを指定します。 ローカル・システム上のデフォルト・ロケーションを使用するように選択することもできます。

HTMLServlet クラスは、サーブレットが使用可能なパラメーターを指定する HTMLParameter クラスと共に機能します。

HTMLServlet クラスのメソッドには、次のものが含まれます。

- HTMLParameter をサーブレット・タグに追加したり、 そこから除去する。
- サーブレットの位置を取得したり、設定する。
- サーブレットの名前を取得したり、設定する。
- サーブレットの代替テキストを取得したり、設定する。

例: HTMLServlet タグを作成する

以下の例では、HTMLServlet タグを作成します。

```
     // Create an HTMLServlet.
HTMLServlet servlet = new HTMLServlet("myServlet", "http://server:port/dir");
     // Create a parameter, then add it to the servlet.
HTMLParameter param = new HTMLParameter("parm1", "value1");
servlet.addParameter(param);
     // Create and add second parameter
HTMLParameter param2 = servlet.add("parm2", "value2");
     // Create the alternate text if the Web server does not support the servlet tag.
servlet.setText("The Web server providing this page does not support the SERVLET tag.")
System.out.println(servlet);
```

上記の例では、以下のタグが作成されます。

```
<servlet name="myServlet" codebase="http://server:port/dir">
<param name="parm1" value="value1">
<param name="parm2" value="value2">
The Web server providing this page does not support the SERVLET tag.
</servlet>
```

HTML Table クラス

HTMLTable クラスを使うと、 HTML ページ内で使用できるテーブルを簡単に設定できます。 このクラスは、 テーブルのさまざまな属性を取得および設定する以下のメソッドを提供しています。

- 枠の幅を取得および設定する
- テーブル内の行数を取得する
- テーブルの最後に列または行を追加する
- 指定した列または行位置から列または行を除去する

HTMLTable クラスは、テーブルの作成が簡単になるよう、他の HTML クラスを使用します。 テーブルの作成に使用される他の HTML クラスには以下のものがあります。

- HTMLTableCell: テーブル・セルを作成する
- HTMLTableRow: テーブル行を作成する
- HTMLTableHeader: テーブル・ヘッダー・セルを作成する
- HTMLTableCaption: テーブル・キャプションを作成する

例

例: HTMLTable クラスを使用する。

HTMLTableCell クラス

HTMLTableCell クラスは、 任意の HTMLTagElement オブジェクトを入力として受け取り、 指定された要素を持つテーブル・セル・タグを作成します。 要素は、コンストラクターで、 あるいは 2 つの setElement() メソッドのいずれかを通じて設定することができます。

HTMLTableCell クラスにより提供されるメソッドを使用すると、 多くのセル属性の検索または更新を行うことができます。 これらのメソッドを使用すれば、以下のアクションを実行できます。

- 行幅の取得または設定
- セル高さの取得または設定
- セル・データで通常の HTML 改行規則が使用されるかどうかの設定

以下の例では、HTMLTableCell オブジェクトを作成し、タグを表示します。

```
//Create an HTMLHyperlink object.
HTMLHyperlink link = new HTMLHyperlink("http://www.ibm.com",
                     "IBM Home Page");
HTMLTableCell cell = new HTMLTableCell(link);
cell.setHorizontalAlignment(HTMLConstants.CENTER);
System.out.println(cell.getTag());
```

上記の getTag() メソッドの出力は、 次のようになります。

```
<td align="center"><a href="http://www.ibm.com">IBM Home Page</a></td>
```

HTMLTableRow クラス

HTMLTableRow クラスは、 テーブル内の行を作成します。 このクラスは、行の属性を取得および設定するための各種のメソッドを提供します。 これらのメソッドを使用すれば、以下のことを実行できます。

- 行の列の追加または除去
- 指定された列索引にある列データの取得
- 指定されたセルを持つ列の列索引の取得
- 行内の列の数の取得
- 横方向および縦方向の位置合わせの設定

以下は、HTMLTableRow の例です。

```
// Create a row and set the alignment.
HTMLTableRow row = new HTMLTableRow();
row.setHorizontalAlignment(HTMLTableRow.CENTER);

// Create and add the column information to the row.
HTMLText account = new HTMLText(customers_[rowIndex].getAccount());
HTMLText name = new HTMLText(customers_[rowIndex].getName());
HTMLText balance = new HTMLText(customers_[rowIndex].getBalance());

row.addColumn(new HTMLTableCell(account));
row.addColumn(new HTMLTableCell(name));
row.addColumn(new HTMLTableCell(balance));

// Add the row to an HTMLTable object (assume that the table already exists).
table.addRow(row);
```

HTMLTableHeader クラス

HTMLTableHeader クラスは、 HTMLTableCell クラスから継承されます。 これは、ヘッダー・セルという特定タイプのセルを作成し、 <td> セルではなく <th> セルを提供します。
HTMLTableCell クラスと同様に、ヘッダー・セルの属性を更新または検索するには、各種のメソッドを呼び出します。

以下は、HTMLTableHeader の例です。

```
// Create the table headers.
HTMLTableHeader account_header = new HTMLTableHeader(new HTMLText("ACCOUNT"));
HTMLTableHeader name_header = new HTMLTableHeader(new HTMLText("NAME"));
HTMLTableHeader balance_header = new HTMLTableHeader();
HTMLText balance = new HTMLText("BALANCE");
balance_header.setElement(balance);

// Add the table headers to an HTMLTable object (assume that the table already exists).
table.addColumnHeader(account_header);
table.addColumnHeader(name_header);
table.addColumnHeader(balance_header);
```

HTMLTableCaption クラス

HTMLTableCaption クラスは、 HTML テーブルのキャプションを作成します。 このクラスは、キャプションの属性を更新および検索するためのメソッドを提供します。 たとえば、setAlignment() メソッドを使用すれば、 キャプションがテーブルのどの部分に位置合わせされるかを指定することができます。 以下は、HTMLTableCaption の例です。

```
// Create a default HTMLTableCaption object and set the caption text.
HTMLTableCaption caption = new HTMLTableCaption();
caption.setElement("Customer Account Balances - January 1, 2000");

// Add the table caption to an HTMLTable object (assume that the table already exists).
table.setCaption(caption);
```

HTML Text クラス

HTMLText クラスを使うと、 HTML ページのテキスト・プロパティーにアクセスできます。 HTMLText クラスを使用すると、 以下のようにテキスト属性の状況を取得、設定、および検査できます。

- フォントのサイズを取得または設定する
- 太字属性をオン (真) またはオフ (偽) に設定したり、 それが既にオンになっているかどうかを判別する
- 下線属性をオン (真) またはオフ (偽) に設定したり、 それが既にオンになっているかどうかを判別する
- テキストの横方向位置合わせを取得または設定する

以下の例では、HTMLText オブジェクトの作成方法を示し、その太字属性をオンにしてからフォント・サイズを 5 にします。

```
HTMLText text = new HTMLText("IBM");
text.setBold(true);
text.setSize(5);
System.out.println(text.getTag());
```

印刷ステートメントでは、以下のタグが作成されます。

```
<font size="5"><b>IBM</b></font>
```

HTML ページでこのタグを使用すると以下のように表示されます。

IBM

HTMLTree クラス

HTMLTree クラスを使用すると、 HTML ページ内で使用できる HTML 要素の階層ツリーを簡単に設定できます。 このクラスは、ツリーのさまざまな属性を取得および設定するメソッドに加えて、 以下の事柄を行えるメソッドを提供しています。

- HTTP サーブレット要求の取得および設定。
- HTMLTreeElement および FileTreeElement のツリーへの追加。
- HTMLTreeElement または FileTreeElement からのツリーからの除去。

HTMLTree クラスは、階層ツリーを簡単に作成できるようにする以下の HTML クラスを使用します。

- HTMLTreeElement: ツリー要素を作成します。
- FileTreeElement: ファイル・ツリー要素を作成します。
- FileListElement: ファイル・リスト要素を作成します。
- » FileListRenderer: ファイルおよびディレクトリーのリストを渡します。«

例

以下の例では、HTMLTree クラスを使用するさまざまな方法を示します。

- 例: HTMLTree クラスを使用する
- 例: 走査可能な統合ファイル・システム・ツリーを作成する

HTMLTreeElement クラス

HTMLTreeElement クラスは、 HTMLTree または他の HTMLTreeElement 内の階層要素を表します。

HTMLTreeElement クラスにより提供されるメソッドを使用すると、 多くのツリー属性を検索して更新することができます。 これらのメソッドを使用すれば、以下のアクションを実行できます。

- ツリー要素の表示可能テキストを取得または設定する
- 展開アイコンおよび縮小アイコンの URL を取得または設定する
- ツリー要素が展開されるかどうかを設定する

以下の例では、HTMLTreeElement オブジェクトを作成し、タグを表示します。

```
// Create an HTMLTree.
HTMLTree tree = new HTMLTree();

// Create parent HTMLTreeElement.
HTMLTreeElement parentElement = new HTMLTreeElement();
parentElement.setTextUrl(new HTMLHyperlink("http://myWebPage", "My Web Page"));


// Create HTMLTreeElement Child.
HTMLTreeElement childElement = new HTMLTreeElement();
childElement.setTextUrl(new HTMLHyperlink("http://anotherWebPage", "Another Web Page"));
parentElement.addElement(childElement);

// Add the tree element to the tree.
tree.addElement(parentElement);
System.out.println(tree.getTag());
```

上記の例の getTag() メソッドは、 下記のような HTML タグを生成します。

```
<table cellpadding="0" cellspacing="3">
<tr>
<td><font color="#0000FF"><u>-</u></font> </td>
<td><font color="#0000FF"><u>My Web Page</u></font></td>
</tr>

<tr>
<td> </td>
<td>
<table cellpadding="0" cellspacing="3">
<tr>
<td><font color="#0000FF"><u>-</u></font> </td>
<td><font color="#0000FF"><u>Another Web Page</u></font> </td>
</tr>
</table>
</td>
</tr>
</table>
```

FileTreeElement クラス

FileTreeElement クラスは、 HTMLTree ビュー内にある統合ファイル・システムを表します。

HTMLTreeElement クラスにより提供されるメソッドを使用すると、 多くのツリー属性を検索して更新することができます。 » さらに NetServer 共用ドライブの名前とパスを取得および設定できます。 «

これらのメソッドが実行を可能にするアクションを以下にいくつか示します。

- 展開アイコンおよび縮小アイコンの URL を取得または設定する (継承されたメソッド)
- ツリー要素が展開されるかどうかを設定する (継承されたメソッド)
- NetServer 共用ドライブの名前を » 取得または設定する «
- NetServer 共用ドライブのパスを»取得または設定する«

以下の例では、FileTreeElement オブジェクトを作成し、タグを表示します。

```
// Create an HTMLTree.
HTMLTree tree = new HTMLTree();

// Create a URLParser object.
URLParser urlParser = new URLParser(httpServletRequest.getRequestURI());

// Create an AS400 object.
AS400 system = new AS400(mySystem, myUserId, myPassword);

// Create an IFSJavaFile object.
IFSJavaFile root = new IFSJavaFile(system, "/QIBM");

// Create a DirFilter object and get the directories.
DirFilter filter = new DirFilter();
File[] dirList = root.listFiles(filter);

for (int i=0; i < dirList.length; i++)
{

    // Create a FileTreeElement.
    FileTreeElement node = new FileTreeElement(dirList[i]);

    // Set the Icon URL.
    ServletHyperlink sl = new ServletHyperlink(urlParser.getURI());
    sl.setHttpServletResponse(resp);
    element.setIconUrl(sl);

    // Add the FileTreeElement to the tree.
```

```
            tree.addElement(element);
        }

        System.out.println(tree.getTag());
```

上記の getTag() メソッドは、 例に示す出力を与えます。

FileListElement クラス

FileListElement クラスを使用して、 統合ファイル・システム・ディレクトリーの内容を表すファイル・リスト要素を作成できます。

» NetServer 共用ドライブの名前およびパスを取得して設定することにより、FileListElement オブジェクトを使用して NetServer 共用ドライブの内容を表すことができます。 «

FileListElement クラスには、以下のことを行えるメソッドがあります。

- ファイル・リスト要素をリストしたり、ソートしたりする。
- HTTP サーブレット要求を取得したり、設定したりする。
- FileListRenderer を取得したり、設定したりする。
- ファイル・リストを表示する HTMLTable を取得したり、設定したりする。
- » NetServer 共用ドライブの名前の取得または設定«
- » NetServer 共用ドライブのパスの取得または設定«

FileListElement クラスを HTML パッケージの他のクラスと一緒に使用できます。

- FileListRenderer と使用する場合、 ファイルのリストをどのように表示するかを指定できます。
- FileTreeElement クラスと使用する場合、統合ファイル・システム・ファイル »または NetServer 共用ファイル «の走査可能なリストを作成することができます。

FileListElement javadoc は、 FileListElement オブジェクトの作成および表示の方法を示します。

例

以下の例では、FileListElement クラスを HTMLTree クラス (FileTreeElement および HTMLTreeElement) と共に使用して、 走査可能な統合ファイル・システム・ツリーを作成する方法を示します。 » さらに例には、NetServer 共用ドライブのパスを設定するコードも組み込まれています。 «

- 例: 走査可能な統合ファイル・システム・ツリーを作成する

»FileListRenderer クラス

FileListRenderer クラスは、 File オブジェクト (ディレクトリーおよびファイル) のすべてのフィールドを FileListElement に渡します。

FileListRenderer クラスは、以下のアクションを実行できるようにするメソッドを提供します。

- ディレクトリーの名前を取得する
- ファイルの名前を取得する
- 親ディレクトリーの名前を取得する
- FileListElement に表示する行データを戻す

この例では、renderer によって FileListElement オブジェクトを作成します。

```
// Create a FileListElement.
FileListElement fileList = new FileListElement(sys, httpservletrequest);

// Set the renderer specific to this servlet, which extends
// FileListRenderer and overrides applicable methods.
fileList.setRenderer(new myFileListRenderer(request));
```

デフォルト・レンダラーを使用したくない場合には、FileListRenderer を拡張し、 メソッドを指定変更するか新規のものを作成します。 たとえば、特定のディレクトリーまたはある拡張子の付いたファイルの名前を FileListElement に渡さないようにすることができます。 クラスを拡張し、適切なメソッドを指定変更することにより、これらのファイルおよびディレクトリーにヌル値を戻し、それらが表示されないようにできます。

FileListElement の行を完全にカスタマイズするには、 getRowData() メソッドを使用します。 getRowData() を使用して行データをカスタマイズする例としては、 行データに列を追加したり、列を再配置します。 FileListRenderer のデフォルト動作が適合する場合、 FileListElement クラスがデフォルトの FileListRenderer を作成するのでさらにプログラミングする必要はありません。 «

ReportWriter クラス

com.ibm.as400.util.reportwriter パッケージで提供されるクラスは、 サーブレットまたは JavaServer Pages(TM) により作られるデータ、 あるいは XML ソース・ファイルからのデータを、 より簡単に操作し、またフォーマットするために iSeries を使用することを可能にします。 reportwriter パッケージは、 異なってはいても関連した 3 つのパッケージを名前で見分けるのに好都合です。

- com.ibm.as400.util.reportwriter.pclwriter
- com.ibm.as400.util.reportwriter.pdfwriter
- com.ibm.as400.util.reportwriter.processor

» これらのパッケージには、さまざまなクラスが組み込まれています。 それらのクラスを使用すると、XML データ・ストリームをフォーマットし、 それらのクラスの様式で報告書を生成することをができます。 CLASSPATH の中に必要な JAR ファイルがあることを確認してください。 reportwriter の JAR ファイルについての詳細は、JAR ファイルを参照してください。 «

(pclwriter および pdfwriter パッケージ内にある) Context クラスは、ReportProcessor クラスが、XML データおよび JSP データを選択した様式で表現するために必要なメソッドを定義します。

- PCLContext を ReportWriter クラスと組み合わせて使用すると、 報告書を Hewlett Packard プリンター・コントロール言語 (PCL) フォーマットで生成することができます。
- PDFContext を ReportWriter クラスと組み合わせて使用すると、 報告書を Adobe Portable Document Format (PDF) で生成することができます。

(processor パッケージの中にある) ReportProcessor クラスを使用すると、 アプリケーションが XML ソース・データ、Java サーブレット、 および JavaServer Pages (JSP) から収集する情報を元に、 フォーマット済み報告書を生成することができます。

- JSPReportProcessor クラスを使用して、 サーブレットおよび JSP ページからデータを検索し、 使用可能な様式 (コンテキスト) で報告書を作成します。
- XSLReportProcessor クラスを使用して、 XML データを XSL スタイルシートで処理し、使用可能な様式 (コンテキスト) で報告書を作成します。

Context クラス

Context クラスは特定のデータ形式をサポートします。すなわち、 OutputQueue および SpooledFileOutputStream クラスの組み合わせで、 ReportWriter クラスがこの形式のレポートを生成し、そのレポートをスプール・ファイルに書き込むことができるようにします。

アプリケーションは Context クラスのインスタンスを作成することだけが必要で、その後 ReportWriter クラスがそれを使ってレポートを生成します。アプリケーションは、いずれの Context クラス内のメソッドも決して直接呼び出さないでください。 PCLContext および PDFContext メソッドは、 ReportWriter クラスにより内部的に使用されることになっています。

Context クラスのインスタンスを構成するには、 OutputStream (java.io パッケージから) および PageFormat (java.awt.print パッケージから) が必要です。以下の例は、レポートを生成するために他の ReportWriter クラスと共に Context クラスを構成して使用する方法を示します。

例: PCLContext と共に XSLReportProcessor を使用する

例: PDFContext と共に JSPReportProcessor を使用する

JSPReportProcessor クラス

JSPReportProcessor クラスにより、 JavaServer Page(TM) (JSP) または Java サーブレットから文書または報告書を作成できます。

このクラスを使用して、指定された URL から JSP またはサーブレットを入手し、その内容から文書を作成します。 JSP またはサーブレットは、XSL formatting object を含む文書データを提供する必要があります。 文書のページを生成する前に、出力コンテキストおよび JSP 入力データ・ソースを指定しなければなりません。 その後、報告書データを指定された出力データ・ストリーム形式に変換することができます。

JSPReportProcessor クラスを使用して、以下を行うことができます。

- 報告書の処理
- テンプレートとしての URL の設定

以下の例は、JSPReportProcessor および PDFContext クラスを使用して報告書を生成する方法を示します。 » 以下の例には Java および JSP コードが含まれ、下記のリンクを使用して表示できます。 JSPReportProcessor および XSLReportProcessor の例のための JSP、XML、 および XSL ソース・ファイルの例を含む、 ZIP ファイルをダウンロードすることもできます。 «

- 例: PDFContext と共に JSPReportProcessor を使用する
- » 例: JSPReportProcessor サンプル JSP ファイル «

JSP についての詳細は、Sun 社の Java Web サイト の Java Server Pages technology を参照してください。

XSLReportProcessor クラス

XSLReportProcessor クラスを使用すると、 XSL スタイルシートを使用して XML ソース・データを変換およびフォーマットすることにより、 文書あるいは報告書を作成することが可能になります。 XSL フォーマット・オブジェクト (FO) を含んでいる XSL スタイルシートを使用して報告書を作成するために、このクラスを使用してください。XSL FO は、XSL の仕様に準拠している必要があります。 その後 Context クラスを使用して、報告書データを指定された出力データ・ストリーム形式に変換します。

XSLReportProcessor クラスを使用して、以下を行うことができます。

- XSL スタイルシートの設定
- XML データ・ソースの設定
- XSL FO ソースの設定
- 報告書の処理

以下の例は、XSLReportProcessor クラスを使用して報告書を生成する方法を示しています。 » その例には、Java、XML、および XSL のコードが含まれており、 それらのコードは、以下のリンクを使用して表示することができます。 さらに、XSLReportProcessor の例および JSPReportProcessor の両方の例のための、 XML、XSL、および JSP のソース・ファイル例を含む ZIP ファイルをダウンロードすることもできます。 «

- 例: PCLContext と共に XSLReportProcessor を使用する
- » 例: XSLReportProcessor サンプル XML ファイル «
- » 例: XSLReportProcessor サンプル XSL ファイル «

XML および XSL についての詳細は、 Information Center の XML のトピックを参照してください。

リソース・クラス

com.ibm.as400.resource パッケージには、 各種の AS400 オブジェクトおよびリストを扱うための汎用フレームワークが用意されています。 このフレームワークにより、そのようなすべてのオブジェクトおよびリストへの、 一貫性のあるプログラミング・インターフェースが提供されます。

リソース・パッケージには、以下のクラスが含まれています。

- Resource - ユーザー、プリンター、ジョブ、 メッセージ、またはファイルなど、iSeries リソースを表すオブジェクト。 リソースの具象サブクラスには、次のものが含まれます。
  - RIFSFile
  - RJavaProgram
  - RJob
  - RPrinter
  - RQueuedMessage
  - RSoftwareResource
  - RUser

  注: access パッケージ内の NetServer クラスも、 Resource の具象サブクラスです。
- ResourceList - ユーザー、プリンター、ジョブ、 メッセージ、またはファイルのリストなど、iSeries リソースのリストを表すオブジェクト。 リソースの具象サブクラスには、次のものが含まれます。
  - RIFSFileList
  - RJobList
  - RJobLog
  - RMessageQueue
  - RPrinterList
  - RUserList
- プレゼンテーション - エンド・ユーザーにリソース・オブジェクト、 リソース・リスト、属性、選択、およびソートを表示するために使用できるオブジェクト。

Resource および ChangeableResource クラス

com.ibm.as400.resource.Resource および com.ibm.as400.resource.ChangeableResource 抽象クラスは、 iSeries リソースを表します。

Resource

Resource は、任意のリソースの属性への汎用アクセスを提供する抽象クラスです。 それぞれの属性は属性 ID を使用して識別されます。 Resource の特定のサブクラスでは、通常、Resource によってサポートされる属性 ID を記述します。

Resource では、属性値への読み取りアクセスだけが提供されます。

IBM Toolbox for Java では、以下のリソース・オブジェクトを提供しています。

- RIFSFile - iSeries 統合ファイル・システム内のファイルまたはディレクトリーを表します。
- RJavaProgram - iSeries 上の Java プログラムを表します。
- RJob - iSeries ジョブを表します。
- RPrinter - iSeries プリンターを表します。
- RQueuedMessage - iSeries メッセージ待ち行列またはジョブ・ログ内のメッセージを表します。
- RSoftwareResource - iSeries 上のライセンス・プログラムを表します。
- RUser - iSeries ユーザーを表します。

ChangeableResource

ChangeableResource 抽象クラスは Resource のサブクラスであり、 iSeries リソースの属性値を変更するための機能を追加します。 属性の変更は、コミットされるかまたは取り消されるまで、内部でキャッシュに入れられます。 これにより、一度に複数の属性値を変更することができます。

注: access パッケージ内の NetServer クラスも、 Resource および ChangeableResource の具象サブクラスです。

例

以下の例では、Resource および ChangeableResource の具象サブクラスを直接に使用する方法と、 汎用コードで Resource または ChangeableResource サブクラスを扱う方法を示します。

- Resource の具象サブクラスである RUser からの属性値の取り出し
- ChangeableResource の具象サブクラスである RJob の属性値の設定
- 汎用コードを使用してリソースにアクセスする

リソース・リスト

com.ibm.as400.resource.ResourceList クラスは、 iSeries リソースのリストを表します。 これは、リストの内容への汎用アクセスを提供する抽象クラスです。

IBM Toolbox for Java では、以下のリソース・リストを提供します。

- RIFSFileList - iSeries 統合ファイル・システム内のファイルおよびディレクトリーのリストを表します。
- RJobList - iSeries ジョブのリストを表します。
- RJobLog - iSeries ジョブ・ログ内のメッセージのリストを表します。
- RMessageQueue - iSeries メッセージ待ち行列内のメッセージのリストを表します。
- RPrinterList - iSeries プリンターのリストを表します。
- RUserList - iSeries ユーザーのリストを表します。

リソース・リストは常に、オープンされているか、クローズされているかのどちらかです。 リソース・リストの内容にアクセスするには、それをオープンしなければなりません。 リストの内容への即時アクセスを提供し、メモリーを効率的に管理するために、 大部分のリソース・リストは増分的にロードされます。

リソース・リストを使用して、以下のことを行うことができます。

- リストのオープン
- リストのクローズ
- リスト内の特定のリソースへのアクセス
- 特定のリソースのロードを待つ
- 完全なリソース・リストのロードを待つ

さらに、選択値を使用することによって、リソース・リストをフィルターに掛けることができます。 それぞれの選択値は、選択 ID を使用して識別されます。 同様に、リソース・リストは、ソート値を使用してソートすることができます。 それぞれのソート値は、ソート ID を使用して識別されます。 ResourceList の特定のサブクラスでは、通常、 ResourceList によってサポートされる選択 ID とソート ID を記述します。

例

以下の例では、リソース・リストを扱うさまざまな方法を示します。

- 例: ResourceList の内容を取得および印刷する
- 例: 汎用コードを使用して ResourceList にアクセスする
- 例: サーブレットでリソース・リストを表示する (HTML テーブル)

Presentation クラス

すべてのリソース・オブジェクト、リソース・リスト、およびメタデータ・オブジェクトは、関連する com.ibm.as400.resource.Presentation オブジェクトを持ちます。 このオブジェクトは、名前、フルネーム、およびアイコンなど、変換された情報を提供するものです。

例: プレゼンテーションを使用してリソース・リストとそのソート値を印刷する

プレゼンテーション情報を使用して、エンド・ユーザーにリソース・オブジェクト、リソース・リスト、属性、 選択、およびソートをテキスト形式で表示することができます。

```
void printCurrentSort(ResourceList resourceList) throws ResourceException
{
    // Get the presentation for the ResourceList and print its full name.
    Presentation resourceListPresentation = resourceList.getPresentation();
    System.out.println(resourceListPresentation.getFullName());

    // Get the current sort value.
    Object[] sortIDs = resourceList.getSortValue();

    // Print each sort ID.
    for(int i = 0; i < sortIDs.length; ++i)
    {
        ResourceMetaData sortMetaData = resourceList.getSortMetaData(sortIDs[i]);
        System.out.println("Sorting by " + sortMetaData.getName());
    }
}
```

セキュリティー・クラス

IBM Toolbox for Java のセキュリティー・クラスは、サーバーへのセキュアな接続を提供し、ユーザーの識別を検査し、 ローカル・サーバーでの実行時にユーザーにオペレーティング・システム・スレッドを関連付けるために使用します。 組み込まれているセキュリティー・サービスは、次のとおりです。

- » Java Secure Socket Extension (JSSE) は、 クライアントとサーバー・セッションの間で交換されるデータを暗号化し、 サーバー認証を実行することにより、セキュアな接続を提供します。

  注: Secure Sockets Layer (SSL) の使用に関する情報は、 後方互換性についてのみ取り上げています。 «
- 認証サービスは、以下の機能を提供します。
  - ユーザーの識別およびパスワードを OS/400 ユーザー・レジストリーと照合して認証する。
  - 現行の OS/400 スレッドに識別を割り当てる。

Secure Sockets Layer

Secure Sockets Layer (SSL) は、次のことを行ってセキュア接続を提供します。

- クライアントとサーバーのセッションの間で交換されるデータの暗号化
- サーバー認証の実行

» 注: セキュア接続を提供するために以下の方式を取り入れる代わりに、 Java Secure Socket Extension (JSSE) を使用することを検討してください。 SSL の使用に関する以下の情報は、後方互換性のためにのみ提供されています。 «

SSL 接続の実行速度は、暗号化を使用しない接続の場合よりも遅いため、 SSL を使用するとパフォーマンスが低下します。 SSL 接続は、転送されるデータの機密性がパフォーマンスのコスト増加に見合う場合 (たとえば、 クレジット・カードや銀行口座計算書の情報を転送する場合) に使用してください。

SSL を IBM Toolbox for Java と共に使うためには、 法的責任を理解しておかなければなりません。

»SSL のアルゴリズム

IBM Toolbox for Java には、データの暗号化および暗号解除に必要なアルゴリズムは含まれていません。 V5R2 では、以下のアルゴリズムは、 iSeries Client Encryption (128-bit) ライセンス・プログラム (5722-CE3) と共に出荷されます。

注: Toolbox for Java は、 iSeries Client Encryption (56-bit) ライセンス・プログラム (5722-CE2) とも互換性がありますが、 このプログラムはもはや更新されませんし、V5R2 で利用できません。 Client Encryption (56-bit) が持っているアルゴリズムは、 Client Encryption (128-bit) のものほど強力でないため、 128-bit encryption にアップグレードすることを検討してください。

Client Encryption (128-bit) (5722-CE3) の詳細または注文方法については、IBM 担当員にお問い合わせください。 «

SSL 環境をセットアップする

IBM Toolbox for Java は、SSL を使用してデータを暗号化するための 2 つの環境に対応します。 これらの環境を正しくセットアップする必要があります。

- IBM Toolbox for Java クラスと OS/400 サーバーの間で暗号化を使用する
- Proxy クライアントと Proxy サーバーの間で暗号化を使用する

»IBM Toolbox for Java の以前のバージョンとの互換性

IBM Toolbox for Java、暗号化アルゴリズム、 およびキー・リング・クラス・ファイル の V5R2 バージョンを使用する場合、 Client Encryption ライセンス・プログラムの V5R1 あるいは V5R2 バージョンのいずれかを使用しなければなりません。

注: OS/400 バージョン V4R5 以前のものからアップグレードする場合は、KeyRing.class ファイルを更新する必要があります。

V5R2 の IBM Toolbox for Java および互換性のあるバージョンの Client Encryption をクライアント上で使用しているならば、 OS/400 の V4R4 以上のバージョンに接続することができます。 互換性のあるバージョンの Client Encryption についての詳細は、 SSL のアルゴリズムを参照してください。«

SSL に関する法的責任

IBM iSeries Client Encryption (128-bit) ライセンス製品は、 128 ビット暗号化アルゴリズムを使用して SSL バージョン 3.0 暗号化サポートを提供します。

このプログラムには、米国商務省の特別輸出ライセンス条件の規制を受けるデータ暗号化が含まれています。 他の国や地域においても、輸出入ライセンス条件が適用される場合があります。

このため、お客様と同じ国や地域または異なる国や地域のユーザーによる同一プログラムの使用、 またはそれらのユーザーへのプログラムの転送は、禁止されるか、 あるいは以下の規制を受ける可能性があります。

- ユーザーの国における輸入関連の特別な法律、規定、または政策
- お客様の国における輸出関連の特別な法律、規定、または政策

お客様には、現時点から (さらに、ライセンスの期限を超えても)、 プログラムがすべての該当する輸出入関連の法律、規定、 および政策に従って使用または転送されるように保証する責任があります。

お客様とお客様のユーザーは、その他の国や地域の輸出入法に従う必要があります。

SSL を使用して IBM Toolbox for Java と OS/400 サーバーの間でデータを暗号化する

SSL を使用して、IBM Toolbox for Java クラスと OS/400 サーバーの間で交換されるデータを暗号化することができます。 クライアント側では、IBM iSeries Client Encryption ライセンス・プログラム (5722-CE2 または 5722-CE3) に付属しているファイルを使用して、 データを暗号化します。 サーバー側では、OS/400 ディジタル証明書マネージャーを使用して、 暗号化されたデータを交換するように OS/400 サーバーを構成しなければなりません。

SSL を使用するためにクライアントとサーバーをセットアップする

IBM Toolbox for Java クラスと OS/400 サーバーの間でやり取りされるデータを暗号化するには、 以下の作業を行ってください。

1. 暗号化されたデータを交換するためにサーバーをセットアップします。
2. 暗号化されたデータを交換するためにクライアント (IBM Toolbox for Java クラス) をセットアップします。 このステップの手順は、サーバー上で SSL をセットアップする際に使用した証明書の種類によって異なります。
   - 承認を受けた機関からのサーバー証明書を使用する
   - 自己署名証明書を使用する

     注: 承認を受けた機関からの証明書を使用してクライアントをセットアップする方が、 自己署名証明書を使用するよりもはるかに簡単で迅速です。
3. SecureAS400 オブジェクトを使用して、 IBM Toolbox for Java がデータを暗号化するように強制します。

   注: 上記の最初の 2 つのステップが完了すると、 クライアントとサーバーの間のセキュア・パスが作成されます。 アプリケーションでは、SecureAS400 オブジェクトを使用して、 どのデータを暗号化するかを IBM Toolbox for Java に指示しなければなりません。 SecureAS400 オブジェクトを通じてやり取りされるデータだけが暗号化されます。 AS400 オブジェクトを使用する場合には、データは暗号化されず、 サーバーへの通常のパスが使用されます。

SSL を使用するための iSeries サーバーのセットアップ

SSL を IBM Toolbox for Java と共に使用するために iSeries サーバーをセットアップするには、 以下の手順に従ってください。

1. »iSeries サーバーに以下のものをインストールします。
   - IBM Cryptographic Access Provider 128-bit for iSeries (5722-AC3)。 これは、サーバー側の暗号化を行います。
   - IBM iSeries Client Encryption (128-bit) (5722-CE3)。 これは、クライアント側の IBM Toolbox for Java クラスによって使用される Java クラスおよびユーティリティーを提供します。

   注: Toolbox for Java は、 V5R1 バージョンの Cryptographic Access Provider 56-bit for iSeries (5722-AC2) および V5R1 バージョンの Client Encryption (56-bit) (5722-CE2) とも互換性があります。 «
2. クライアント暗号化ファイルを含むディレクトリーの権限を変更します。
3. サーバー証明書を取得および構成します。
4. IBM Toolbox for Java によって使用される以下の iSeries サーバーに証明書を適用します。
   - QIBM_OS400_QZBS_SVR_CENTRAL
   - QIBM_OS400_QZBS_SVR_DATABASE
   - QIBM_OS400_QZBS_SVR_DTAQ
   - QIBM_OS400_QZBS_SVR_NETPRT
   - QIBM_OS400_QZBS_SVR_RMTCMD
   - QIBM_OS400_QZBS_SVR_SIGNON
   - QIBM_OS400_QZBS_SVR_FILE
   - QIBM_OS400_QRW_SVR_DDM_DRDA

クライアント暗号化ファイルを含むディレクトリーの権限を変更する

暗号化アルゴリズムの使用時に必要とされる SSL に関する法的責任を満たす上で役立つように、 ファイルを含むディレクトリーは、共通認可 *EXCLUDE を持つ状態で出荷されています。 暗号化アルゴリズムの使用を許可されたユーザーによるアクセスだけを許可するために、 ディレクトリーの権限を変更しなければなりません。

OS/400 オブジェクト・セキュリティーを使用してクライアント暗号化ファイルへのアクセスを制御するには、 以下の手順に従ってください。

1. サーバーで、次のコマンドを入力します。

   ```
   wrklnk '/QIBM/ProdData/HTTP/Public/jt400/*'
   ```
2. SSL56 または SSL128 ディレクトリーでオプション 9 を選択します。
3. *PUBLIC に *EXCLUDE 権限があることを確認します。
4. SSL ファイルにアクセスする必要があるユーザーまたはユーザーのグループに、 ディレクトリーに対する *RX 権限を与えます。

   注: *ALLOBJ 特殊権限を持つユーザーに対して、SSL ファイルへのアクセスを否認することはできません。

サーバー証明書を取得および構成する

サーバー証明書を取得および構成する前に、次の製品をインストールする必要があります。

- IBM HTTP Server for iSeries (5722-DG1) ライセンス・プログラム
- 基本オペレーティング・システム・オプション 34 (ディジタル証明書マネージャー)

サーバー証明書を取得および構成するために行う処理は、 使用する証明書の種類によって異なります。

- 承認を受けた機関 (VeriSign, Inc. や RSA Data Security, Inc. など) から証明書を取得する場合は、 iSeries に証明書をインストールした後で、それをホスト・サーバーに適用します。
- 認証を受けた機関からの証明書を使用しない場合は、 iSeries 上で使用される独自の証明書を作成することができます。 証明書を作成するには、ディジタル証明書マネージャーを使用します。
  1. iSeries サーバー上で認証局を作成します。 Information Center のトピック独自の CA としての役割を参照してください。
  2. 作成した認証局からシステム証明書を作成します。
  3. 作成したシステム証明書を使用するホスト・サーバーを割り当てます。

承認を受けた機関からの証明書を使用する

IBM Toolbox for Java には、以下の会社に代表される、 承認を受けた一連の機関からのサーバー証明書をサポートするキー・リング・ファイルが付属しています。

- IBM World Registry
- Integrion Financial Network
- RSA Data Security, Inc.
- Thawte Consulting
- VeriSign, Inc.

キー・リング・ファイルでは、 これらの承認を受けた機関のいずれかから取得する証明書がすでにサポートされています。 行わなければならないのは、暗号化アルゴリズムが入った ZIP ファイルを入手することと、 それを CLASSPATH ステートメントに追加することだけです。

証明書を使用するには、以下の手順に従ってください。

1. ZIP ファイルを置くディレクトリーを選択します。
2. 使用したい SSL のバージョンをダウンロードします。 それには、選択したディレクトリーに ZIP ファイルをコピーします。
    - 56 ビット暗号化 (ライセンス・プログラム 5722-CE2 で使用される) の場合、 /QIBM/ProdData/HTTP/Public/jt400/SSL56/sslightx.zip をコピーします。
    - 128 ビット暗号化 (ライセンス・プログラム 5722-CE3 で使用される) の場合、 /QIBM/ProdData/HTTP/Public/jt400/SSL128/sslightu.zip をコピーします。
3. ZIP ファイルを CLASSPATH ステートメントに追加します。

自己署名証明書を使用する

» 承認を受けた機関からの証明書を使用しない場合には、 自己署名した認証局 (CA) 証明書を (自己署名 CA 証明書を持つそれぞれのサーバーから) ダウンロードして、 IBM Toolbox for Java クラスがそれを使用できるようにしなければなりません。 « さらに、暗号化アルゴリズムが入った ZIP ファイルを入手し、それを CLASSPATH ステートメントに追加する必要があります。

自己署名証明書を使用するには、以下の手順に従ってください。

1. ZIP ファイルを置くディレクトリーを選択します。
2. 使用したい SSL のバージョンをダウンロードします。 それには、暗号化アルゴリズムと、自己署名証明書の処理に必要なユーティリティーの両方をコピーします。
   - 56 ビット暗号化 (ライセンス・プログラム 5722-CE2 で使用される) の場合、 /QIBM/ProdData/HTTP/Public/jt400/SSL56/sslightx.zip、cfwk.zip、および ssltools.jar をコピーします。
   - 128 ビット暗号化 (ライセンス・プログラム 5722-CE3 で使用される) の場合、 /QIBM/ProdData/HTTP/Public/jt400/SSL128/sslightu.zip、cfwk.zip、および ssltools.jar をコピーします。
3. ssltools.jar および ZIP ファイルを CLASSPATH ステートメントに追加します。
4. クライアント上で、<SSL>\com\ibm\as400\access という名前のディレクトリーを作成します。 <SSL> は、JAR および ZIP ファイルをコピーしたディレクトリーです。
5. クライアント上の <SSL> ディレクトリー内のコマンド・プロンプトから、 次のコマンドを実行します。

   ```
   java utilities.KeyringDB com.ibm.as400.access.KeyRing -connect <systemname>:<port>
   ```

   <port> は、いずれかのホスト・サーバーのサーバー・ポートです。 たとえば、iSeries 上のセキュア・サインオン・サーバーのデフォルト・ポートである 9476 を使用することができます。
6. »キー・リングに追加する認証局 (CA) 証明書の番号を入力します。 « サイト証明書ではなく、必ず CA 証明書を追加してください。
7. 証明書名を入力するようにプロンプトが出されたら、 任意の英数字ストリングを入力することができます。

   注: 自己署名証明書を持つ各サーバーに対して KeyringDB を実行して、 それぞれの証明書を KeyRing クラスに追加することが必要です。 SSL 接続を使用する予定のそれぞれの iSeries で、次のコマンドを実行して証明書を追加してください。

   ```
   java utilities.KeyringDB com.ibm.as400.access.KeyRing -connect <systemname>:<port>
   ```

上記の手順を終えれば、自己署名証明書のセットアップは完了です。 CLASSPATH ステートメントに次のものが入っていることを確認した後で、 アプリケーションを実行することができます。

- com\ibm\as400\access\KeyRing.class を含むディレクトリー
- jt400.jar
- sslightx.zip または sslightu.zip (ダウンロードしたファイルによって異なる)

jt400.jar には KeyRing.class クラスのデフォルトのコピーが入っているため、 com\ibm\as400\access\KeyRing.class を含むディレクトリーは、 CLASSPATH の中で jt400.jar の前になければなりません。

注: KeyRing.class ファイルを含むディレクトリーを CLASSPATH ステートメントに追加する代わりに、 jt400.jar 内の古いクラスを新しい KeyRing.class に置き換えることができます。

SSL を使用して Proxy クライアントと Proxy サーバーの間でデータを暗号化する

SSL を使用して、Proxy クライアントと Proxy サーバーの間で交換されるデータを暗号化することができます。 データの暗号化には、IBM iSeries Client Encryption ライセンス・プログラム (5722-CE2 または 5722-CE3) に付属しているファイルを使用します。 IBM Toolbox for Java と同様に、これらのファイルはプラットフォームに依存しない Java クラスです。 このため、Proxy クライアントと Proxy サーバーは、 Java 仮想マシンを持つ任意のプラットフォームで実行することができます。

Proxy クライアントと Proxy サーバーの間でやり取りされるデータを暗号化するには、 以下の作業を行ってください。

1. 暗号化されたデータを処理するために Proxy サーバーをセットアップする。
2. 暗号化されたデータを処理するために Proxy クライアントをセットアップする。
3. SecureAS400 オブジェクトを使用して、 IBM Toolbox for Java がデータを暗号化するように強制する。

注: 最初の 2 つのステップでは、 Proxy クライアントと Proxy サーバーの間のセキュア・パスが作成されるだけです。 IBM Toolbox for Java に、セキュア・パスを介してデータを送るよう指示するには、 アプリケーションで SecureAS400 オブジェクトを使用しなければなりません。 AS400 オブジェクトを使用すると、データは暗号化されず、 サーバーへの通常のパスが使用されます。

Proxy クライアントと Proxy サーバーの間でやり取りされるデータを暗号化したい場合には、 Client Encryption ライセンス・プログラム (5722-CE2 または 5722-CE3) に付属している Java クラスだけが必要です。 Proxy サーバーと iSeries サーバーの間でやり取りされるデータを暗号化したい場合には、 さらに、Proxy サーバーと iSeries サーバーの間の暗号化をセットアップする必要があります。

Proxy サーバーでの SSL のセットアップ

SSL を使用できるようにするには、Proxy サーバーにサーバー証明書がなければなりません。 Proxy サーバーで使用されるサーバー証明書を作成するには、 IKeyman グラフィカル・ユーザー・インターフェース (GUI) を使用してください。 IKeyman は GUI ツールであるため、クライアントで実行しなければなりません。 証明書を作成したら、それを iSeries にコピーすることができます (Proxy サーバーが iSeries で実行される場合)。

暗号化されたデータを処理するために Proxy サーバーをセットアップするには、以下の作業を行ってください。

1. IKeyman GUI を実行するためにクライアントをセットアップします。
2. Proxy サーバーのサーバー証明書を作成します。
3. 作成した証明書を使用して、Proxy サーバーを開始します。

IKeyman GUI を実行するためにクライアントをセットアップする

IKeyman GUI は、Java Swing 1.1 インターフェースを基にした Java プログラムです。 IKeyman を使用するためには、 クライアントで Java 1.1.8 JVM (および Swing 1.1 プラグイン) または Java 2 JVM が実行されていなければなりません。

IKeyman GUI は、ssltools.jar 内の IBM iSeries Client Encryption ライセンス・プログラム (5722-CE2 または 5722-CE3) の一部です。 SSL を使用するため (および IKeyman を実行するため) にクライアントをセットアップする手順は、 実行されているライセンス・プログラムのバージョンによって異なります。

SSL を使用するためにクライアントをセットアップするには、以下の手順に従ってください。

1. ワークステーション上で、必要な JAR および ZIP ファイルを置くディレクトリーを選択します。
2. 選択したディレクトリーに、必要なファイルをコピーします。
    - 56 ビット暗号化を使用する場合は、iSeries 上でライセンス・プログラム 5722-CE2 をロードした後で、 ワークステーションに次のファイルをコピーします。
        - /QIBM/ProdData/http/public/jt400/ssl56/sslightx.zip
        - /QIBM/ProdData/http/public/jt400/ssl56/ssltools.jar
        - /QIBM/ProdData/http/public/jt400/ssl56/cfwk.zip
        - /QIBM/ProdData/http/public/jt400/ssl56/cfwk.sec
    - 128 ビット暗号化を使用する場合は、iSeries 上でライセンス・プログラム 5722-CE3 をロードした後で、 ワークステーションに次のファイル

をコピーします。

- /QIBM/ProdData/http/public/jt400/ssl128/sslightu.zip
- /QIBM/ProdData/http/public/jt400/ssl128/ssltools.jar
- /QIBM/ProdData/http/public/jt400/ssl128/cfwk.zip
- /QIBM/ProdData/http/public/jt400/ssl128/cfwk.sec

3. JAR ファイルと ZIP ファイルを CLASSPATH ステートメントに追加します。.sec ファイルを CLASSPATH に追加しないでください。

   注: cfwk.zip は、CLASSPATH 内の最初の項目でなければなりません。

サーバー証明書を作成する

自己署名証明書を作成するには、IKeyman GUI を使用します。

注: IKeyman GUI の実行が停止された場合は、 cfwk.zip が CLASSPATH の最初の項目であることと、 cfwk.sec が cfwk.zip と同じディレクトリーにあることを確認してください。

Proxy サーバーのサーバー証明書を作成するには、以下の手順に従ってください。

1. 次のコマンドを使用して IKeyman GUI を開始します。

   ```
   java -Dkeyman.javaOnly=true com.ibm.gsk.ikeyman.Ikeyman
   ```

2. IKeyman の「キー・データベース・ファイル (Key Database File)」メニューから、 「新規 (New)」を選択します。
3. 「新規 (New)」ダイアログでは、 「キー・データベース・タイプ (Key Database Type)」を変更しないでください。 これは、「SSLight キー・データベース・クラス (SSLight key database class)」でなければなりません。
4. 「ファイル名 (File Name)」(たとえば、ProxyServerKeyring.class) を入力するか、 または「参照 (Browse)」をクリックして、キー・リングのために使用したいクラス・ファイルを見つけます。

   注: キー・リング・ファイル名は、 セキュア Proxy サーバーを開始するために必要となるため、覚えておいてください。
5. 「位置 (Location)」(パス) を入力するか、 またはデフォルトの位置 (現行作業ディレクトリー) を受け入れ、 「OK」をクリックします。
6. 「パスワード・プロンプト (Password Prompt)」ダイアログで、 「パスワード (Password)」と「パスワードの確認 (Confirm Password)」を入力し、 「OK」をクリックします。 (「満了時刻の設定 (Set expiration time)」はオプションであり、 選択する必要はありません。)

   注: パスワードは、 セキュア Proxy サーバーを開始するために必要となるため、覚えておいてください。 このダイアログのかぎ型のアイコンは、パスワードの相対的な強度を表します。 強度の高いパスワードには、大文字と小

文字の混ざった英数字が必要です。

7. IKeyman の「作成 (Create)」ファイル・メニューから、 「新規の自己署名証明書 (New Self-Signed Certificate)」 を選択します。
8. 「新規の自己署名証明書の作成 (Create New Self-Signed Certificate)」 ダイアログで、 「キー・ラベル (Key Label)」(たとえば、MyCertificate) と「組織名 (Organization)」を入力します。
9. 「国 (Country)」リストをクリックして国または地域を選択し、 「有効期間 (Validity Period)」を入力するかまたはデフォルトを受け入れ、 「OK」をクリックします。
10. IKeyman の「キー・データベース・ファイル (Key Database File)」メニューから、 「クローズ (Close)」を選択し、 (同じメニューから)「終了 (Exit)」をクリックします。

これで、現行ディレクトリーに、作成したキー・リングが見られるようになるはずです。

新しい証明書を使用して Proxy サーバーを開始する

Proxy サーバーを開始する前に、Proxy サーバーの CLASSPATH に jt400.jar、sslightx.zip、および Proxy サーバーのキー・リングの位置が含まれていることを確認してください。

作成した証明書を使用して、Proxy サーバーを開始してください。 この情報を Proxy サーバーに渡すには、-keyringName および -keyringPassword パラメーターを使用します。 たとえば、次のようにします。

```
java com.ibm.as400.access.ProxyServer -keyringName ProxyServerKeyring
-keyringPassword pxypswrd
```

Proxy クライアントでの SSL のセットアップ

以下の手順では、クライアント上の証明書データベースにサーバー証明書を追加する方法を説明します。 この証明書は、Java .class ファイルに保管されます。 サーバーは自己署名証明書を使用するため、クライアントにサーバー証明書を追加することが必要です。

暗号化されたデータを交換するために Proxy クライアントをセットアップするには、 以下の作業を行ってください。

1. 暗号化されたデータを処理するために Proxy サーバーをセットアップし、 その後、Proxy サーバーを開始します。
2. SSL を使用するためにクライアントをセットアップします。
3. KeyringDB を使用して、Proxy サーバーのサーバー証明書を取得します。
4. 更新された KeyRing.class ファイルを使用するためにクライアントをセットアップします。
5. クライアント上でセキュア Proxy 設定を設定します。

SSL を使用するためにクライアントをセットアップする

証明書 (KeyringDB) をダウンロードするツールは、Java プログラムです。 このプログラムを使用するには、クライアントで Java 1.1.8 または Java 2 JVM が実行されていなければなりません。 KeyringDB は、 ssltools.jar 内の IBM iSeries Client Encryption ライセンス・プログラム (5722-CE2 または 5722-CE3) の一部です。 SSL を使用するためにクライアントをセットアップする手順は、 実行されているライセンス・プログラムのバージョンによって異なります。

Proxy サーバーをセットアップした後で、 以下の手順に従って、SSL を使用するためにクライアントをセットアップしてください。

1. ワークステーション上で、必要な JAR および ZIP ファイルを置くディレクトリーを選択します。
2. 選択したディレクトリーに、必要なファイルをコピーします。
   - 56 ビット暗号化を使用する場合は、iSeries 上でライセンス・プログラム 5722-CE2 をロードした後で、 ワークステーションに次のファイルをコピーします。
     - /QIBM/ProdData/http/public/jt400/ssl56/sslightx.zip
     - /QIBM/ProdData/http/public/jt400/ssl56/ssltools.jar
     - /QIBM/ProdData/http/public/jt400/ssl56/cfwk.zip
   - 128 ビット暗号化を使用する場合は、サーバー上でライセンス・プログラム 5722-CE3 をロードした後、 ワークステーションに次のファイルをコピーします。
     - /QIBM/ProdData/http/public/jt400/ssl128/sslightu.zip
     - /QIBM/ProdData/http/public/jt400/ssl128/ssltools.jar
     - /QIBM/ProdData/http/public/jt400/ssl128/cfwk.zip
3. JAR ファイルと ZIP ファイルを CLASSPATH ステートメントに追加します。
4. クライアント上で、<SSL>\com\ibm\as400\access という名前のディレクトリーを作成します。 <SSL> は、JAR および ZIP ファイルをコピーしたディレクトリーです。

Proxy サーバーのサーバー証明書を追加する

KeyringDB は、サーバー証明書を含む新しい KeyRing.class ファイルを作成し、 それを現行ディレクトリーの com\ibm\as400\access サブディレクトリーに置きます。

KeyringDB ツールを使用して KeyRing.class にサーバー証明書を追加するには、 以下の手順に従ってください。

1. JAR および ZIP ファイルを置いたディレクトリーから、次のコマンドを実行します。

   ```
   java utilities.KeyringDB com.ibm.as400.access.KeyRing -connect proxyServerName:port
   ```

   ここで:

   - *proxyServerName* は、Proxy サーバーを実行するマシンの名前です。
   - *port* は、セキュア Proxy サーバーが listen するポート (デフォルトでは、3471) です。

   たとえば、次のようにします。

   ```
   java utilities.KeyringDB com.ibm.as400.access.KeyRing -connect myProxyServer:3471
   ```

2. 使用する証明書を尋ねるプロンプトが出されたら、サイト証明書 0 を選択します。
3. 証明書名を入力するようにプロンプトが出されたら、 任意の英数字ストリングを入力することができます。

更新された KeyRing.class ファイルを使用するためにクライアントをセットアップする

jt400Proxy.jar ファイルには、KeyRing.class が含まれています。 更新された KeyRing.class ファイルを使用するためにクライアントをセットアップするには、 CLASSPATH ステートメントに次のものが入っていることを確認してください。

- com\ibm\as400\access\KeyRing.class を含むディレクトリー
- jt400Proxy.jar
- sslightx.zip または sslightu.zip (ダウンロードしたファイルによって異なる)
- cfwk.zip

jt400Proxy には、KeyRing.class クラスのデフォルトのコピーが入っているため、com\ibm\as400\access\KeyRing.class を含むディレクトリーは、 CLASSPATH の中で jt400Proxy の前になければなりません。

注: KeyRing.class ファイルを含むディレクトリーを CLASSPATH ステートメントに追加する代わりに、 新しい KeyRing.class を jt400Proxy.jar ファイルに追加することができます。 新しい KeyRing.class を jt400Proxy.jar に追加すると、古いバージョンが上書きされます。

クライアント上でセキュア Proxy 設定を設定する

Proxy クライアントに、セキュア接続を介して Proxy サーバーと通信するよう指示するには、次のシステム・プロパティーを設定してください。

com.ibm.as400.access.AS400.proxyServer=*proxyServer*

proxyServer は、Proxy サーバーを実行するマシンの名前です。

com.ibm.as400.access.SecureAS400.proxyEncryptionMode=*mode*

mode は、次のいずれかの整数です。

- 1 (Proxy クライアントと Proxy サーバーの間の暗号化)
- 2 (Proxy サーバーと iSeries サーバーの間の暗号化)
- 3 (Proxy クライアントと Proxy サーバーの間、 および Proxy サーバーと iSeries サーバーの間の暗号化)

たとえば、次のコマンドは、SSL を使用してアプリケーションを開始します。

```
java -Dcom.ibm.as400.access.AS400.proxyServer=myProxyServer
   -Dcom.ibm.as400.access.SecureAS400.proxyEncryptionMode=1   myApplication
```

認証サービス

IBM Toolbox for Java では、 OS/400 によって提供されるセキュリティー・サービスと対話するクラスが用意されています。 特に、ユーザーの識別 (プリンシパル と呼ばれることもある) とパスワードを OS/400 ユーザー・レジストリーと照合して認証するためのサポートが提供されます。 したがって、認証済みのユーザーを表す信任証を確立することができます。 信任証を使用すると、現行 OS/400 スレッドの識別を、 認証済みのユーザーの権限および許可に基づいて作業を実行するように変更することができます。 実際に、この識別の交換により、スレッドは、 認証済みのユーザーによってサインオンが実行されたかのように動作するようになります。

注: 信任証を確立して交換するためのサービスは、 リリース V5R1M0 以上のサーバーでのみサポートされます。

提供されるサポートの概要

AS400 オブジェクトは、 サーバーに対する特定のユーザー・プロファイルとパスワードに認証を提供するようになりました。 さらに、システムに対する認証済みのユーザー・プロファイルとパスワードを表す » Kerberos チケットおよびプロファイル・トークン « を検索することもできます。

»注: Kerberos チケットを使用するためには、J2SDK、v1.4 をインストールし、Java 汎用セキュリティー・サービス (JGSS) アプリケーション・プログラミング・インターフェースを構成する必要があります。 JGSS に関する詳細は、J2SDK, v1.4 Security Documentation を参照してください。«

»Kerberos チケットの使用の際は、 システム名のみ (パスワードは設定しない) を AS400 オブジェクト内に設定してください。 ユーザーの識別は、JGSS フレームワークを通して検索されます。 AS400 オブジェクトに 1 度に設定できる認証の方法は 1 つだけです。 パスワードを設定すると、いずれの Kerberos チケットあるいはプロファイル・トークンも消去されます。«

プロファイル・トークンを使用するには、 getProfileToken() メソッドを使用して、 ProfileTokenCredential クラスのインスタンスを検索します。 プロファイル・トークンは、 特定のサーバーに対する認証済みのユーザー・プロファイルとパスワードを表すものと考えてください。 プロファイル・トークンの有効期限は時間に基づいており、最長 1 時間で有効期限は切れますが、 場合によっては更新を行って存続期間を長くすることができます。

»以下の例では、 システム・オブジェクトを作成し、そのオブジェクトを使用してプロファイル・トークンを生成します。 その例ではその後、プロファイル

・トークンを使用してもう 1 つ別のシステム・オブジェクトを作成し、 その 2 番目のシステム・オブジェクトを使用してコマンド・サービスに接続します。

```
AS400 system = new AS400("mySystemName", "MYUSERID", "MYPASSWORD");
ProfileTokenCredential myPT = system.getProfileToken();
AS400 system2 = new AS400("mySystemName", myPT);
system2.connectService(AS400.COMMAND); «
```

スレッドの識別を設定する

信任証は、リモート・コンテキストかローカル・コンテキスト上で確立することができます。 信任証を作成すると、呼び出しアプリケーションの要求に応じて、 これをシリアル化したり配布したりすることができます。 関連したサーバー上の実行プロセスに渡されると、 OS/400 スレッドの識別を変更または交換 したり、 事前に認証されたユーザーに代わって作業を実行するために、 信任証を使用することができます。

このサポートの実際的な適用は、2 層にわたる適用となります。 つまり、最初の層 (すなわち PC) で、 ユーザー・プロファイルとパスワードの認証がグラフィカル・ユーザー・インターフェースによって行われ、 2 番目の層 (サーバー) で、そのユーザーのために作業が実行されます。 ProfileTokenCredentials を使用することによって、アプリケーションは、 ユーザー ID とパスワードをネットワークを介して直接渡すことを避けることができます。 その後、プロファイル・トークンは 2 番目の層にあるプログラムに配布されます。 このプログラムは、*swap()* を実行し、 ユーザーに割り当てられた OS/400 権限と許可の下で機能します。

注: プロファイル・トークンは、存続期間が限られているため、 ユーザー・プロファイルとパスワードを渡すことよりも本質的にセキュアですが、 それでも、アプリケーションで機密情報として見なし、そのように扱うことが必要です。 トークンは、認証済みのユーザーとパスワードを表すので、 そのユーザーに代わって作業を実行するために、攻撃的なアプリケーションによって不正に使用される可能性があります。 最終的には、信任証がセキュアな方法でアクセスされるように保証するのはアプリケーションの責任となります。

例

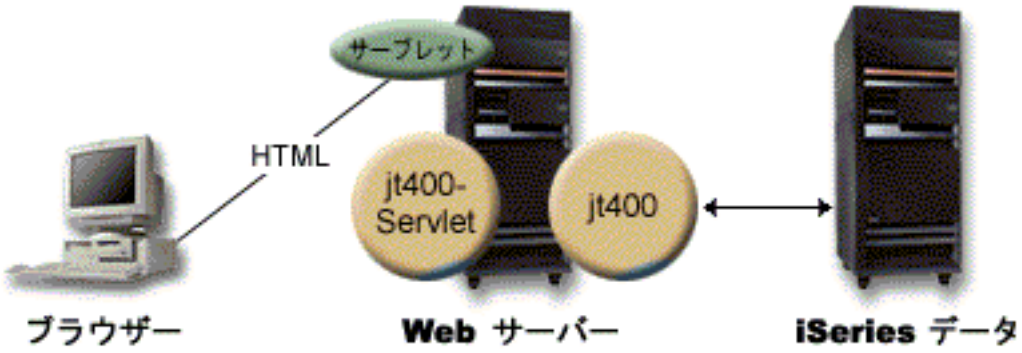
プロファイル・トークンを使用して OS/400 スレッドの識別を交換し、 特定のユーザーに代わって作業を実行する方法の例については、 このコードを参照してください。

サーブレット・クラス

IBM Toolbox for Java とともに提供されるサーブレット・クラスは、 Web サーバーに置かれているアクセス・クラスを使用して、 ユーザーが iSeries サーバー上の情報にアクセスできるようにします。 独自のサーブレット・プロジェクトに役立てるために、サーブレット・クラスを使用する方法を決定してください。

以下の図では、ブラウザー、Web サーバー、 および iSeries データの間でサーブレット・クラスが機能する方法を示します。 ブラウザーは、サーブレットを実行している Web サーバーに接続します。 サーブレット・クラスは、データおよびデータを渡す HTML クラスを検索するためにいくつかのアクセス・クラスを使用するため、 »jt400Servlet.jar ファイルと jt400.jar ファイル«は Web サーバーに常駐しています。 Web サーバーは、データが置かれている iSeries システムに接続されます。

図 1: サーブレットの仕組み



IBM Toolbox for Java には、次の 4 つのタイプのサーブレット・クラスが組み込まれています。

- 認証クラス
- RowData クラス
- RowMetaData クラス
- Converter クラス

注: jt400Servlet.jar ファイルには、 HTML クラスとサーブレット・クラスの両方が入っています。 » com.ibm.as400.util.html および com.ibm.as400.util.servlet パッケージの中でクラスを使用する場合は、 jt400Servlet.jar と jt400.jar の両方を指すように CLASSPATH を更新する必要があります。 «

サーブレットの一般情報については、 参照のセクションを参照してください。

認証クラス

サーブレット・パッケージ内の 2 つのクラス (AuthenticationServlet と AS400Servlet) は、 サーブレットに対する認証を実行します。

AuthenticationServlet クラス

AuthenticationServlet は、 サーブレットに対する基本認証を実行する HttpServlet のインプリメンテーションです。 AuthenticationServlet のサブクラスでは、 次のメソッドの 1 つまたは複数のものを変更しなければなりません。

- 認証を実行するために validateAuthority() メソッドを変更する (必須)
- サブクラスが特定の要求だけを認証するように bypassAuthentication() メソッドを変更する
- 認証の後で要求の追加処理を許可するために postValidation() メソッドを変更する

AuthenticationServlet クラスには、以下のことを行えるメソッドがあります。

- サーブレットを初期設定する。
- 認証済みのユーザー ID を取得する。
- 認証をバイパスした後でユーザー ID を設定する。
- 例外およびメッセージを記録する。

AS400Servlet クラス

AS400Servlet クラスは、 HTML サーブレットを表す、AuthenticationServlet の抽象サブクラスです。 接続プールを使用して、接続を共用し、 サーブレットのユーザーがサーブレットに対して持つことができる接続の数を管理することができます。

AS400Servlet クラスには、以下のことを行えるメソッドがあります。

- ユーザー権限の妥当性検査を行う (AuthenticationServlet クラスの validateAuthority() メソッドを変更することによる)。
- システムに接続する。
- プールから接続プール・オブジェクトを取得したり、プールに返却したりする。
- 接続プールをクローズする。
- HTML 文書の見出しタグを取得および設定する。
- HTML 文書の終了タグを取得および設定する。

サーブレットの一般情報については、 参照のセクションを参照してください。

RowData クラス

RowData クラスは、 データのリストを記述およびアクセスするための手段となる抽象クラスです。

RowData クラスを拡張する 4 つの主要なクラスがあります。

- ListRowData
- RecordListRowData
- »ResourceListRowData«
- SQLResultSetRowData

RowData クラスを使用すると、以下のことを行うことができます。

- 現在位置の取得および設定
- getObject() メソッドによる、特定の列にある行データの取得
- 行のメタデータの取得
- 特定の列にあるオブジェクトのプロパティーの取得または設定
- length() メソッドの使用によるリスト内の行数の取得

RowData の位置

リスト内の現在位置を取得および設定するのに使用できるメソッドは、いくつかあります。 以下の表では、RowData クラス用の設定メソッドと取得メソッドの両方をリストしています。

設定メソッド		取得メソッド
absolute()	next()	getCurrentPosition()
afterLast()	previous()	isAfterLast()
beforeFirst()	relative()	isBeforeFirst()
first()		isFirst()
last()		isLast()

ListRowData クラス

ListRowData クラスでは、以下のことが行えます。

- 結果リストに行を追加したり、行を消去する。
- 行の取得および設定
- getMetaData() メソッドを使用して、リストの列についての情報を取得する。
- setMetaData() メソッドで、列情報を設定する。

ListRowData クラスはデータのリストを表します。 ListRowData は、IBM Toolbox for Java アクセス・クラスを使用して、 以下のものを含む多くの情報の種類を表します。

- 統合ファイル・システム内のディレクトリー
- ジョブ・リスト
- メッセージ待ち行列内のメッセージ・リスト
- ユーザー・リスト
- プリンター・リスト
- スプール・ファイル・リスト

この例は、 ListRowData および HTMLTableConverter の動作方法を示します。 また、Java コード、HTML コード、および HTML の外観を示します。

RecordListRowData クラス

RecordListRowData クラスを使用すると、以下のことを行うことができます。

- レコード・リストの行の追加および除去
- 行の取得および設定
- setRecordFormat メソッドによるレコード様式の設定
- レコード様式の取得

RecordListRowData クラスは、レコードのリストを表します。 レコードは、以下を含むさまざまな形式でサーバーから取得できます。

- サーバー・ファイルで読み書きされるレコード
- データ待ち行列内の項目
- プログラム呼び出しからのパラメーター・データ
- サーバー形式と Java 形式との間で変換しなければならない戻りデータ

この例は、RecordListRowData および HTMLTableConverter の動作方法を示します。 また、Java コード、HTML コード、および HTML の外観を示します。

»ResourceListRowData クラス

ResourceListRowData クラスはデータのリソース・リストを表します。 ResourceListRowData オブジェクトを使用して、 ResourceList インターフェースの任意のインプリメンテーションを表すことができます。

リソース・リストは、一連の行の形式にフォーマット設定されており、 各行は、列属性 ID の数値により決定される有限数の列を含んでいます。 行内部の各列には、個別のデータ項目が入っています。

ResourceListRowData クラスは、以下の処理の実行を可能にするメソッドを提供しています。

- 列属性 ID を取得したり、 設定したりする。
- リソース・リストを取得したり、 設定したりする。
- リスト内の行数を取得する。
- 現在行の列データを取得する。
- データ・オブジェクトのプロパティー・リストを取得する。
- リスト用のメタデータを取得する。

例: サーブレットでリソース・リストを表示する。«

SQLResultSetRowData クラス

SQLResultSetRowData クラスは、 SQL ResultSet をデータのリストとして表します。 このデータは、JDBC を介して SQL ステートメントによって生成されます。 提供されているメソッドを使用して、ResultSet メタデータを取得したり、設定することができます。

この例は、 ListRowData および HTMLTableConverter の動作方法を示します。 また、Java コード、HTML コード、および HTML の外観を示します。

RowMetaData クラス

RowMetaData クラスは、 RowData オブジェクトの列に関する情報を検出するために使用されるインターフェースを定義します。

RowMetaData クラスを使用して、以下を行うことができます。

- 列の数の取得
- 列の名前、 タイプ、 またはサイズの取得
- 列ラベルの取得または設定
- 列データの精度または位取りの取得
- 列データがテキスト・データであるかどうかの判別

RowMetaData クラスを実装するクラスは、主に 3 つあります。 これらのクラスには、上記のすべての RowMetaData 関数と、固有の関数が備わっています。

- ListMetaData
- RecordFormatMetaData
- SQLResultSetMetaData

ListMetaData クラス

ListMetaData を使用すれば、 ListRowData クラスの情報を取得して列の設定を変更できます。 setColumns() メソッドを使用すれば、 列数を設定できます。その際には、以前の列情報はすべて消去されます。 また、コンストラクターのパラメーターを設定する際に列の数を受け渡すこともできます。

この例は、 ListMetaData、ListRowData および HTMLTableConverter の動作方法を示します。 また、Java コード、HTML コード、および HTML の外観を示します。

RecordFormatMetaData クラス

RecordFormatMetaData は、 IBM Toolbox for Java の RecordFormat クラスを利用します。このクラスを使用すると、コンストラクターのパラメーターを設定するときや、 get および set メソッドを使用してレコード様式にアクセスするときに、 レコード様式を提供することができます。

以下の例では、RecordFormatMetaData オブジェクトを作成する方法を示します。

```
// Create a RecordFormatMetaData object from a sequential file's record format.
RecordFormat recordFormat = sequentialFile.getRecordFormat();
RecordFormatMetaData metadata = new RecordFormatMetaData(recordFormat);

// Display the file's column names.
int numberOfColumns = metadata.getColumnCount();
for (int column=0; column < numberOfColumns; column++)
{
   System.out.println(metadata.getColumnName(column));
}
```

SQLResultSetMetaData クラス

SQLResultSetMetaData クラスは、 SQLResultSetRowData オブジェクトの列に関する情報を戻します。 ResultSet は、コンストラクターのパラメーターを設定するとき、 あるいは get および set メソッドを使用して、 ResultSet のメタデータにアクセスするときに提供することができます。

以下の例では、SQLResultSetMetaData オブジェクトを作成する方法を示します。

```
// Create an SQLResultSetMetaData object from the result set's metadata.
SQLResultSetRowData rowdata = new SQLResultSetRowData(resultSet);
SQLResultSetMetaData sqlMetadata  = rowdata.getMetaData();

// Display the column precision for non-text columns.
String name = null;
int numberOfColumns = sqlMetadata.getColumnCount();
for (int column=0; column < numberOfColumns; column++)
{
   name = sqlMetadata.getColumnName(column);
   if (sqlMetadata.isTextData(column))
   {
      System.out.println("Column: " + name + " contains text data.");
   }
   else
   {
      System.out.println("Column: " + name + " has a precision of " +
sqlMetadata.getPrecision(column));
   }
}
```

コンバーター・クラス

コンバーター・クラスを使用すると、行データを形式設定されたストリング配列に変換できます。変換の結果は HTML フォームになって、 HTML ページ上で表示することができるようになります。以下のクラスによってこのような変換が行われます。

- StringConverter
- HTMLFormConverter
- HTMLTableConverter

StringConverter クラス

StringConverter クラスは、 行データ・ストリング・コンバーターを表す抽象クラスです。 このクラスでは、行データを変換するための convert() メソッドが提供されます。 このメソッドは、その行のデータをストリング配列で表現して戻します。

HTMLFormConverter クラス

HTMLFormConverter クラスは、 追加の変換メソッドである convertToForms() を提供することにより、 StringConverter を拡張します。 このメソッドは、 行データを単一行の HTML テーブルの配列に変換します。 これらのテーブル・タグを使用して、 ブラウザー上に定様式の情報を表示できます。

さまざまな取得メソッドや設定メソッドを使用して、フォームの属性を表示または変更することにより、 HTML フォームの表示方法を変更することができます。 たとえば、設定できる属性には以下のようなものがあります。

- 位置合わせ
- セル・スペーシング
- ヘッダー・ハイパーリンク
- 幅

例

例: HTMLFormConverter を使用する。 (この例のコンパイルおよび実行は、Web サーバーの実行中に行ってください)

HTMLTableConverter クラス

HTMLTableConverter クラスは、 convertToTables() メソッドを提供することによって、 StringConverter を拡張します。 このメソッドは、行データを、サーブレットがブラウザーでリストを表示するのに使用できる HTML テーブルの配列に変換します。

getTable() メソッドや setTable() メソッドを使用すると、 変換中に使用されるデフォルトのテーブルを選択することができます。 HTML テーブル・オブジェクト内でテーブル・ヘッダーを設定することもできますし、 setUseMetaData() を true に設定して、 ヘッダー情報のメタデータを使用することもできます。

setMaximumTableSize() メソッドを使用すると、 1 つのテーブルでの行の数を制限することができます。 行データが、指定されたテーブルのサイズに適合しない場合、 コンバーターは出力配列で別の HTML テーブル・オブジェクトを生成します。 この処理は、すべての行データの変換が完了するまで継続します。

例

以下の例では、HTMLTableConverter クラスを使用する方法を示します。

- 例: ListRowData を使用する
- 例: RecordListRowData を使用する
- 例: SQLResultSetRowData を使用する
- 例: サーブレットで ResourceList を表示する

ユーティリティー・クラス

ユーティリティー・クラスは、特定のタスクの実行を援助します。 IBM Toolbox for Java は次のユーティリティーを提供します。

- AS400ToolboxInstaller: クライアント上で IBM Toolbox for Java クラスをインストールおよび更新できるようにします。 この機能は、Java プログラムとしてもアプリケーション・プログラミング・インターフェース (API) としても使用できます。
- AS400ToolboxJarMaker: 大きい JAR ファイルから小さい JAR ファイルを作成するか、 または JAR ファイルを選択して解凍して個々の内容ファイルへのアクセスを取得することにより、 高速ロード可能な IBM Toolbox for Java の JAR ファイルを生成します。
- CommandPrompter: 指定されたコマンドのパラメーターの入力を求めるプロンプトを出します。 CommandPrompter は、iSeries の CL コマンド・プロンプト (F4 を押す) に類似した機能およびマネージメント・セントラル・コマンド・プロンプトと同様の機能を提供します。
- RunJavaApplication および VRunJavaApplication: コマンド行プロンプトから iSeries サーバー上で Java プログラムを実行できるようにします。
- JPing: どのサービスが活動状態であるかを検出するために、 サーバーの照会を行えるようにします。 また、SSL ポートを PING するかどうかをも指定することができます。

クライアント・インストールおよびクラスの更新

IBM Toolbox for Java クラスは、サーバーの統合ファイル・システム上に配置されています。 プログラム一時修正 (PTF) は、これらのクラスが配置されているディレクトリーに適用されるので、 サーバーの上記クラスに直接アクセスする Java プログラムは、自動的にその更新を受けることができます。 サーバーからのクラスへのアクセスは常に動作するわけではなく、特に以下の状態ではそう言えます。

- 低速通信リンクでサーバーおよびクライアント間が接続されている場合、 サーバーからクラスをロードする際に好ましいパフォーマンスが得られないことがあります。
- Java アプリケーションが CLASSPATH 環境変数を用いてクライアントのファイル・システム上にあるクラスにアクセスする場合、 iSeries Access for Windows を用いたファイル・システム呼び出しの宛先をサーバーに変更することが必要になります。 iSeries Access for Windows が、クライアント上に存在できないことがあります。

上記の場合、クライアント上にクラスをインストールした方がより良い結果を得ることができます。 AS400ToolboxInstaller クラスは、 IBM Toolbox for Java クラスがクライアント上にある場合に、 それらを管理するためのクライアント・インストールおよび更新機能を提供します。

AS400ToolboxInstaller の使用

»AS400ToolboxInstaller オブジェクトはプログラムであり、 プログラマブル・インターフェースでもあります。 オブジェクトには main() メソッドがありますので、コマンド行から実行できます。 さらに共通ワーカー・メソッドもあるため、 アプリケーションに組み込み、アプリケーションから実行することもできます。

AS400ToolboxInstaller オブジェクトは、 クライアント上の Toolbox ファイルをインストールするためにも、 それらを最新のものに保持するためにも使用できます。 初めて使用される際、Toolbox ファイルはサーバーからワークステーションにコピーされます。 PTF がサーバーに適用される場合、 AS400ToolboxInstaller オブジェクトはサーバー上の新規ファイルでワークステーションのファイルを更新します。«

AS400ToolboxInstaller クラスは、 クライアントのローカル・ファイル・システムにファイルをコピーします。 このクラスはアプレットでは機能しないことがあります。 多くのブラウザーでは、Java プログラムはローカル・ファイル・システムへの書き込みを行えないからです。

コマンド行からの AS400ToolboxInstaller クラスの実行

AS400ToolboxInstaller クラスはスタンドアロン型プログラムとして使用でき、 コマンド行から実行できます。 コマンド行から AS400ToolboxInstaller を実行するということは、 ユーザーがプログラムを作成する必要がないということです。 そのかわり、IBM Toolbox for Java クラスのインストール、 アンインストール、または更新を行うには、このクラスを Java アプリケーションとして実行します。

適切なインストール、アンインストール、または比較オプションを指定して、 以下のコマンドで AS400ToolboxInstaller クラスを呼び出してください。

```
java utilities.AS400ToolboxInstaller [options]
```

-source オプションは IBM Toolbox for Java クラスの検出先を示し、 -target は IBM Toolbox for Java クラスの格納先を示します。

ツールボックス全体または特定の関数だけをインストールするオプションも提供されています。 »たとえば、ワークステーションで Toolbox for Java クラス・パッケージ (jt400.jar) のクラスをインストールまたは更新するには、 以下のコマンドを使用します。

```
java utilities.AS400ToolboxInstaller -install -package ACCESS -source myAS400
-target c:/toolbox
```

上記の例では、c:\toolbox は Toolbox for Java の jar ファイルが含まれるディレクトリーであると想定してい

ます。 «

プログラムへの AS400ToolboxInstaller クラスの組み込み

AS400ToolboxInstaller クラスは、 クライアント上のプログラム内での IBM Toolbox for Java クラスのインストール、 アンインストールおよび更新に必要な、 アプリケーション・プログラミング・インターフェース (API) を提供します。

クライアントで IBM Toolbox for Java クラスをインストールまたは更新するには、 install() メソッドを使います。インストールまたは更新のためには、Java プログラムでソース・パス、 ターゲット・パス、およびクラス・パッケージの名前を指定します。 ソースの URL は、サーバー上の制御ファイルの位置を示します。 ディレクトリー構造は、サーバーからクライアントにコピーされます。

install() メソッドはファイルのコピーだけを行い、 CLASSPATH 環境変数は更新しません。 install() メソッドが成功すると、 Java プログラムは getClasspathAdditions() メソッドを呼び出して、 CLASSPATH 環境変数に追加しなければならないものを確認することができます。

以下の例では、AS400ToolboxInstaller クラスを使用して、 "mySystem" という名前のサーバーからドライブ D: のディレクトリー "jt400" に、 ファイルをインストールする方法を示します。 その後に、CLASSPATH 環境変数に何を追加しなければならないかを判別する方法を示します。

```
                    // Install the IBM Toolbox for Java
                    // classes on the client.
URL sourceURL = new URL("http://mySystem.myCompany.com/QIBM/ProdData/HTTP/Public/jt400/");

if (AS400ToolboxInstaller.install(
        "ACCESS",
        "d://jt400",
        sourceURL))

{
                    // If the IBM Toolbox for Java classes were installed
                    // or updated, find out what must be added to the
                    // CLASSPATH environment variable.
   Vector additions = AS400ToolboxInstaller.getClasspathAdditions();

                    // If updates must be made to CLASSPATH
   if (additions.size() > 0)
   {
                    // ... Process each classpath addition
   }
}

                    // ... Else no updates were needed.
```

IBM Toolbox for Java のクラスがクライアント上にインストール済みかどうかを判別するには、 isInstalled() メソッドを使用します。 isInstalled() メソッドを使うと、その時点でインストールを完了するのか、 または都合の良いときまで延期するのかを決定できます。

install() メソッドは、クライアントでのファイルのインストールと更新を行います。 Java プログラムは isUpdateNeeded() メソッドを呼び出して、 install() を呼び出す前に更新する必要があるかどうかを判別することができます。

クライアントから IBM Toolbox for Java クラスを除去するには、 unInstall() メソッドを使います。 unInstall メソッドはファイルを除去するだけで、 CLASSPATH 環境変数は変更しません。 getClasspathRemovals() メソッドを呼び出して、 CLASSPATH 環境変数から除去できるものを判別してください。

クライアント・ワークステーション上のプログラム内で AS400ToolboxInstaller クラスをインストールおよび更新する方法を示した例について詳しくは、 Install/Update の例を参照してください。

AS400ToolboxJarMaker

JAR ファイル形式はもともと Java プログラム・ファイルのダウンロードを高速化するよう設計されていますが、 AS400ToolboxJarMaker クラスは大きい JAR ファイルを小さい JAR ファイルに作成し直して、 IBM Toolbox for Java の JAR ファイルのロードをさらに高速化しています。

また、 AS400ToolboxJarMaker クラスは ZIP 圧縮された JAR ファイルを解凍することもできるので、 それぞれのファイルの内容にアクセスする基本的な方法として使用することができます。

AS400ToolboxJarMaker の柔軟性

AS400ToolboxJarMaker のすべての関数は、 以下のように JarMaker クラスおよび AS400ToolboxJarMaker サブクラスを使用して実行されます。

- 汎用 JarMaker ツールはどのような JAR ファイルまたは ZIP ファイルでも処理します。 このツールは JAR ファイルを分割し、 使用されないクラスを除去して、JAR ファイルのサイズを減らします。
- AS400ToolboxJarMaker は、 JarMaker 関数をカスタマイズして拡張し、IBM Toolbox for Java の JAR ファイルで簡単に使用できるようにします。

必要に応じて、AS400ToolboxJarMaker メソッドを独自の Java プログラム内か、またはコマンド行から呼び出すことができます。 コマンド行から AS400ToolboxJarMaker を呼び出すには、次の構文を使います。

```
java utilities.JarMaker [options]
```

ここで、

- options = 1 つ以上の使用可能なオプションです。

コマンド行プロンプトで実行できるオプションの詳細については、 以下を参照してください。

- JarMaker 基本クラスのオプション
- AS00ToolboxJarMaker サブクラスの拡張オプション

AS400ToolboxJarMaker の使用

JAR ファイルを解凍する

JAR ファイルに組み込まれているファイルの中から 1 つだけを解凍するとします。 AS400ToolboxJarMaker を使って、ファイルを以下のいずれかに拡張することができます。

- 現行ディレクトリー (extract(jarFile))
- 別のディレクトリー (extract(jarFile, outputDirectory))

たとえば、以下のコードを使用すれば、 AS400.class とそれに従属するすべてのクラスが jt400.jar から抽出されます。

```
java utilities.AS400ToolboxJarMaker -source jt400.jar
    -extract outputDir
    -requiredFile com/ibm/as400/access/AS400.class
```

1 つの JAR ファイルを複数のより小さい JAR ファイルに分割する

最大 JAR ファイル・サイズに応じて、 大きい JAR ファイルをそれよりも小さい JAR ファイルに

分割するとします。 AS400ToolboxJarMaker にはそれに対応する split(jarFile, splitSize) 関数が用意されています。

以下のコードでは、 jt400.jar が 300K 以下のファイルの集合に分割されます。

```
java utilities.AS400ToolboxJarMaker -split 300
```

JAR ファイルから使用しないファイルを除去する AS400ToolboxJarMaker を使用すれば、 アプリケーションで不要ないずれの IBM Toolbox for Java ファイルでも除外することができます。 そのためには、アプリケーションの実行に必要な IBM Toolbox for Java コンポーネント、 言語、および CCSID だけを選択します。 また AS400ToolboxJarMaker では、 選択したコンポーネントに関連付けられた JavaBean ファイルを組み入れたり除外したりするオプションも利用することができます。

たとえば、以下のコマンドは、 IBM Toolbox for Java の作業の CommandCall および ProgramCall コンポーネントを作成するのに必要な IBM Toolbox for Java クラスだけが入った JAR ファイルを作成します。

```
java utilities.AS400ToolboxJarMaker -component CommandCall,ProgramCall
```

さらに、 Unicode の変換テーブルと 2 バイト文字セット (DBCS) の変換テーブルとの間で、 テキスト・ストリングを変換する必要がない場合、 以下のように -ccsid オプションを指定して必要のない変換テーブルを省略すれば、 400K バイト小さい JAR ファイルを作成することができます。

```
java utilities.AS400ToolboxJarMaker -component CommandCall,ProgramCall -ccsid 61952
```

注: 変換クラスは、プログラム呼び出しクラスには入っていません。 プログラム呼び出しクラスを組み込む場合、-ccsid オプションを指定して、 プログラムが使用する変換クラスを明示的に入れなければなりません。

IBM Toolbox for Java でサポートされているコンポーネント

以下に AS400ToolboxJarMaker ツールを呼び出すときに指定できるコンポーネント ID を示します。

- 最初の列には、コンポーネントの共通名のリストが示されています。
- 2 番目の列には、 -component オプション・タグを使用するときに指定すべきキーワードが示されています。
- 3 番目の列には、 setComponents() および getComponents() に指定すべき整数値のリストが示されています。

コンポーネント	キーワード	定数
サーバー・オブジェクト	AS400	AS400ToolboxJarMaker.AS400
コマンド呼び出し	CommandCall	AS400ToolboxJarMaker.COMMAND_CALL
接続プール	ConnectionPool	AS400ToolboxJarMaker.CONNECTION_POOL
データ域	DataArea	AS400ToolboxJarMaker.DATA_AREA
データ記述および変換	DataDescription	AS400ToolboxJarMaker.DATA_DESCRIPTION
データ待ち行列	DataQueue	AS400ToolboxJarMaker.DATA_QUEUE
ディジタル証明書	DigitalCertificate	AS400ToolboxJarMaker.DIGITAL_CERTIFICATE
FTP	FTP	AS400ToolboxJarMaker.FTP
統合ファイル・システム	IntegratedFileSystem	AS400ToolboxJarMaker.INTEGRATED_FILE_SYSTEM
JAAS	JAAS	AS400ToolboxJarMaker.JAAS
Java アプリケーション呼び出し	JavaApplicationCall	AS400ToolboxJarMaker.JAVA_APPLICATION_CALL
JDBC	JDBC	AS400ToolboxJarMaker.JDBC
ジョブおよびジョブ待ち行列	Job	AS400ToolboxJarMaker.JOB
メッセージおよびメッセージ待ち行列	Message	AS400ToolboxJarMaker.MESSAGE

数値データ・タイプ	NumericDataTypes	AS400ToolboxJarMaker.NUMERIC_DATA_TYPES
»NetServer	NetServer	AS400ToolboxJarMaker.NETSERVER«
ネットワーク印刷	Print	AS400ToolboxJarMaker.PRINT
プログラム呼び出し	ProgramCall	AS400ToolboxJarMaker.PROGRAM_CALL
レコード・レベル・アクセス	RecordLevelAccess	AS400ToolboxJarMaker.RECORD_LEVEL_ACCESS
Secure Server	SecureAS400	AS400ToolboxJarMaker.SECURE_AS400
サービス・プログラム呼び出し	ServiceProgramCall	AS400ToolboxJarMaker.SERVICE_PROGRAM_CALL
システム状況	SystemStatus	AS400ToolboxJarMaker.SYSTEM_STATUS
システム値	SystemValue	AS400ToolboxJarMaker.SYSTEM_VALUE
トレースおよびログ記録	Trace	AS400ToolboxJarMaker.TRACE
ユーザーおよびグループ	User	AS400ToolboxJarMaker.USER
ユーザー・スペース	UserSpace	AS400ToolboxJarMaker.USER_SPACE
ビジュアル・サーバー・オブジェクト	AS400Visual	AS400ToolboxJarMaker.AS400_VISUAL
ビジュアル・コマンド呼び出し	CommandCallVisual	AS400ToolboxJarMaker.COMMAND_CALL_VISUAL
ビジュアル・データ待ち行列	DataQueueVisual	AS400ToolboxJarMaker.DATA_QUEUE_VISUAL

ビジュアル統合ファイル・システム	IntegratedFileSystemVisual	AS400ToolboxJarMaker.INTEGRATED_FILE_SYSTEM_VISUAL
ビジュアル Java アプリケーション呼び出し	JavaApplicationCallVisual	AS400ToolboxJarMaker.JAVA_APPLICATION_CALL_VISUAL
ビジュアル JDBC	JDBCVisual	AS400ToolboxJarMaker.JDBC_VISUAL
ビジュアル・ジョブおよびジョブ待ち行列	JobVisual	AS400ToolboxJarMaker.JOB_VISUAL
ビジュアル・メッセージおよびメッセージ待ち行列	MessageVisual	AS400ToolboxJarMaker.MESSAGE_VISUAL
ビジュアル・ネットワーク印刷	PrintVisual	AS400ToolboxJarMaker.PRINT_VISUAL
ビジュアル・プログラム呼び出し	ProgramCallVisual	AS400ToolboxJarMaker.PROGRAM_CALL_VISUAL
ビジュアル・レコード・レベル・アクセス	RecordLevelAccessVisual	AS400ToolboxJarMaker.RECORD_LEVEL_ACCESS_VISUAL
ビジュアル・ユーザーおよびグループ	UserVisual	AS400ToolboxJarMaker.USER_VISUAL

IBM Toolbox for Java でサポートされている CCSID およびエンコード値

ご購入の IBM Toolbox for Java には CCSID にちなんで命名された一連の変換テーブルが付属しています。 データを iSeries または AS/400e サーバーに (あるいは、 iSeries または AS/400e サーバーから) 変換するときに、 内部的に IBM Toolbox for Java クラス (CharConverter など) でこのテーブルは使われます。 たとえば、CCSID 1027 の変換テーブルはファイル com/ibm/as400/access/ConvTable1027.class 内にあります。 以下の CCSID 用の変換テーブルは IBM Toolbox for Java の JAR ファイルに組み込まれていますが、 JDK を使用するその他のエンコードもサポートされます。 サーバー上のセントラル・サーバーは今後、実行時のテーブルのダウンロードには使用されません。 指定した CCSID 用の変換テーブルや JDK エンコードが見つからないと、例外が出されることになります。 そのようなテーブルのうちの一部は、ご使用の JDK に備えられているテーブルの冗長テーブルであることがあります。 現在、IBM Toolbox for Java は、以下の 122 種類の iSeries および AS/400e の CCSID をサポートします。

iSeries および AS/400e サーバーで認識されている CCSID の全リストを含め、CCSID のその他の詳細は、Information Center のグローバリゼーションのトピックを参照してください。

IBM Toolbox for Java でサポートされる CCSID

CCSID	形式	説明
37	単一バイト EBCDIC	米国ほか
273	単一バイト EBCDIC	オーストリア、ドイツ
277	単一バイト EBCDIC	デンマーク、ノルウェー
278	単一バイト EBCDIC	フィンランド、スウェーデン
280	単一バイト EBCDIC	イタリア
284	単一バイト EBCDIC	スペイン、ラテンアメリカ
285	単一バイト EBCDIC	英国
290	単一バイト EBCDIC	日本語カタカナ (単一バイト)
297	単一バイト EBCDIC	フランス
300	2 バイト EBCDIC	日本語図形 (16684 のサブセット)
367	ASCII/ISO/Windows	ASCII (ANSI X3.4 標準)
420	単一バイト EBCDIC (両方向)	アラビア語 EBCDIC ST4

423	単一バイト EBCDIC	ギリシャ語 (互換性に関しては 875 を参照)
424	単一バイト EBCDIC (両方向)	ヘブライ語 EBCDIC ST4
437	ASCII/ISO/Windows	ASCII (USA PC データ)
500	単一バイト EBCDIC	ラテン 1 (MNCS)
720	ASCII/ISO/Windows	アラビア語 (MS-DOS)
737	ASCII/ISO/Windows	ギリシャ語 (MS-DOS)
775	ASCII/ISO/Windows	バルト語 (MS-DOS)
813	ASCII/ISO/Windows	ISO 8859-7 (ギリシャ語/ラテン語)
819	ASCII/ISO/Windows	ISO 8859-1 (ラテン 1)
833	単一バイト EBCDIC	韓国語 (単一バイトのみ)
834	2 バイト EBCDIC	韓国語図形 (4930 のサブセット)
835	2 バイト EBCDIC	中国語 (繁体字) 図形
836	単一バイト EBCDIC	中国語 (簡体字) (単一バイトのみ)
837	2 バイト EBCDIC	中国語 (簡体字) 図形
838	単一バイト EBCDIC	タイ語
850	ASCII/ISO/Windows	ラテン 1
851	ASCII/ISO/Windows	ギリシャ語
852	ASCII/ISO/Windows	ラテン語 2
855	ASCII/ISO/Windows	キリル文字
857	ASCII/ISO/Windows	トルコ語
860	ASCII/ISO/Windows	ポルトガル語
861	ASCII/ISO/Windows	アイスランド
862	ASCII/ISO/Windows (両方向)	ヘブライ語 ASCII ST4
863	ASCII/ISO/Windows	カナダ
864	ASCII/ISO/Windows (両方向)	アラビア語 ASCII ST5
865	ASCII/ISO/Windows	デンマーク/ノルウェー
866	ASCII/ISO/Windows	キリル文字/ロシア語
869	ASCII/ISO/Windows	ギリシャ語
870	単一バイト EBCDIC	ラテン語 2
871	単一バイト EBCDIC	アイスランド
874	ASCII/ISO/Windows	タイ語 (9066 のサブセット)
875	単一バイト EBCDIC	ギリシャ語
878	ASCII/ISO/Windows	ロシア語

880	単一バイト EBCDIC	キリル文字多国語 (互換性に関しては 1025 を参照)
912	ASCII/ISO/Windows	ISO 8859-2 (ラテン語 2)
914	ASCII/ISO/Windows	ISO 8859-4 (ラテン語 4)
915	ASCII/ISO/Windows	ISO 8859-5 (キリル文字 8 ビット)
916	ASCII/ISO/Windows (両方向)	ISO 8859-8 (ヘブライ語) ST5
920	ASCII/ISO/Windows	ISO 8859-9 (ラテン語 5)
921	ASCII/ISO/Windows	ISO 8859-13 (バルト語 8 ビット)
922	ASCII/ISO/Windows	エストニア語 ISO-8
923	ASCII/ISO/Windows	ISO 8859-15 (ラテン語 9)
930	混合バイト EBCDIC	日本語 (5026 のサブセット)
933	混合バイト EBCDIC	韓国語 (1364 のサブセット)
935	混合バイト EBCDIC	中国語 (簡体字) (1388 のサブセット)
937	混合バイト EBCDIC	中国語 (繁体字)
939	混合バイト EBCDIC	日本語 (5035 のサブセット)
1025	単一バイト EBCDIC	キリル文字
1026	単一バイト EBCDIC	トルコ語
1027	単一バイト EBCDIC	日本語英数小文字 (単一バイトのみ)
1046	ASCII/ISO/Windows (両方向)	Windows アラビア語 ST5
1089	ASCII/ISO/Windows (両方向)	ISO 8859-6 (アラビア語) ST5
1112	単一バイト EBCDIC	バルト語 (多国語)
1122	単一バイト EBCDIC	エストニア語
1123	単一バイト EBCDIC	ウクライナ
1125	ASCII/ISO/Windows	ウクライナ
1129	ASCII/ISO/Windows	ベトナム語
1130	単一バイト EBCDIC	ベトナム語
1131	ASCII/ISO/Windows	ベラルーシ
1132	単一バイト EBCDIC	ラオ語
1140	単一バイト EBCDIC	米国およびその他 (ユーロ通貨記号サポート)
1141	単一バイト EBCDIC	オーストリア、ドイツ (ユーロ通貨記号サポート)
1142	単一バイト EBCDIC	デンマーク、ノルウェー (ユーロ通貨記号サポート)

1143	単一バイト EBCDIC	フィンランド、スウェーデン (ユーロ通貨記号サポート)
1144	単一バイト EBCDIC	イタリア (ユーロ通貨記号サポート)
1145	単一バイト EBCDIC	スペイン、ラテンアメリカ (ユーロ通貨記号サポート)
1146	単一バイト EBCDIC	英国 (ユーロ通貨記号サポート)
1147	単一バイト EBCDIC	フランス (ユーロ通貨記号サポート)
1148	単一バイト EBCDIC	ラテン 1 (MNCS) (ユーロ通貨記号サポート)
1149	単一バイト EBCDIC	アイスランド (ユーロ通貨記号サポート)
1200	Unicode	Unicode UCS-2 (リトル・エンディアン)
1250	ASCII/ISO/Windows	Windows ラテン語 2
1251	ASCII/ISO/Windows	Windows キリル文字
1252	ASCII/ISO/Windows	Windows ラテン 1
1253	ASCII/ISO/Windows	Windows ギリシャ語
1254	ASCII/ISO/Windows	Windows トルコ
1255	ASCII/ISO/Windows (両方向)	Windows ヘブライ語 ST5
1256	ASCII/ISO/Windows (両方向)	Windows アラビア語 ST5
1257	ASCII/ISO/Windows	Windows バルト語
1258	ASCII/ISO/Windows	Windows ベトナム語
1364	混合バイト EBCDIC	日本語
1388	混合バイト EBCDIC	中国語 (簡体字)
1399	混合バイト EBCDIC	日本語 (V4R5 以上の場合)
4396	2 バイト EBCDIC	日本語 (300 のサブセット)
4930	2 バイト EBCDIC	韓国語
4931	2 バイト EBCDIC	中国語 (繁体字) (835 のサブセット)
4933	2 バイト EBCDIC	中国語 (簡体字) GBK 図形
4948	ASCII/ISO/Windows	ラテン語 2 (852 のサブセット)
4951	ASCII/ISO/Windows	キリル文字 (855 のサブセット)
5026	混合バイト EBCDIC	日本語
5035	混合バイト EBCDIC	日本語
5123	単一バイト EBCDIC	日本語 (単一バイトのみ、ユーロ通貨記号サポート)
5351	ASCII/ISO/Windows (両方向)	Windows ヘブライ語 (ユーロ通貨記号サポート) ST5

8492	2 バイト EBCDIC	日本語 (300 のサブセット)
8612	単一バイト EBCDIC	アラビア語 EBCDIC ST5
9026	2 バイト EBCDIC	韓国語 (834 のサブセット)
9029	2 バイト EBCDIC	中国語 (簡体字) (4933 のサブセット)
9066	ASCII/ISO/Windows	タイ語 (拡張 SBCS)
12588	2 バイト EBCDIC	日本語 (300 のサブセット)
13122	2 バイト EBCDIC	韓国語 (834 のサブセット)
16684	2 バイト EBCDIC	日本語 (V4R5 で使用可能)
17218	2 バイト EBCDIC	韓国語 (834 のサブセット)
12708	単一バイト EBCDIC	アラビア語 EBCDIC ST7
13488	Unicode	Unicode UCS-2 (ビッグ・エンディアン)
28709	単一バイト EBCDIC	中国語 (繁体字) (単一バイトのみ)
61952	Unicode	iSeries および AS/400e の Unicode (主に IFS で使用)
62211	単一バイト EBCDIC	ヘブライ語 EBCDIC ST5
62224	単一バイト EBCDIC	アラビア語 EBCDIC ST6
62235	単一バイト EBCDIC	ヘブライ語 EBCDIC ST6
62245	単一バイト EBCDIC	ヘブライ語 EBCDIC ST10

CommandPrompter クラス

CommandPrompter クラスは、指定されたコマンドのパラメーターを求めるプロンプトを出します。 CommandPrompter は、iSeries CL コマンド・プロンプト (F4 を押す) と似た機能、およびマネージメント・セントラルのコマンド・プロンプトと同じ機能を提供します。

CommandPrompter を使用するには、サーバーは OS/400 V4R4 以降のバージョンを実行している必要があります。詳細は、iSeries Navigator Information APARs を参照し、グラフィカル・コマンド・プロンプター・サポート用の必要な修正をご覧ください。

また CommandPrompter を使用するには、CLASSPATH に以下の jar ファイルがなければなりません。

- jt400.jar
- jui400.jar
- util400.jar
- x4j400.jar
- jhall.jar

jhall.jar 以外のすべての jar ファイルは、Toolbox for Java に組み込まれています。 jar ファイルの詳細は、JAR ファイルを参照してください。 jhall.jar のダウンロードの詳細は、 Sun JavaHelp(TM) Web サイトを参照してください。

CommandPrompter オブジェクトを構成するには、プロンプターを立ち上げる親フレーム用のパラメーター、プロンプト指示されるコマンドの AS400 オブジェクト、およびコマンド・ストリングをオブジェクトに渡します。コマンド・ストリングは、コマンド名、絶対コマンド・ストリング、または crt* のような部分コマンド名にできます。

CommandPrompter 表示は、ユーザーが親フレームに戻る前にクローズしなければならない形式指定ダイアログです。 CommandPrompter はプロンプトの際に生じるすべてのエラーを処理します。

例: CommandCall および AS400Message クラスと共に CommandPrompter を使用して、コマンド用のプロンプトを出して実行する «

RunJavaApplication

RunJavaApplication クラスと VRunJavaApplication クラスは、 iSeries JVM で Java プログラムを実行するためのユーティリティーです。 Java プログラムから呼び出す JavaApplicationCall クラスや VJavaApplicationCall クラスとは異なり、 RunJavaApplication と VRunJavaApplication は完全なプログラムです。

RunJavaApplication クラスは、コマンド行ユーティリティーです。 これを使用すると、Java プログラムの環境 (たとえば、CLASSPATH およびプロパティーなど) を設定できます。 Java プログラムの名前とこのパラメーターを指定してから、Java プログラムを開始します。 いったん Java プログラムを開始すれば、入力を Java プログラムに送信できます。 Java プログラムは、標準入力を介してこの入力を受信します。 また、標準出力および標準エラーに出力を書き込みます。

VRunJavaApplication ユーティリティーにも、同じ機能があります。 異なる点は、 VJavaApplicationCall はグラフィカル・ユーザー・インターフェースを使用し、 JavaApplicationCall はコマンド行インターフェースであるということです。

JPing

JPing クラスは、 サーバーに照会を出して、実行中のサービスとサービスを提供しているポートが何であるかを確認するためのコマンド入力行ユーティリティーです。 Java アプリケーション内からサーバーを照会するには、 AS400JPing クラスを使用します。

Java アプリケーション内からの JPing の使用に関する詳細は、 JPing javadoc を参照してください。

JPing をコマンド入力行から呼び出すには、次の構文を使用します。

```
java utilities.JPing System [options]
```

ここで:

- System = 照会する iSeries サーバー
- [options] = 使用可能な 1 つ以上のオプション

オプション

次の 1 つまたは複数のオプションを使用することができます。 略語のあるオプションの場合、その略語を括弧内にリストします。

-help (-h または -?)

ヘルプ・テキストを表示します。

-service *OS/400_Service* (-s *OS/400_Service*)

1 つの特定のサービスを指定して PING します。 デフォルトのアクションでは、すべてのサービスを PING します。 このオプションを使用して、as-file、 as-netprt、as-rmtcmd、as-dtaq、as-database、as-ddm、as-central、および as-signon のサービスのうちの 1 つを指定できます。

-ssl

ssl ポートを PING するかどうかを指定します。 デフォルトのアクションでは、ssl ポートを PING しません。

-timeout (-t)

タイムアウト期間をミリ秒単位で指定します。 デフォルト設定は 20000、つまり 20 秒です。

例: コマンド入力行から JPing を使用する

たとえば、ssl ポートを含む as-dtaq サービスに 5 秒のタイムアウトを設定して PING するには、 次のコマンドを使用します。

```
java utilities.JPing myServer -s as-dtaq -ssl -t 5000
```

Vaccess クラス

IBM Toolbox for Java には、 vaccess パッケージ内に グラフィカル・ユーザー・インターフェース (GUI) クラスのセットが備えられています。 このクラスは、アクセス・クラスを使って、データを取り出したり、データをユーザーに対して提示します。

IBM Toolbox for Java の vaccess クラスを使う Java プログラムでは、Swing 1.1 が必要です。 Swing 1.1 は、 Java 2 を実行するか、 Swing 1.1 を Sun Microsystems, Inc. からダウンロードすることによって入手できます。 以前の IBM Toolbox for Java では Swing 1.0.3 が必要でしたが、 V4R5 からは Swing 1.1 がサポートされています。 Swing 1.1 への移行において、一部のプログラミング変更が行われました。 そのため、ユーザーもプログラミング変更を行わなければならない可能性があります。 Swing についての詳細は、Sun Microsystems, Inc. JFC ページを参照してください。

IBM Toolbox for Java の GUI クラス、アクセス・クラス、 および Java Swing の関係の詳細については、Vaccess クラスの図を参照してください。

iSeries データを表示するには、 AS400 ペイン・クラスを使います。

API を使うと、次の iSeries リソースおよびツールにアクセスすることができます。

- コマンド呼び出し
- データ待ち行列
- エラー・イベント*
- 統合ファイル・システム
- JavaApplicationCall
- JDBC
- ジョブ*
- メッセージ*
- 許可
- スプール・ファイル・ビューアーを含む 印刷*
- ProgramCall および ProgramParameter
- レコード・レベルでのアクセス
- リソース・リスト
- システム状況
- システム値

- ユーザーおよびグループ

注: AS400 ペインを他の vaccess クラス (上記のアスタリスク付きの項目) と一緒に使うと、 iSeries リソースを表示したり、操作できるようになります。

IBM Toolbox for Java の GUI コンポーネントを使ったプログラミングの場合、エラー・イベント・クラスを使って、エラー・イベントの処理やユーザーに対する報告を行います。

iSeries データへのアクセスに関する詳細は、アクセス・クラスを参照してください。

Vaccess クラス

IBM Toolbox for Java には、 Vaccess パッケージの中にグラフィカル・ユーザー・インターフェース (GUI) クラスが備えられており、 サーバー・データを検索や表示したり、場合によっては、操作をすることも可能です。 これらのクラスは、Java Swing 1.1 フレームワークを使用します。 図 1 は、これらのクラスの相互関係を示します。

図 1: Vaccess クラス

AS400Panes

AS400Panes は、 GUI で 1 つ以上のサーバー・リソースを表示したり操作したりするための vaccess パッケージ内のコンポーネントです。 それぞれのサーバー・リソースがどのように機能するかは、リソースのタイプによって異なります。

どのペインであっても Java Component クラスを拡張します。 したがって、これらのペインはどの AWT Frame、Window、または Container にも追加できます。

以下の AS400Panes を使用できます。

- AS400DetailsPane は、 テーブル内にあるサーバー・リソースのリストを表します。 テーブル内の各行は、単一のリソースについての詳細情報を表示します。 テーブルを使用して、1 つ以上のリソースを選択できます。
- AS400ExplorerPane は、 AS400TreePane および AS400DetailsPane との組み合わせで使用され、ツリー内で選択されたリソースを詳しく表示します。
- AS400JDBCDataSourcePane は、 AS400JDBCDataSource オブジェクトのプロパティー値を表します。
- AS400ListPane は、サーバー・リソースのリストを表し、 1 つ以上のリソースを選択できるようにします。
- AS400TreePane は、サーバー・リソースのツリー階層を表し、 1 つ以上のリソースを選択できるようにします。

サーバー・リソース

サーバー・リソースは、 アイコンやテキストを持ったグラフィカル・ユーザー・インターフェースとして表されます。 サーバー・リソースは、階層関係で定義されます。 階層関係の中では、各リソースは 1 つの親とゼロ以上の子を持ちます。 これらは事前定義された関係であり、この関係によって、 AS400Pane にどのリソースを表示するかを指定します。 たとえば、VJobList はゼロ以上の VJob の親になり、 この階層関係は AS400Pane にグラフィック表示されます。

IBM Toolbox for Java では、以下のサーバー・リソースにアクセスできます。

- VIFSDirectory は、 統合ファイル・システム内のディレクトリーを表します
- VJob および VJobList は、 それぞれジョブおよびジョブのリストを表します
- VMessageList および VMessageQueue は、 それぞれ CommandCall または ProgramCall から戻されたメッセージのリスト、 およびメッセージ・キューを表します
- VPrinter、VPrinters、および VPrinterOutput は、 それぞれプリンター、プリンターのリスト、およびスプール・ファイルのリストを表します
- VUserList は、ユーザーのリストを表します

すべてのリソースは、 VNode インターフェースのインプリメンテーションです。

ルートの設定

AS400Pane がどのサーバー・リソースを表すかを指定するには、 コンストラクターまたは setRoot() メソッドを使用してルートを設定します。 ルートにはトップレベルのオブジェクトが定義されます。 ルートはペインに基づいて別々に使用されます。

- AS400ListPane は、ルートの子すべてをリストにして表します
- AS400DetailsPane は、ルートの子すべてをテーブルにして表します
- AS400TreePane は、そのルートをツリーのルートとして使用します

- AS400ExplorerPane は、そのルートをツリーのルートとして使用します

ペインとルートをどのように組み合わせるかは自由です。

以下の例では、AS400DetailsPane を作成して、 システムに定義されているユーザーのリストを表示します。

```
                    // Create the server resource
                    // representing a list of users.
                    // Assume that "system" is an AS400
                    // object created and initialized
                    // elsewhere.
    VUserList userList = new VUserList (system);

                    // Create the AS400DetailsPane object
                    // and set its root to be the user
                    // list.
    AS400DetailsPane detailsPane = new AS400DetailsPane ();
    detailsPane.setRoot (userList);

                    // Add the details pane to a frame.
                    // Assume that "frame" is a JFrame
                    // created elsewhere.
    frame.getContentPane ().add (detailsPane);
```

コンテンツのロード

AS400Pane オブジェクトおよびサーバー・リソース・オブジェクトは、 作成時にはデフォルト状態に初期設定されています。 ペインのコンテンツを構成する関連情報は、作成時にはロードされません。

ペインのコンテンツをロードするためには、 アプリケーションが load() メソッドを明示的に呼び出す必要があります。 ほとんどの場合は、これによりサーバーとの通信が開始され、 関連情報が収集されます。 この情報を収集するのに多少の時間がかかる場合があるので、 アプリケーションはいつ収集を行うかを正確に制御できます。 たとえば、以下のようなことができます。

- ペインをフレームに追加する前に、コンテンツをロードすることができます。 すべての情報がロードされるまでは、フレームは表示しません。
- ペインをフレームに追加し、そのフレームを表示した後で、内容をロードすることもできます。 フレームは表示されますが、 最初はその中に情報は含まれていません。 "待機カーソル" が表示され、 ロードが完了した情報からフレームの中に含まれていきます。

以下の例では、フレームに追加する前に、 詳細ペインの内容をロードします。

```
                    // Load the contents of the details
                    // pane. Assume that the detailsPane
                    // was created and initialized
                    // elsewhere.
    detailsPane.load ();

                    // Add the details pane to a frame.
                    // Assume that "frame" is a JFrame
                    // created elsewhere.
    frame.getContentPane ().add (detailsPane);
```

アクションおよびプロパティー・ペイン

実行時には、 ユーザーはどのサーバー・リソース上でもポップアップ・メニューから選択を行うことができます。 このポップアップ・メニューは、 そのリソースで実行可能な関連アクションのリストを表示します。 ポップアップ・メニューからユーザーがアクションを選択すると、 そのアクションが実行されます。 定義されているアクションは、リソースごとに異なります。

場合によっては、 ユーザーがプロパティー・ペインを表示できるようにするための項目がポップアップ・メニューに表されることもあります。 このプロパティー・ペインは、 リソースに関する詳細情報を示します。 ユーザーにこれらの詳細の変更を許可することもできます。

ペインに対して setAllowActions() メソッドを使用することにより、 どのアクションおよびプロパティー・ペインを使用可能にするかをアプリケーションは制御できます。

モデル

AS400Panes は、モデル・ビュー・コントローラー・パラダイムを使用して実装されます。 このパラダイムでは、データとユーザー・インターフェースが別々のクラスに分離されます。 AS400Panes は、Toolbox for Java モデルを Java GUI コンポーネントと統合します。 このモデルはサーバー・リソースを管理し、Vaccess コンポーネントは、 サーバー・リソースをグラフィカルに表示し、ユーザーとの対話を処理します。

AS400Panes は機能性が高く、大抵の要件は満たすことができますが、 JFC コンポーネントをそれ以上に制御することをアプリケーションが必要とする場合は、 サーバー・モデルに直接アクセスすることにより、 別の Vaccess コンポーネントとの統合をカスタマイズすることができます。

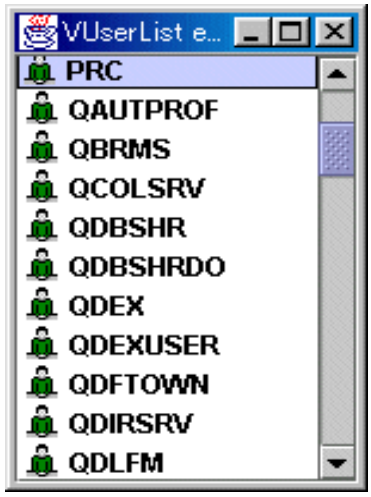
以下のモデルを使用できます。

- AS400ListModel は、JFC ListModel インターフェースをサーバー・リソースのリストとして実装します。 このモデルは JFC JList オブジェクトと一緒に使用することもできます。
- AS400DetailsModel は、 JFC TableModel インターフェースをサーバー・リソースのテーブルとして実装します。 テーブル内の各行には、単一のリソースについての詳細情報が含められます。 このモデルは JFC JTable オブジェクトと一緒に使用することもできます。
- AS400TreeModel は、JFC TreeModel インターフェースをサーバー・リソースのツリー階層として実装します。 このモデルは JFC JTree オブジェクトと一緒に使用することもできます。

例

- AS400ListPane を VUserList オブジェクトと一緒に使用することにより、 システム上のユーザーのリストを表します。図 1 は、完成後のプロダクトを示します。

  図 1: AS400ListPane を VUserList オブジェクトと共に使用する

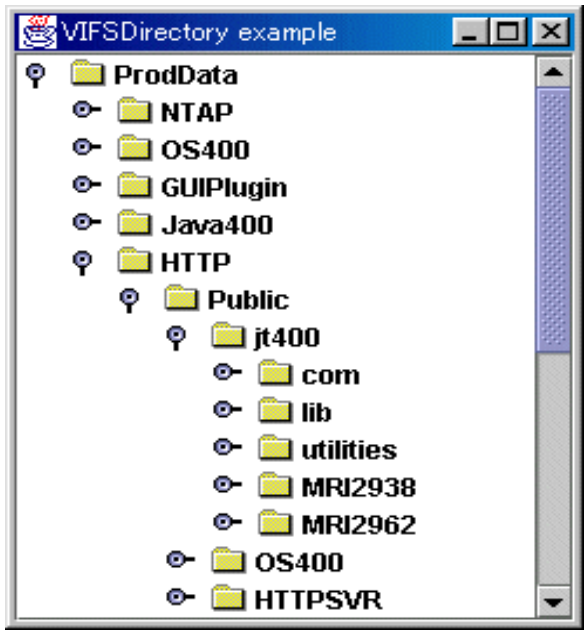


- AS400DetailsPane を VMessageList オブジェクトと一緒に使用することにより、 コマンド呼び出しが生成するメッセージのリストを表します。 図 2 は、完成後のプロダクトを示します。

  図 2: AS400DetailsPane を VMessageList オブジェクトと共に使用する

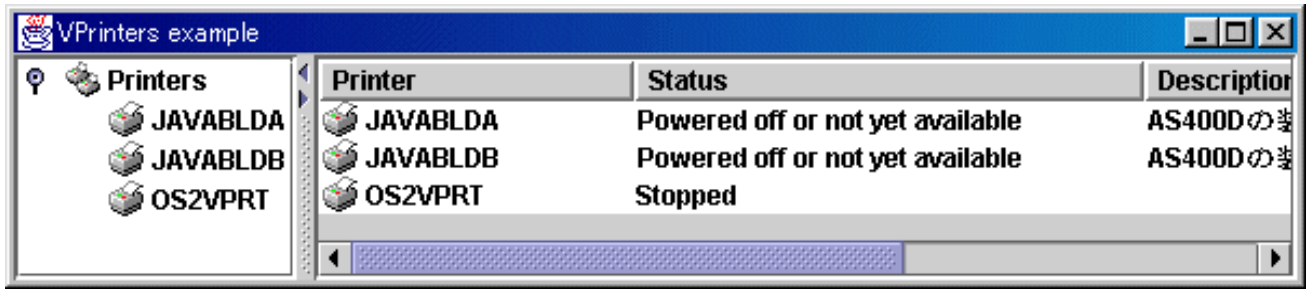
- AS400TreePane を VIFSDirectory オブジェクトと一緒に使用することにより、 統合ファイル・システムのディレクトリー階層を表します。 図 3 は、完成後のプロダクトを示します。

  図 3: AS400TreePane を VIFSDirectory オブジェクトと共に使用する



- AS400ExplorerPane を VPrinters オブジェクトと一緒に使用することにより、 印刷リソースを表します。 図 4 は、完成後のプロダクトを示します。

  図 4: AS400ExplorerPane を VPrinters オブジェクトと共に使用する


VPrinters example
Printers
JAVABLDA
JAVABLDB
OS2VPRT
Printer
Status
Description
JAVABLDA
Powered off or not yet available
AS400Dの
JAVABLDB
Powered off or not yet available
AS400Dの
OS2VPRT
Stopped

コマンド呼び出し

コマンド呼び出し Vaccess (GUI) コンポーネントを使用すれば、 Java プログラムは非対話式サーバー・コマンドを呼び出すボタンやメニュー項目を表示することができます。

CommandCallButton オブジェクトは、 押されるとサーバー・コマンドを呼び出すボタンを表します。 CommandCallButton クラスは、Java Foundation Classes (JFC) JButton クラスを拡張して、 すべてのボタンが一貫性のある外観や動作を持つようにします。

同様に、 CommandCallMenuItem オブジェクトは、 選択されるとサーバー・コマンドを呼び出すメニュー項目を表します。 CommandCallMenuItem クラスも、JFC JMenuItem クラスを拡張して、すべてのメニュー項目が一貫性のある外観や動作を持つようにします。

コマンド呼び出しグラフィカル・ユーザー・インターフェース・コンポーネントを使用するためには、 システム・プロパティーとコマンド・プロパティーの両方を設定する必要があります。 これらのプロパティーを設定するには、コンストラクターを使用したり、 setSystem() メソッドや setCommand() メソッドを使用したりします。

以下の例では、CommandCallButton を作成します。 実行時にボタンを押すと、"FRED" というライブラリーが作成されます。

```
                    // Create the CommandCallButton
                    // object. Assume that "system" is
                    // an AS400 object created and
                    // initialized elsewhere.  The button
                    // text says "Press Me", and there is
                    // no icon.
  CommandCallButton button = new CommandCallButton ("Press Me", null, system);

                    // Set the command that the button will run.
  button.setCommand ("CRTLIB FRED");

                    // Add the button to a frame. Assume
                    // that "frame" is a JFrame created
                    // elsewhere.
  frame.getContentPane ().add (button);
```

サーバー・コマンドが実行されると、 サーバー・メッセージが戻される場合があります。 サーバー・コマンドがいつ実行されたかを検出するには、 addActionCompletedListener() メソッドを使用して ActionCompletedListener をボタンまたはメニュー項目に追加します。 コマンドを実行すると、それらのリスナーすべてに ActionCompletedEvent が向けられます。 リスナーは getMessageList() メソッドを使用して、 コマンドが生成したどのサーバー・メッセージでも検索することができます。

以下の例では、 コマンドが生成したすべてのサーバー・メッセージを処理する ActionCompletedListener を追加します。

```
                    // Add an ActionCompletedListener that
                    // is implemented using an anonymous
                    // inner class. This is a convenient
                    // way to specify simple event
                    // listeners.
```

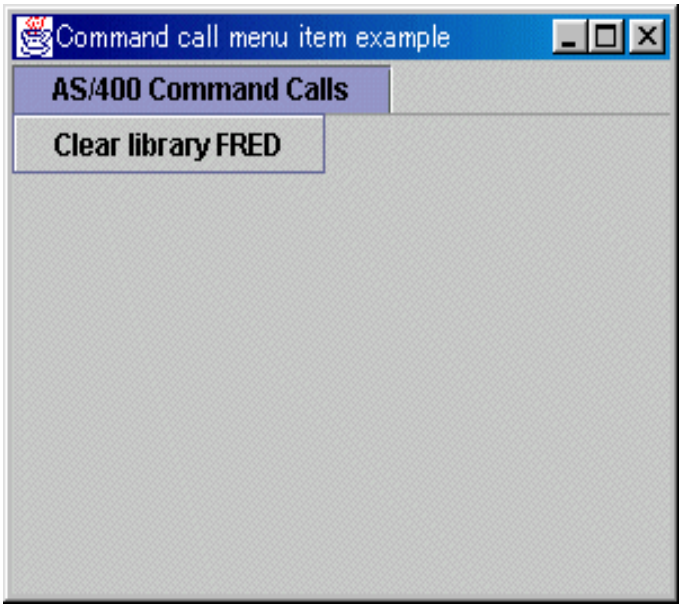
```
button.addActionCompletedListener (new ActionCompletedListener ()
{
     public void actionCompleted (ActionCompletedEvent event)
     {
               // Cast the source of the event to a
               // CommandCallButton.
          CommandCallButton sourceButton = (CommandCallButton) event.getSource ();

               // Get the list of server messages
               // that the command generated.
          AS400Message[] messageList = sourceButton.getMessageList ();

               // ... Process the message list.
     }
});
```

例

以下の例は、 アプリケーションで CommandCallMenuItem を使用する方法を示します。

図 1 は、CommandCall グラフィカル・ユーザー・インターフェース・コンポーネントを示します。

図 1: CommandCall GUI コンポーネント



データ待ち行列

データ待ち行列グラフィカル・コンポーネントを使用すれば、 Java プログラムは Java Foundation Classes (JFC) グラフィカル・テキスト・コンポーネントをどれでも使用して、 サーバー・データ待ち行列を読み書きできます。

DataQueueDocument クラスおよび KeyedDataQueueDocument クラスは、JFC 文書インターフェースのインプリメンテーションです。 これらのクラスは、直接、JFC グラフィカル・テキスト・コンポーネントと一緒に使用できます。 単一行フィールド (JTextField) や複数行テキスト・エリア (JTextArea) など、 いくつかのテキスト・コンポーネントが JFC で使用できます。

データ待ち行列文書は、テキスト・コンポーネントの中身をサーバー・データ待ち行列に関連付けます。 (テキスト・コンポーネントは、 ユーザーが任意で編集できるテキストを表示するためのグラフィカル・コンポーネントです。) Java プログラムは、 テキスト・コンポーネントとデータ待ち行列の間でいつでも読み書きすることができます。 順次データ待ち行列の場合は DataQueueDocument を使用し、 キー順データ待ち行列の場合は KeyedDataQueueDocument を使用します。

DataQueueDocument を使用するためには、 システム・プロパティーとパス・プロパティーの両方を設定する必要があります。 これらのプロパティーを設定するには、 コンストラクターを使用したり、 setSystem() メソッドや setPath() メソッドを使用したりします。 その後、通常はテキスト・コンポーネントのコンストラクターや setDocument() メソッドを使用して、 DataQueueDocument オブジェクトがテキスト・コンポーネントに「接続」されます。 KeyedDataQueueDocuments も同様に機能します。

以下の例では、中身がデータ待ち行列に関連付けられている DataQueueDocument を作成します。

```
                        // Create the DataQueueDocument
                        // object. Assume that "system" is
                        // an AS400 object created and
                        // initialized elsewhere.
     DataQueueDocument dqDocument = new DataQueueDocument (system,
"/QSYS.LIB/MYLIB.LIB/MYQUEUE.DTAQ");

                        // Create a text area to present the
                        // document.
     JTextArea textArea = new JTextArea (dqDocument);

                        // Add the text area to a frame.
```

```
                        // Assume that "frame" is a JFrame
                        // created elsewhere.
    frame.getContentPane ().add (textArea);
```

最初は、テキスト・コンポーネントの中身は空です。 read() や peek() などが使用されて、 待ち行列の次の項目で埋められていきます。 Write() は、 テキスト・コンポーネントの中身をデータ待ち行列に書き込むために使用されます。 これらの文書が処理するのは、 String データ待ち行列項目だけであることに注意してください。

例

以下の例は、アプリケーションでの DataQueueDocument の使用例です。

図 1 は、JTextField で使用される DataQueueDocument グラフィカル・ユーザー・インターフェース・コンポーネントを示します。 ボタンが GUI インターフェースとして追加されて、 ユーザーがテスト・フィールドの中身をデータ待ち行列に書き込めるようになります。

図 1: DataQueueDocument GUI コンポーネント

エラー・イベント

多くの場合、IBM Toolbox for Java のグラフィカル・ユーザー・インターフェース (GUI) コンポーネントは、 例外を発行する代わりに、エラー・イベントを出します。

エラー・イベントとは、内部コンポーネントから出された例外を包むラッパーのことです。

特定のユーザー・インターフェース・コンポーネントから出された、すべてのエラー・イベントを扱うエラー・リスナーを提供することができます。例外が発行されるごとにリスナーが呼び出されます。これは、該当するエラー・レポートを示すことができます。デフォルトでは、エラー・イベントが発行されても、アクションはとられません。

IBM Toolbox for Java には、 ErrorDialogAdapter という名前の GUI コンポーネントが備わっています。これは、エラー・イベントが発行されるたびに、自動的にユーザーに対してダイアログを表示します。

例

以下の例は、エラーの処理方法および単純なエラー・リスナーの定義方法を示しています。

例: ダイアログを表示してエラー・イベントを処理する

以下の例は、ダイアログを表示してエラー・イベントを処理する方法を示しています。

```
                    // ... all the setup work to lay out
                    // a graphical user interface
                    // component is done. Now add an
                    // ErrorDialogAdapter as a listener
                    // to the component. This will report
                    // all error events fired by that
                    // component through displaying a
                    // dialog.
ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (parentFrame);
component.addErrorListener (errorHandler);
```

これとは別の方法でエラーを処理するためのカスタム・エラー・リスナーを作成することもできます。そのためには、 ErrorListener インターフェースを使用します。

例: エラー・リスナーを定義する

以下の例は、System.out にだけエラーを出力する単純なエラー・リスナーの定義方法を示しています。

```
class MyErrorHandler
implements ErrorListener
{
                    // This method is invoked whenever
                    // an error event is fired.
          public void errorOccurred(ErrorEvent event)
          {
               Exception e = event.getException ();
               System.out.println ("Error: " + e.getMessage ());
          }
}
```

例: エラー・リスナーを使用してエラー・イベントを処理する

以下の例は、次のカスタム・ハンドラーを使って、 GUI コンポーネントのエラー・イベントを処理する方法を示しています。

```
MyErrorHandler errorHandler = new MyErrorHandler ();
component.addErrorListener (errorHandler);
```

統合ファイル・システム

統合ファイル・システムのグラフィカル・ユーザー・インターフェース・コンポーネントを使用すると、Java プログラムが、 サーバー上の統合ファイル・システム内のディレクトリーおよびファイルを、GUI で表示できるようになります。

以下のコンポーネントを使用できます。

- IFSFileDialog は、 ユーザーがディレクトリー階層内を移動して、 ディレクトリーとファイルを選択できるようにするダイアログを提供します。
- VIFSDirectory は、AS400Panesでの使用を目的としている、 統合ファイル・システム内のディレクトリーを表すリソースです。
- IFSTextFileDocument は、 特定の Java Foundation Classes (JFC) グラフィカル・テキスト・コンポーネントで使用するためのテキスト・ファイルを表します。

統合ファイル・システム・グラフィカル・ユーザー・インターフェース・コンポーネントを使用するためには、 システム・プロパティーと、パスまたはディレクトリー・プロパティーの両方を設定しなければなりません。 これらのプロパティーを設定するには、 コンストラクターを使用したり、 setDirectory() メソッド (IFSFileDialog の場合) を使用したり、 setSystem() および setPath() メソッド (VIFSDirectory および IFSTextFileDocument の場合) を使用します。

"/QSYS.LIB" 以外のファイルへのパスを設定するようお勧めします。 その理由は、このディレクトリーのサイズは大きい場合が多く、ダウンロードに時間がかかるからです。

ファイル・ダイアログ

IFSFileDialog クラスは、 ユーザーがサーバー上の統合ファイルシステムのディレクトリーを走査してファイルを選択するために使用できるダイアログです。 呼び出し元はダイアログのボタンにテキストを設定できます。 さらに、呼び出し元は FileFilter オブジェクトも使用できます。 このオブジェクトはユーザーが選択できるファイルを制限するためのものです。

ユーザーがダイアログにあるファイルを選択する場合、 getFileName() メソッドを使用して、選択したファイルの名前を知ることができます。 getAbsolutePath() メソッドを使用すれば、 選択したファイルの完全パス名を知ることができます。

以下の例では、 2 つのファイル・フィルターを持った統合ファイル・システム・ダイアログをセットアップします。

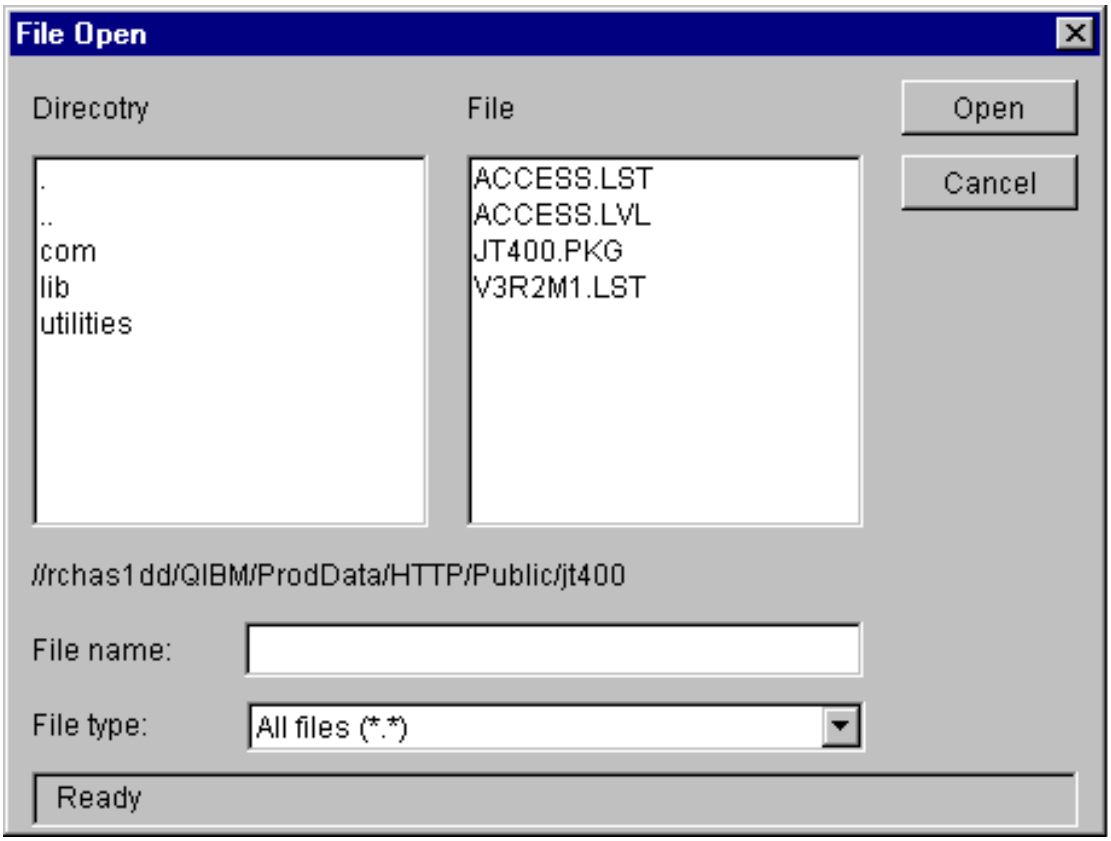
```
                    // Create a IFSFileDialog object
                    // setting the text of the title bar.
                    // Assume that "system" is an AS400
                    // object and "frame" is a JFrame
                     // created and initialized elsewhere.
IFSFileDialog dialog = new IFSFileDialog (frame, "Select a file", system);

                    // Set a list of filters for the dialog.
                    // The first filter will be used
                    // when the dialog is first displayed.
FileFilter[] filterList = {new FileFilter ("All files (*.*)", "*.*"),
                           new FileFilter ("HTML files (*.HTML", "*.HTM")};
         // Then, set the filters in the dialog.
dialog.setFileFilter (filterList, 0);

                    // Set the text on the buttons.
dialog.setOkButtonText ("Open");
dialog.setCancelButtonText ("Cancel");

                    // Show the dialog. If the user
                    // selected a file by pressing the
                    // "Open" button, then print the path
                    // name of the selected file.
if (dialog.showDialog () == IFSFileDialog.OK)
     System.out.println (dialog.getAbsolutePath ());
```

例

IFSFileDialog を表示し、 選択されたものがあればそれを印刷します。

図 1 は、IFSFileDialog グラフィカル・ユーザー・インターフェース・コンポーネントを示します。

図 1: IFSFileDialog GUI コンポーネント



AS400Panes のディレクトリー

AS400Panes は、 1 つ以上のサーバー・リソースを表示したり操作したりするための GUI コンポーネントです。 VIFSDirectory オブジェクトは、 AS400Panes での使用を目的とする、統合ファイル・システムでのディレクトリーを表すリソースです。 AS400Pane と VIFSDirectory オブジェクトを一緒に使用すると、統合ファイル・システムのビューを複数表示して、 ユーザーがディレクトリーやファイルをナビゲート、操作、および選択するのに役立てることができます。

VIFSDirectory を使用するためには、 システム・プロパティーとパス・プロパティーの両方を設定する必要があります。 これらのプロパティーを設定するには、コンストラクターを使用したり、 setSystem() メソッドや setPath() メソッドを使用したりします。 その後、AS400Pane のコンストラクターまたは setRoot() メソッドを使用し、 ルートとして VIFSDirectory オブジェクトを AS400Pane に接続します。

VIFSDirectory には、 AS400Panes に表示されているディレクトリーやファイルのセットを定義するのに役立つ他のプロパティーも含まれています。 たとえば、setInclude() を使用すれば、表示内容をディレクトリーとファイルのいずれかに限ったり、 あるいは両方とも表示したりするよう指定できます。 setPattern() を使用すれば、一致すべきファイル名のパターンを指定して、 表示する項目をフィルターに掛けることができます。 パターンには、"*" や "?" などのワイルドカード文字を使用できます。 同様に、setFilter() を使用すれば、 IFSFileFilter オブジェクトをフィルターに掛けることができます。

AS400Pane オブジェクトおよび VIFSDirectory オブジェクトは、 作成時にはデフォルト状態に初期設定されています。 ルート・ディレクトリーの内容を構成するサブディレクトリーおよびファイルは、 その時点ではまだロードされていません。 ルート・ディレクトリーの内容をロードするためには、 呼び出し元がどちらかのオブジェクトに対して load() メソッドを明示的に呼び出すことにより、 ディレクトリーの内容を収集するサーバー・システムへの通信を開始しなければなりません。

実行時には、ユーザーはどのディレクトリーまたはファイルに対しても、 それを右マウス・ボタン・クリックしてコンテキスト・メニューを表示させることにより、アクションを実行できます。 ディレクトリー・コンテキスト・メニューには以下の項目を組み込むことができます。

- ファイルの作成 - ディレクトリー内にファイルを作成できます。 ファイルにはデフォルトの名前が与えられます
- ディレクトリーの作成 - デフォルト名のサブディレクトリーを作成できます
- 名前変更 - ディレクトリーの名前を変更できます
- 削除 - ディレクトリーを削除できます
- プロパティー - 場所、ファイルとサブディレクトリーの数、 変更日付などのプロパティーを表示できます

ファイル・コンテキスト・メニューには以下の項目を組み込むことができます。

- 編集 - 別のウィンドウ内にあるテキスト・ファイルを編集できます
- 表示 - 別のウィンドウ内にあるテキスト・ファイルを表示できます

- 名前変更 - ファイルの名前を変更できます
- 削除 - ファイルを削除できます
- プロパティー - 場所、サイズ、変更日付、属性などのプロパティーを表示できます

ユーザーが読み書きできるディレクトリーおよびファイルは、 読み書きの許可を受けているディレクトリーおよびファイルだけです。 一方、呼び出し元では、 ペインに対して setAllowActions() メソッドを実行することにより、 ユーザーがアクションを実行できないようにすることができます。

以下の例では、VIFSDirectory を作成し、それを AS400ExplorerPane に表します。

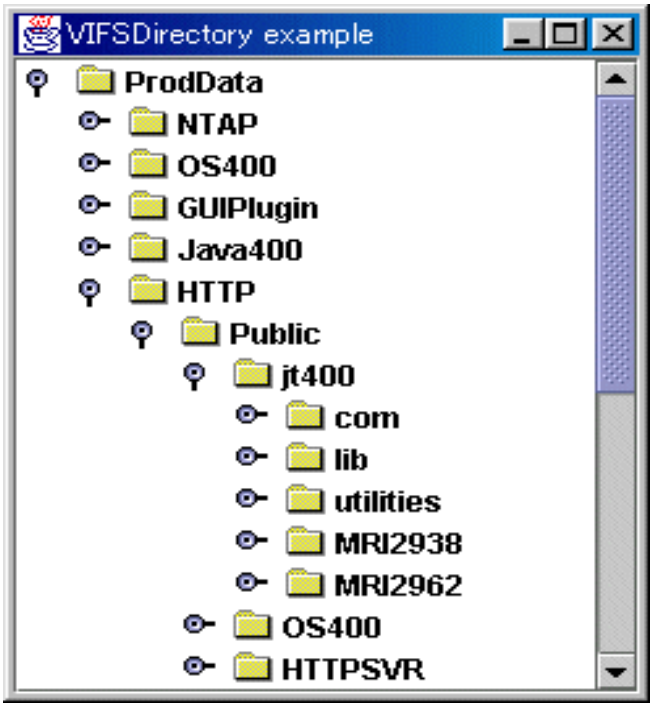
```
                    // Create the VIFSDirectory object.
                    // Assume that "system" in an AS400
                    // object created and initialized
                    // elsewhere.
VIFSDirectory root = new VIFSDirectory (system, "/DirectoryA/DirectoryB");

                    // Create and load an AS400ExplorerPane object.
AS400ExplorerPane explorerPane = new AS400ExplorerPane (root);
explorerPane.load ();

                    // Add the explorer pane to a frame.
                    // Assume that "frame" is a JFrame
                    // created elsewhere.
frame.getContentPane ().add (explorerPane);
```

例

AS400TreePane を VIFSDirectory オブジェクトと一緒に使用することにより、 統合ファイル・システムのディレクトリー階層を表します。

図 1 は、VIFSDirectory グラフィカル・ユーザー・インターフェース・コンポーネントを示します。

図 1: VIFSDirectory GUI コンポーネント


VIFSDirectory example
ProdData
NTAP
OS400
GUIPlugin
Java400
HTTP
Public
jt400
com
lib
utilities
MRI2938
MRI2962
OS400
HTTPSVR

IFSTextFileDocument

テキスト・ファイル文書を使用すれば、 Java プログラムは任意の Java Foundation Classes (JFC) グラフィカル・テキスト・コンポーネントを使用して、 サーバー上の統合ファイル・システム内のテキスト・ファイルを編集したり表示したりできます。 (テキスト・コンポーネントは、 ユーザーが任意で編集できるテキストを表示するためのグラフィカル・コンポーネントです。)

IFSTextFileDocument クラスは、JFC 文書インターフェースのインプリメンテーションです。 このクラスは、直接、JFC グラフィカル・テキスト・コンポーネントと一緒に使用できます。 単一行フィールド (JTextField) や複数行テキスト・エリア (JTextArea) など、 いくつかのテキスト・コンポーネントが JFC で使用できます。

テキスト・ファイル文書は、テキスト・コンポーネントの中身をテキスト・ファイルに関連付けます。 Java プログラムは、 テキスト・コンポーネントとテキスト・ファイルの間でいつでもロードおよび保管を実行できます。

IFSTextFileDocument を使用するためには、 システム・プロパティーとパス・プロパティーの両方を設定する必要があります。 これらのプロパティーを設定するには、 コンストラクターを使用したり、 setSystem() メソッドや setPath() メソッドを使用したりします。 その後、通常はテキスト・コンポーネントのコンストラクターや setDocument() メソッドを使用して、 IFSTextFileDocument オブジェクトがテキスト・コンポーネントに「接続」されます。

最初は、テキスト・コンポーネントの中身は空です。 load() を使用して、テキスト・ファイルから中身をロードします。 save() は、テキスト・コンポーネントの中身をテキスト・ファイルに保管するために使用されます。

以下の例では、IFSTextFileDocument を作成およびロードします。

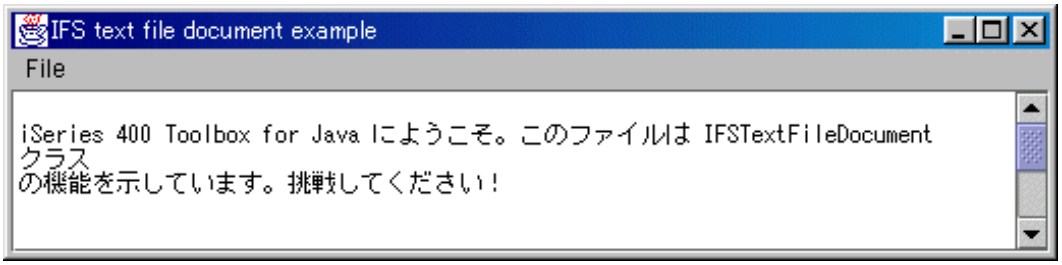
```
                    // Create and load the
                    // IFSTextFileDocument object. Assume
                    // that "system" is an AS400 object
                    // created and initialized elsewhere.
IFSTextFileDocument ifsDocument = new IFSTextFileDocument (system, "/DirectoryA/MyFile.txt");
ifsDocument.load ();

                    // Create a text area to present the
                    // document.
JTextArea textArea = new JTextArea (ifsDocument);

                    // Add the text area to a frame.
                    // Assume that "frame" is a JFrame
                    // created elsewhere.
frame.getContentPane ().add (textArea);
```

例

JTextPane の中に IFSTextFileDocument を表示します。

図 1 は、IFSTextFileDocument グラフィカル・ユーザー・インターフェース・コンポーネントを示します。

図 1: IFS テキスト・ファイル文書の例



VJavaApplicationCall クラス

VJavaApplicationCall クラスは、 グラフィカル・ユーザー・インターフェース (GUI) を使用して、 クライアントからサーバー上の Java アプリケーションを実行することを可能にします。

使用する GUI は、2 つのセクションを持つパネルです。 上部セクションは出力ウィンドウで、 Java プログラムが標準出力および標準エラーに書き込む出力が表示されます。 下部セクションは入力フィールドで、 Java 環境、パラメーターを指定して実行される Java プログラム、 および Java プログラムが標準入力から受け取る入力を入力します。 詳細については、Java コマンド・オプションを参照してください。

たとえば、このコードは、 Java プログラム用に以下のような GUI を作成します。

VJavaApplicationCall は、Java プログラムから呼び出すクラスです。 一方で、IBM Toolbox for Java では、 完全な Java アプリケーションであるユーティリティーも用意されています。 このユーティリティーを使ってワークステーションから Java プログラムを呼び出すことができます。 詳細については、RunJavaApplication クラスを参照してください。

JDBC クラス

JDBC グラフィカル・ユーザー・インターフェース・コンポーネントを使用すれば、Java プログラムは、 SQL (構造化照会言語) ステートメントや照会を使用してデータベースにアクセスするための、 さまざまなビューやコントロールを表示できます。

以下のコンポーネントを使用できます。

- SQLStatementButton および SQLStatementMenuItem はそれぞれ、 クリック時または選択時に SQL ステートメントを発行するボタンおよびメニュー項目です。
- SQLStatementDocument は、SQL ステートメントを発行するために、 任意の Java Foundation Classes (JFC) グラフィカル・テキスト・コンポーネントと一緒に使用できる文書です。
- SQLResultSetFormPane は、 SQL 照会の結果を一定の形式で表示します。
- SQLResultSetTablePane は、SQL 照会の結果をテーブルに表示します。
- SQLResultSetTableModel は、SQL 照会の結果をテーブルで管理します。
- SQLQueryBuilderPane は、 SQL 照会を動的に構築するための対話式ツールを表します。

JDBC グラフィカル・ユーザー・インターフェース・コンポーネントはすべて、 JDBC ドライバーを使用してデータベースと通信します。 この JDBC ドライバーを JDBC ドライバー・マネージャーに登録しなければ、 これらのコンポーネントは使用できません。 以下の例では、iSeries Toolbox for Java JDBC ドライバーを登録します。

```
                    // Register the JDBC driver.
DriverManager.registerDriver (new com.ibm.as400.access.AS400JDBCDriver ());
```

SQL 接続

SQLConnection オブジェクトは、JDBC を使用したデータベースへの接続を表します。 SQLConnection オブジェクトは、 JDBC グラフィカル・ユーザー・インターフェース・コンポーネントすべてで使用されます。

SQLConnection を使用するためには、 コンストラクターまたは setURL() を使用して、URL プロパティーを設定する必要があります。 これにより、 接続の対象となるデータベースが識別されます。 他のオプション・プロパティーを設定することもできます。

- setProperties() を使用すると、JDBC 接続プロパティーのセットを指定できます。
- setUserName() を使用すると、接続に関係するユーザー名を指定できます。
- setPassword() を使用すると、接続するためのパスワードを指定できます。

データベースへの実際の接続が確立されるのは、 SQLConnection オブジェクトが作成されるときではありません。 データベースへの実際の接続が確立されるのは、 getConnection() が呼び出されるときです。 このメソッドは、通常、 JDBC グラフィカル・ユーザー・インターフェース・コンポーネントが自動的に呼び出しますが、 接続が確立される時期を制御するために、いつでも手操作で呼び出せます。

以下の例では、 SQLConnection オブジェクトを作成および初期設定します。

```
                    // Create an SQLConnection object.
SQLConnection connection = new SQLConnection ();

                    // Set the URL and user name properties of the connection.
connection.setURL ("jdbc:as400://MySystem");
connection.setUserName ("Lisa");
```

SQLConnection オブジェクトは、 複数の JDBC グラフィカル・ユーザー・インターフェース・コンポーネントに使用できます。 それらのコンポーネントはすべて、同じ接続を使用するため、 パフォーマンスの向上やリソースの有効利用が得られます。 また、JDBC グラフィカル・ユーザー・インターフェース・コンポーネントごとに、 別々の SQL オブジェクトを使用することもできます。 場合によっては、別々の接続を使用することが必要になるため、 SQL ステートメントは別個のトランザクションで発行されます。

接続が不要になったら、 close() を使用して、SQLConnection オブジェクトをクローズします。 これにより、クライアントとサーバーの双方にある JDBC リソースが解放されます。

ボタンおよびメニュー項目

SQLStatementButton オブジェクトは、 押されると SQL (構造化照会言語) ステートメントを発行するボタンを表します。 SQLStatementButton クラスは、 Java Foundation Classes (JFC) JButton クラスを拡張して、 すべてのボタンが一貫性のある外観や動作を持つようにします。

同様に、 SQLStatementMenuItem オブジェクトは、 選択されると SQL ステートメントを発行するメニュー項目を表します。 SQLStatementMenuItem クラスも、 JFC JMenuItem クラスを拡張して、 すべてのメニュー項目が一貫性のある外観および動作を持つようにします。

これらのクラスのいずれかを使用するためには、 接続プロパティーと SQLStatement プロパティーの両方を設定する必要があります。 これらのプロパティーを設定するには、 コンストラクターを使用したり、 setConnection() メソッドや setSQLStatement() メソッドを使用します。

以下の例では、 SQLStatementButton を作成します。 実行時にこのボタンが押されると、 テーブル内のすべてのレコードが削除されます。

```
                    // Create an SQLStatementButton object.
                    // The button text says "Delete All",
                    // and there is no icon.
SQLStatementButton button = new SQLStatementButton ("Delete All");

                    // Set the connection and SQLStatement
                    // properties.  Assume that "connection"
                    // is an SQLConnection object that is
                    // created and initialized elsewhere.
button.setConnection (connection);
button.setSQLStatement ("DELETE FROM MYTABLE");

                    // Add the button to a frame. Assume
                    // that "frame" is a JFrame created
                    // elsewhere.
frame.getContentPane ().add (button);
```

SQL ステートメントが発行されたら、 getResultSet()、 getMoreResults()、 getUpdateCount()、 または getWarnings() を使用して、 結果を検索します。

SQLStatementDocument クラス

SQLStatementDocument クラスは、 Java Foundation Classes (JFC) 文書インターフェースを実装したものです。 このクラスは、直接、JFC グラフィカル・テキスト・コンポーネントと一緒に使用できます。 単一行フィールド (JTextField) や複数行テキスト・エリア (JTextArea) など、 いくつかのテキスト・コンポーネントが JFC で使用できます。 SQLStatementDocument オブジェクトは、 テキスト・コンポーネントの中身を SQLConnection オブジェクトに関連付けます。 Java プログラムは、文書内に含まれている SQL ステートメントをいつでも実行でき、 結果があればそれを処理することもできます。

SQLStatementDocument を使用するためには、 接続プロパティーを設定する必要があります。 このプロパティーを設定するには、 コンストラクターを使用したり、 setConnection() メソッドを使用したりします。 その後、通常はテキスト・コンポーネントのコンストラクターや setDocument() メソッドを使用して、 SQLStatementDocument オブジェクトがテキスト・コンポーネントに「接続」されます。 execute() を使用すれば、 文書内に含まれている SQL ステートメントをいつでも実行できます。

以下の例では、 SQLStatementDocument を JTextField に作成します。

```
                    // Create an SQLStatementDocument
                    // object. Assume that "connection"
                    // is an SQLConnection object that is
                    // created and initialized elsewhere.
                    // The text of the document is
                    // initialized to a generic query.
    SQLStatementDocument document = new SQLStatementDocument (connection, "SELECT * FROM
QIWS.QCUSTCDT");

                    // Create a text field to present the
                    // document.
    JTextField textField = new JTextField ();
    textField.setDocument (document);

                    // Add the text field to a frame.
                    // Assume that "frame" is a JFrame
                    // created elsewhere.
    frame.getContentPane ().add (textField);

                    // Run the SQL statement that is in
                    // the text field.
    document.execute ();
```

SQL ステートメントが発行されたら、 getResultSet()、 getMoreResults()、 getUpdateCount()、または getWarnings() を使用して結果を検索します。

SQLResultSetFormPane クラス

SQLResultSetFormPane は、 SQL (構造化照会言語) 照会の結果を一定のフォームに表示します。 このフォームは、一度に 1 つずつレコードを表示します。 ユーザーが前方または後方にスクロールしたり、 先頭のレコードまたは最後のレコードまでスクロールしたり、 結果のビューを最新表示するためのボタンも表示します。

SQLResultSetFormPane を使用するためには、 接続プロパティーと照会プロパティーを設定する必要があります。 これらのプロパティーを設定するには、コンストラクターを使用するか、 setConnection() メソッドと setQuery() メソッドを使用します。 load() を使用すれば、 照会を実行し、ResultSet 内の最初のレコードを表示することができます。 結果が不要になったら、 close() を呼び出して ResultSet をクローズします。

以下の例では、SQLResultSetFormPane オブジェクトを作成し、そのオブジェクトをフレームに追加します。

```
                    // Create an SQLResultSetFormPane
                    // object. Assume that "connection"
                    // is an SQLConnection object that is
                    // created and initialized elsewhere.
    SQLResultSetFormPane formPane = new SQLResultSetFormPane (connection, "SELECT * FROM
QIWS.QCUSTCDT");

                    // Load the results.
    formPane.load ();

                    // Add the form pane to a frame.
                    // Assume that "frame" is a JFrame
                    // created elsewhere.
    frame.getContentPane ().add (formPane);
```

SQLResultSetTablePane クラス

SQLResultSetTablePane は、SQL (構造化照会言語) 照会の結果をテーブルに表示します。 テーブルの各行は ResultSet からのレコードを表示し、 テーブルの各列はフィールドを表示します。

SQLResultSetTablePane を使用するためには、 接続プロパティーと照会プロパティーを設定する必要があります。 プロパティーを設定するには、コンストラクターを使用するか、 setConnection() メソッドと setQuery() メソッドを使用します。 load() を使用すれば、照会を実行し、結果をテーブルに表示することができます。 結果が不要になったら、 close() を呼び出して ResultSet をクローズします。

以下の例では、SQLResultSetTablePane オブジェクトを作成し、 そのオブジェクトをフレームに追加します。

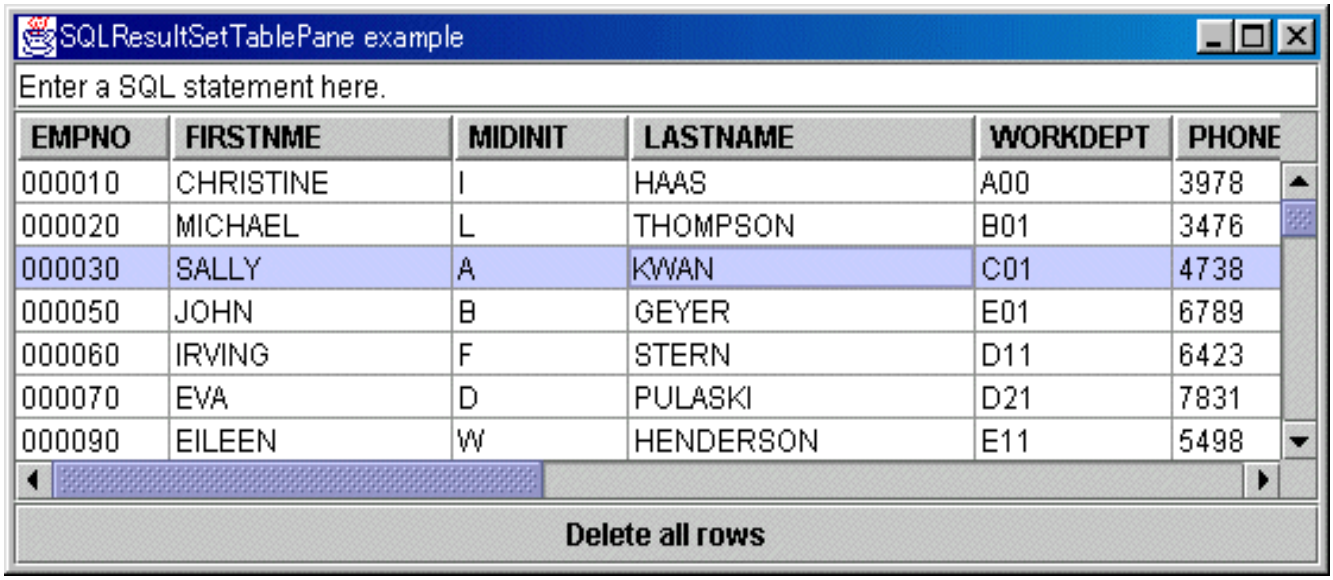
```
                    // Create an SQLResultSetTablePane
                    // object. Assume that "connection"
                    // is an SQLConnection object that is
                    // created and initialized elsewhere.
    SQLResultSetTablePane tablePane = new SQLResultSetTablePane (connection, "SELECT * FROM
QIWS.QCUSTCDT");

                    // Load the results.
    tablePane.load ();

                    // Add the table pane to a frame.
                    // Assume that "frame" is a JFrame
                    // created elsewhere.
    frame.getContentPane ().add (tablePane);
```

例

テーブルの内容を表示する SQLResultSetTablePane を表示します。この例では、 ユーザーが SQL ステートメントを入力できるようにするための SQLStatementDocument (以下のイメージにテキストで "Enter a SQL statement here" と示されている) と、 ユーザーがテーブルからすべての行を削除できるようにするための SQLStatementButton (以下のイメージにテキストで "Delete all rows" と示されている) とを使用します。

図 1 は、SQLResultSetTablePane グラフィカル・ユーザー・インターフェース・コンポーネントを示します。

図 1: SQLResultSetTablePane GUI コンポーネント


SQLResultSetTablePane example
Enter a SQL statement here.
EMPNO
FIRSTNME
MIDINIT
LASTNAME
WORKDEPT
PHONE
000010
CHRISTINE
I
HAAS
A00
3978
000020
MICHAEL
L
THOMPSON
B01
3476
000030
SALLY
A
KWAN
C01
4738
000050
JOHN
B
GEYER
E01
6789
000060
IRVING
F
STERN
D11
6423
000070
EVA
D
PULASKI
D21
7831
000090
EILEEN
W
HENDERSON
E11
5498
Delete all rows

SQLResultSetTableModel クラス

SQLResultSetTablePane は、 モデル・ビュー・コントローラー・パラダイムを使用して実装されます。 このパラダイムでは、 データとユーザー・インターフェースが別々のクラスに分離されます。 この実装により、 SQLResultSetTableModel が Java Foundation Classes (JFC) の JTable に統合されます。 SQLResultSetTableModel クラスは、照会の結果を管理します。一方、 JTable はその結果をグラフィカルに表示し、ユーザーとの対話を処理します。

SQLResultSetTablePane は機能性が高く、 大抵の要件は満たすことができますが、 呼び出し元が JFC コンポーネントをそれ以上に制御することを必要とする場合は、 SQLResultSetTableModel を直接使用することにより、 別の GUI コンポーネントとの統合をカスタマイズすることができます。

SQLResultSetTableModel を使用するためには、 接続プロパティーと照会プロパティーを設定する必要があります。 これらのプロパティーを設定するには、コンストラクターを使用するか、 setConnection() メソッドと setQuery() メソッドを使用します。 load() を使用すれば、照会を実行し、結果をロードすることができます。 結果が不要になったら、 close() を呼び出して ResultSet をクローズします。

以下の例では、 SQLResultSetTableModel オブジェクトを作成し、 そのオブジェクトを JTable で表します。

```
                    // Create an SQLResultSetTableModel
                    // object. Assume that "connection"
                    // is an SQLConnection object that is
                    // created and initialized elsewhere.
    SQLResultSetTableModel tableModel = new SQLResultSetTableModel (connection, "SELECT * FROM
QIWS.QCUSTCDT");

                    // Load the results.
    tableModel.load ();

                    // Create a JTable for the model.
    JTable table = new JTable (tableModel);

                    // Add the table to a frame.  Assume
                    // that "frame" is a JFrame created
                    // elsewhere.
    frame.getContentPane ().add (table);
```

SQL 照会ビルダー

SQLQueryBuilderPane は、SQL 照会を動的に構築するための対話式ツールを表します。

SQLQueryPane を使用するためには、 接続プロパティーを設定する必要があります。このプロパティーを設定するには、コンストラクターを使用するか、 setConnection() メソッドを使用します。 load() を使用すれば、 照会ビルダー・グラフィカル・ユーザー・インターフェースに必要なデータをロードできます。 getQuery() を使用すれば、ユーザーが構築した SQL 照会を取得できます。

以下の例では、SQLQueryBuilderPane オブジェクトを作成し、 そのオブジェクトをフレームに追加します。

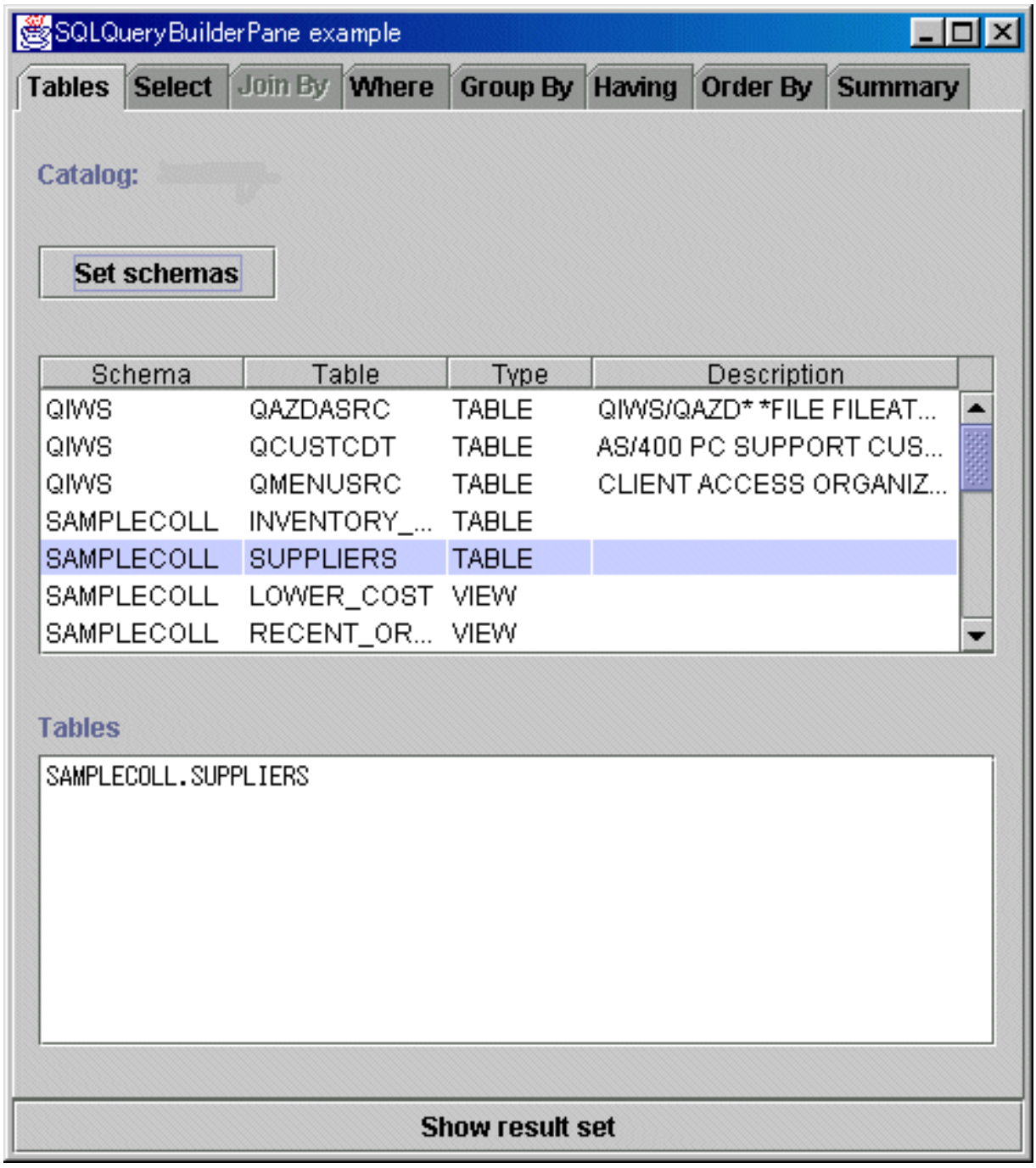
```
                    // Create an SQLQueryBuilderPane
                    // object. Assume that "connection"
                    // is an SQLConnection object that is
                    // created and initialized elsewhere.
SQLQueryBuilderPane queryBuilder = new SQLQueryBuilderPane (connection);

                    // Load the data needed for the query
                    // builder.
queryBuilder.load ();

                    // Add the query builder pane to a
                    // frame. Assume that "frame" is a
                    // JFrame created elsewhere.
frame.getContentPane ().add (queryBuilder);
```

例

SQLQueryBuilderPane およびボタンを表示します。 ボタンがクリックされると、 他のフレーム内にある SQLResultSetFormPane に照会結果が表示されます。

図 1 は、SQLQueryBuilderPane グラフィカル・ユーザー・インターフェース・コンポーネントを示します。

図 1: SQLQueryBuilderPane GUI コンポーネント


SQLQueryBuilderPane example
Tables
Select
Join By
Where
Group By
Having
Order By
Summary
Catalog:
Set schemas
Schema
Table
Type
Description
QIWS QAZDASRC TABLE QIWS/QAZD* *FILE FILEAT...
QIWS QCUSTCDT TABLE AS/400 PC SUPPORT CUS...
QIWS QMENUSRC TABLE CLIENT ACCESS ORGANIZ...
SAMPLECOLL INVENTORY_... TABLE
SAMPLECOLL SUPPLIERS TABLE
SAMPLECOLL LOWER_COST VIEW
SAMPLECOLL RECENT_OR... VIEW
Tables
SAMPLECOLL.SUPPLIERS
Show result set

ジョブ

ジョブ Vaccess (GUI) コンポーネントを使用すると、 Java プログラムを使って、 サーバー・ジョブおよびジョブ・ログ・メッセージのリストを GUI として表せるようになります。

以下のコンポーネントを使用できます。

- VJobList オブジェクトは、 AS400Panes での使用を目的とした、サーバー・ジョブのリストを表すリソースです。
- VJob オブジェクトは、 AS400Panes での使用を目的とした、ジョブ・ログ内のメッセージのリストを表すリソースです。

AS400Panes、VJobList オブジェクト、および VJob オブジェクトを一緒に使用することにより、ジョブ・リストまたはジョブ・ログで構成される多くのビューを表すことができます。

VJobList を使用するためには、システム・プロパティー、名前プロパティー、 番号プロパティー、およびユーザー・プロパティーを設定する必要があります。 これらのプロパティーを設定するには、コンストラクターを使用するか、 setSystem()、 setName()、 setNumber()、および setUser() プロパティーを使用します。

VJob を使用するためには、システム・プロパティーを設定する必要があります。 このプロパティーを設定するには、コンストラクターを使用するか、 setSystem() メソッドを使用します。

その後、AS400Pane のコンストラクターまたは setRoot() メソッドを使用し、 VJobList または VJob オブジェクトのいずれかを AS400Pane に「接続」します。

VJobList には、 AS400Panes で表されているジョブのセットを定義するのに役立つ他のプロパティーも含まれています。 setName() を使用すれば、特定の名前を持つジョブだけを表示するよう指定することができます。 setNumber() を使用して、 特定の番号を持つジョブだけを表示するよう指定することもできます。 同様に、 setUser() を使用すれば、 特定のユーザーに関係したジョブだけを表示するよう指定することもできます。

AS400Pane、VJobList、および VJob オブジェクトは、 作成時にはデフォルト状態に初期設定されています。 ジョブまたはジョブ・ログ・メッセージのリストは、作成時にはロードされません。 このリストの内容をロードするためには、 呼び出し元がいずれかのオブジェクトに対して load() メソッドを明示的に呼び出す必要があります。 これにより、サーバーとの通信が開始され、リストの内容が収集されます。

実行時に、ジョブ、ジョブ・リスト、 またはジョブ・ログ・メッセージを右マウス・ボタン・クリックして、 ショートカット・メニューを表示します。 ショートカット・メニューから「プロパティー」を選択して、 選択したオブジェクトに対するアクションを実行します。

- ジョブ - タイプ、状況などのプロパティーに関する作業を行ったり、 いくつかのプロパティーの値を変更することができます。
- ジョブ・リスト - 名前、数値、およびユーザー・プロパティーを操作します。 リストの内容を変更することもできます。
- ジョブ・ログ・メッセージ - 全テキスト、重大度、 および送信時刻などのプロパティーを表示します。

ユーザーがアクセスできるジョブは、 アクセスの許可を受けているジョブだけです。 さらに、 Java プログラムは、 ペインに対して setAllowActions() メソッドを実行することにより、 ユーザーがアクションを実行できないようにすることができます。

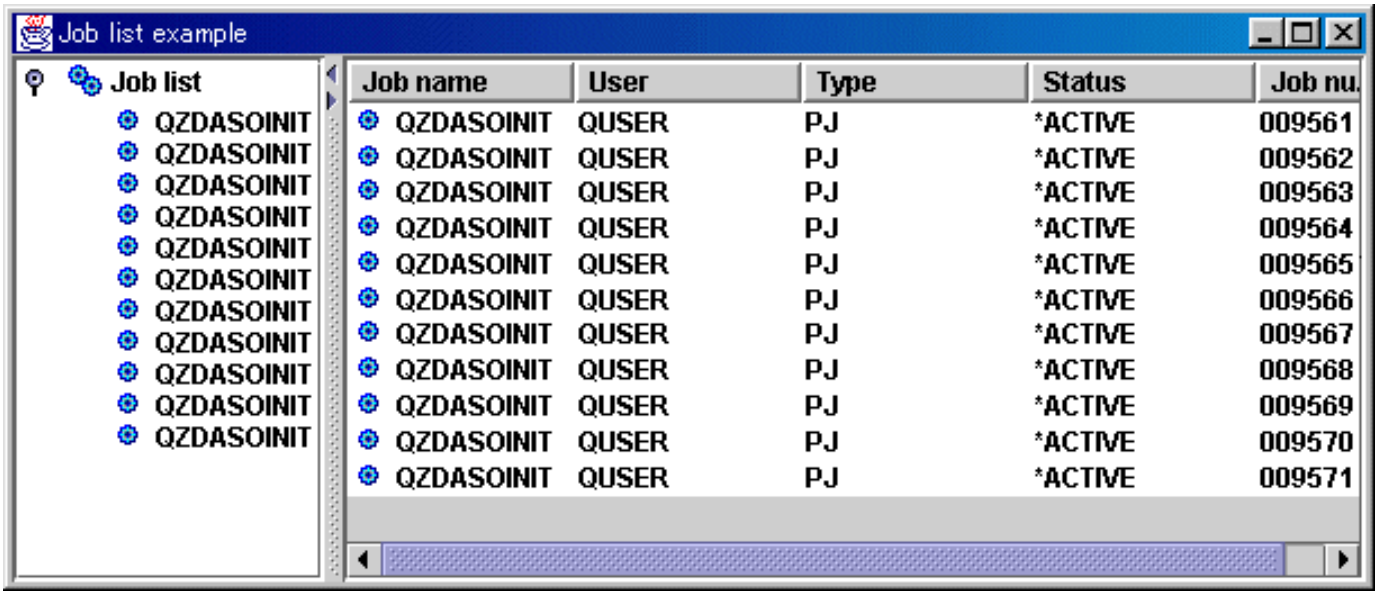
以下の例では、 VJobList を作成し、 それを AS400ExplorerPane に表します。

```
                    // Create the VJobList object. Assume
                    // that "system" is an AS400 object
                    // created and initialized elsewhere.
VJobList root = new VJobList (system);

                    // Create and load an
                    // AS400ExplorerPane object.
AS400ExplorerPane explorerPane = new AS400ExplorerPane (root);
explorerPane.load ();

                    // Add the explorer pane to a frame.
                    // Assume that "frame" is a JFrame
                    // created elsewhere.
frame.getContentPane ().add (explorerPane);
```

例

この VJobList の例 では、 ジョブのリストがロードされた AS400ExplorerPane を表示します。 このリストには、システム上にある、ジョブ名の同じジョブが表示されます。

以下のイメージは、 VJobList グラフィカル・ユーザー・インターフェース・コンポーネントを示します。



Vaccess メッセージ・クラス

メッセージ・グラフィカル・ユーザー・インターフェース・コンポーネントを使用すれば、 Java プログラムはサーバー・メッセージのリストを GUI として表すことができます。

以下のコンポーネントを使用できます。

- メッセージ・リスト・オブジェクトは、 AS400Panes での使用を目的とした、メッセージのリストを表すリソースです。 このメッセージが対象にしているメッセージ・リストは、 コマンドまたはプログラム呼び出しが生成したメッセージ・リストです。
- メッセージ待ち行列オブジェクトは、 AS400Ppanes での使用を目的とした、 サーバー・メッセージ待ち行列内のメッセージを表すリソースです。

AS400Panes は、 1 つ以上のサーバー・リソースを表示したり操作したりするための、 グラフィカル・ユーザー・インターフェース・コンポーネントです。 VMessageList オブジェクトと VMessageQueue オブジェクトは、 AS400Panes で使用されるサーバー・メッセージのリストを表すリソースです。

AS400Pane、VMessageList オブジェクト、および VMessageQueue オブジェクトを一緒に使用すれば、 数多くのメッセージ・リストのビューを表示することができます。 また、ユーザーはメッセージに対する操作を選択して実行できるようになります。

VMessageList クラス

VMessageList オブジェクトは、 AS400Panes での使用を目的とした、メッセージのリストを表すリソースです。 このメッセージが対象にしているメッセージ・リストは、 コマンドまたはプログラム呼び出しが生成したメッセージ・リストです。 以下のメソッドが、メッセージ・リストを戻します。

- CommandCall.getMessageList()
- CommandCallButton.getMessageList()
- CommandCallMenuItem.getMessageList()
- ProgramCall.getMessageList()
- ProgramCallButton.getMessageList()
- ProgramCallMenuItem.getMessageList()

VMessageList を使用するためには、 messageList プロパティーを設定することが必要です。 このプロパティーを設定するには、コンストラクターを使用するか、 setMessageList() メソッドを使用します。 その後、AS400Pane のコンストラクターまたは setRoot() メソッドを使用し、 ルートとして VMessageList オブジェクトを AS400Pane に "接続" します。

AS400Pane および VMessageList オブジェクトは、作成時にはデフォルト状態に初期設定されています。 メッセージのリストは、作成時はロードされません。 このリストの内容をロードするためには、 呼び出し元がいずれかのオブジェクトに対して load() メソッドを明示的に呼び出す必要があります。

実行時には、ユーザーはメッセージを右マウス・ボタン・クリックしてコンテキスト・メニューを表示させることにより、 メッセージに対するアクションを実行できます。 メッセージ・コンテキスト・メニューには、プロパティーと呼ばれる項目を組み込むことができ、 それは、重大度、タイプ、日付などのプロパティーを表示します。

呼び出し元では、ペインに対して setAllowActions() メソッドを実行することにより、 特定のユーザーがアクションを実行できないようにすることができます。

以下の例では、コマンド呼び出しが生成するメッセージを対象にした VMessageList を作成し、 それを AS400DetailsPane に表示します。

```
// Create the VMessageList object.
// Assume that "command" is a
// CommandCall object created and run
```

```
                    // elsewhere.
VMessageList root = new VMessageList (command.getMessageList ());

                    // Create and load an AS400DetailsPane
                    // object.
AS400DetailsPane detailsPane = new AS400DetailsPane (root);
detailsPane.load ();

                    // Add the details pane to a frame.
                    // Assume that "frame" is a JFrame
                    // created elsewhere.
frame.getContentPane ().add (detailsPane);
```

例

AS400DetailsPane を VMessageList オブジェクトと一緒に使用することにより、コマンド呼び出しが生成するメッセージのリストを表します。 図 1 は、VMessageList グラフィカル・ユーザー・インターフェース・コンポーネントを示します。

図 1: VMessageList GUI コンポーネント

VMessageQueue クラス

VMessageQueue オブジェクトは、 AS400Panes での使用を目的とした、 サーバー・メッセージ待ち行列内のメッセージを表すリソースです。

VMessageQueue を使用するためには、 システム・プロパティーとパス・プロパティーの両方を設定する必要があります。 これらのプロパティーを設定するには、 コンストラクターを使用したり、 setSystem() メソッドや setPath() メソッドを使用します。 その後、AS400Pane のコンストラクターあるいは setRoot() メソッドを使用することにより、 ルートとして VMessageQueue オブジェクトを AS400Pane に「接続」します。

VMessageQueue には、 AS400Panes で表されているメッセージのセットを定義するのに役立つ他のプロパティーも含まれています。 setSeverity() を使用すれば、表示するメッセージの重大度を指定できます。 setSelection() を使用すれば、いずれのタイプのメッセージを表示するかを指定できます。

AS400Pane および VMessageQueue オブジェクトは、作成時にはデフォルト状態に初期設定されています。 メッセージのリストは、作成時はロードされません。 このリストの内容をロードするためには、呼び出し元がいずれかのオブジェクトに対して load() メソッドを明示的に呼び出す必要があります。 これにより、サーバーとの通信が開始され、リストの内容が収集されます。

実行時には、ユーザーはどのメッセージまたはメッセージ待ち行列に対しても、 それを右マウス・ボタン・クリックしてコンテキスト・メニューを表示させることにより、 アクションを実行できます。 メッセージ待ち行列のコンテキスト・メニューには、以下の項目を組み込むことができます。

- クリア - メッセージ待ち行列をクリアします
- プロパティー - ユーザーが重大度および選択プロパティーを選択できるようにします。 これを使用すれば、リストの内容を変更できます

メッセージ待ち行列のメッセージに対しては、以下のアクションを実行できます。

- 削除 - メッセージ待ち行列から削除できます
- 応答 - 照会メッセージに応答できます
- プロパティー - 表示装置、タイプ、日付などのプロパティーを表示できます

当然のことながら、ユーザーがアクセスできるメッセージ待ち行列は、 アクセスの許可を受けているメッセージ待ち行列だけです。 一方、呼び出し元では、ペインに対して setAllowActions() メソッドを実行することにより、 ユーザーがアクションを実行できないようにすることができます。

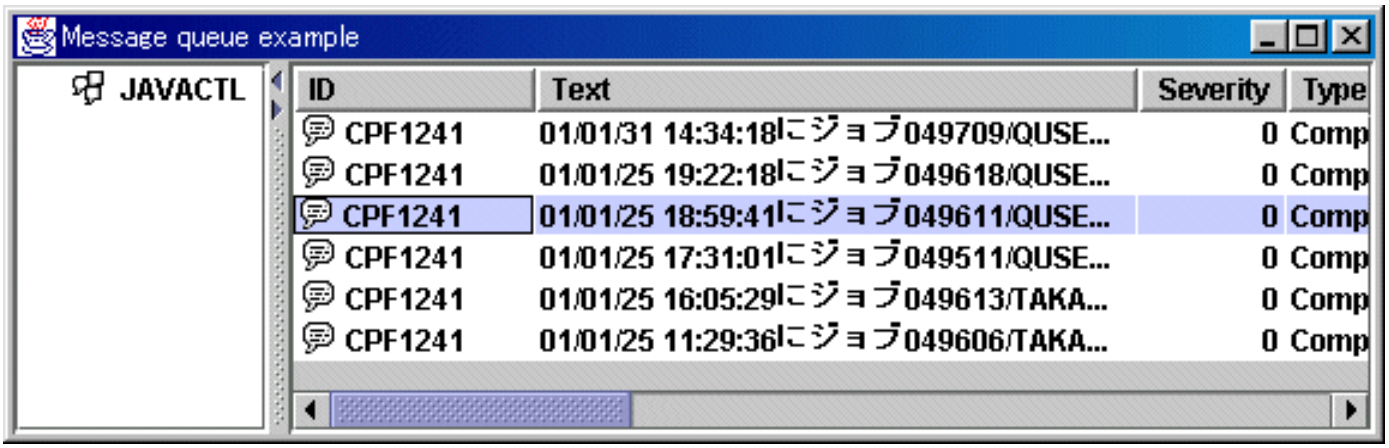
以下の例では、VMessageQueue を作成し、それを AS400ExplorerPane に表します。

```
                    // Create the VMessageQueue object.
                    // Assume that "system" is an AS400
                    // object created and initialized
                    // elsewhere.
VMessageQueue root = new VMessageQueue (system, "/QSYS.LIB/MYLIB.LIB/MYMSGQ.MSGQ");

                    // Create and load an
                    // AS400ExplorerPane object.
AS400ExplorerPane explorerPane = new AS400ExplorerPane (root);
explorerPane.load ();

                    // Add the explorer pane to a frame.
                    // Assume that "frame" is a JFrame
                    // created elsewhere.
frame.getContentPane ().add (explorerPane);
```

例

AS400ExplorerPane を VMessageQueue オブジェクトと一緒に使用することにより、 メッセージ待ち行列内にあるメッセージのリストを表します。 図 1 は、VMessageQueue グラフィカル・ユーザー・インターフェース・コンポーネントを示します。

図 1: VMessageQueue GUI コンポーネント



Permission クラス

Permission クラス情報は、 VIFSFile および VIFSDirectory クラスを使って、 グラフィカル・ユーザー・インターフェース (GUI) で使用することができます。許可は、これら個々のクラスのアクションとして追加されました。

以下の例は、VIFSDirectory クラスで許可を使用する方法を示しています。

```
// Create AS400 object
AS400 as400 = new AS400();

// Create an IFSDirectory using the system name
// and the full path of a QSYS object
VIFSDirectory directory = new VIFSDirectory(as400,
                                "/QSYS.LID/testlib1.lib");

// Create as explorer Pane
AS400ExplorerPane pane = new AS400ExplorerPane((VNode)directory);

// Load the information
pane.load();
```

Vaccess 印刷クラス

vaccess パッケージ内の以下のコンポーネントを使用すれば、Java プログラムを使って、 サーバー印刷リソースのリストをグラフィカル・ユーザー・インターフェースとして表せるようになります。

- VPrinters オブジェクトは、 AS400Panes での使用を目的とした、プリンターのリストを表すリソースです。
- VPrinter オブジェクトは、 AS400Panes での使用を目的とした、プリンターとそのスプール・ファイルを表すリソースです。
- VPrinterOutput オブジェクトは、 AS400Panes での使用を目的とした、スプール・ファイルのリストを表すリソースです。
- SpooledFileViewer オブジェクトは、視覚的にスプール・ファイルを表すリソースです。

AS400Panes は、 1 つ以上のサーバー・リソースを表示したり操作したりするための GUI コンポーネントです。 VPrinters、VPrinter、および VPrinterOutput オブジェクトは、 AS400Panes で使用されるサーバー印刷リソースのリストを表すリソースです。

AS400Pane、VPrinters、VPrinter、および VPrinterOutput オブジェクトを一緒に使用して、 印刷リソースの数多くのビューを表すことができます。 また、 ユーザーはそれらのリソースに関する操作を選択および実行できるようになります。

VPrinters クラス

VPrinters オブジェクトは、 AS400Panes での使用を目的とした、プリンターのリストを表すリソースです。

VPrinters を使用するためには、 システム・プロパティーを設定する必要があります。 このプロパティーを設定するには、コンストラクターを使用するか、 setSystem() メソッドを使用します。 その後、AS400Pane のコンストラクターあるいは setRoot() メソッドを使用し、 ルートとして VPrinters オブジェクトを AS400Pane に "接続" します。

VPrinters オブジェクトには、 AS400Panes に表示されるプリンターのセットを定義するのに役立つプロパティーがもう 1 つあります。 setPrinterFilter() を使用すれば、 どのプリンターを表示するかを定義するフィルターを指定できます。

AS400Pane および VPrinters オブジェクトは、作成時にはデフォルト状態に初期設定されています。 その時点では、プリンターのリストはロードされていません。 このリストの内容をロードするためには、 呼び出し元がいずれかのオブジェクトに対して load() メソッドを明示的に呼び出す必要があります。

実行時には、ユーザーはどのプリンター・リストまたはプリンターに対しても、 それを右マウス・ボタン・クリックしてコンテキスト・メニューを表示させることにより、アクションを実行できます。 プリンター・リストのコンテキスト・メニューは、 プロパティーと呼ばれる項目を組み込むことができ、それによりユーザーは、 リストの内容を変更できるプリンター・フィルター・プロパティーを設定することが可能になります。

プリンター・コンテキスト・メニューには以下の項目を組み込むことができます。

- 保留 - プリンターを保留します
- 解放 - プリンターを解放します
- 開始 - プリンターを開始します
- 停止 - プリンターを停止します
- 使用可能にする - プリンターを使用可能にします
- 使用不可にする - プリンターを使用不可にします
- プロパティー - プリンターのプロパティーを表示し、 ユーザーがフィルターを設定できるようにします

ユーザーがアクセスできるプリンターは、アクセスの許可を受けているプリンターだけです。 一方、呼び出し元では、ペインに対して setAllowActions() メソッドを実行することにより、 ユーザーがアクションを実行できないようにすることができます。

以下の例では、VPrinters オブジェクトを作成し、それを AS400TreePane に表示します。

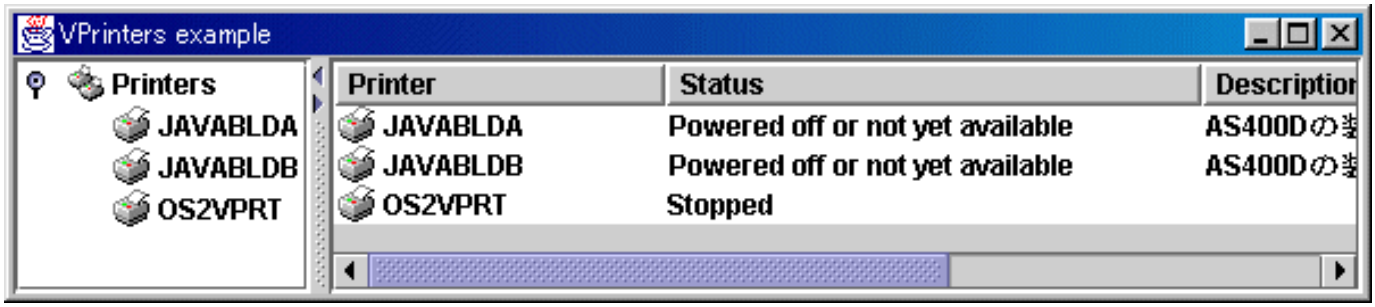
```
                    // Create the VPrinters object.
                    // Assume that "system" is an AS400
                    // object created and initialized
                    // elsewhere.
VPrinters root = new VPrinters (system);

                    // Create and load an AS400TreePane
                    // object.
AS400TreePane treePane = new AS400TreePane (root);
```

```
treePane.load ();

                    // Add the tree pane to a frame.
                    // Assume that "frame" is a JFrame
                    // created elsewhere.
frame.getContentPane ().add (treePane);
```

例

AS400ExplorerPane を VPrinters オブジェクトと一緒に使用することにより、 印刷リソースを表します。 図 1 は、VPrinters グラフィカル・ユーザー・インターフェース・コンポーネントを示します。

図 1: VPrinters GUI コンポーネント



VPrinter

VPrinter オブジェクトは、 AS400Panes での使用を目的とした、 サーバー・プリンターとそのスプール・ファイルを表すリソースです。

VPrinter を使用するためには、プリンター・プロパティーを設定する必要があります。 このプロパティーを設定するには、コンストラクターを使用するか、 setPrinter() メソッドを使用します。 その後、 AS400Pane のコンストラクターあるいは setRoot() メソッドを使用し、 ルートとして VPrinter オブジェクトを AS400Pane に「接続」します。

AS400Pane および VPrinter オブジェクトは、作成時にはデフォルト状態に初期設定されています。 プリンターの属性およびスプール・ファイルのリストは、作成時にはロードされません。

このリストの内容をロードするためには、 呼び出し元がいずれかのオブジェクトに対して load() メソッドを明示的に呼び出す必要があります。 これにより、サーバーとの通信が開始され、リストの内容が収集されます。

実行時には、ユーザーはどのプリンターまたはスプール・ファイルに対しても、 それを右マウス・ボタン・クリックしてコンテキスト・メニューを表示させることにより、 アクションを実行できます。 メッセージ待ち行列のコンテキスト・メニューには、以下の項目を組み込むことができます。

- 保留 - プリンターを保留します
- 解放 - プリンターを解放します
- 開始 - プリンターを開始します
- 停止 - プリンターを停止します
- 使用可能にする - プリンターを使用可能にします
- 使用不可にする - プリンターを使用不可にします
- プロパティー - プリンターのプロパティーを表示し、 ユーザーがフィルターを設定できるようにします

プリンター用にリストされたスプール・ファイルのコンテキスト・メニューには以下の項目を組み込むことができます。

- 応答 - スプール・ファイルに応答します
- 保留 - スプール・ファイルを保留します
- 解放 - スプール・ファイルを解放します
- 次を印刷 - 次のスプール・ファイルを印刷します
- 送信 - スプール・ファイルを送信します
- 移動 - スプール・ファイルを移動します
- 削除 - スプール・ファイルを削除します
- プロパティー - スプール・ファイルのいろいろなプロパティーを表示するとともに、 ユーザーがそれらを変更できるようにします

ユーザーがアクセスできるプリンターおよびスプール・ファイルは、 アクセスの許可を受けているプリンターおよびスプール・ファイルだけです。 一方、呼び出し元では、 ペインに対して setAllowActions() メソッドを実行することにより、 ユーザーがアクションを実行できないようにすることができます。

以下の例では、VPrinter を作成し、それを AS400ExplorerPane に表示します。

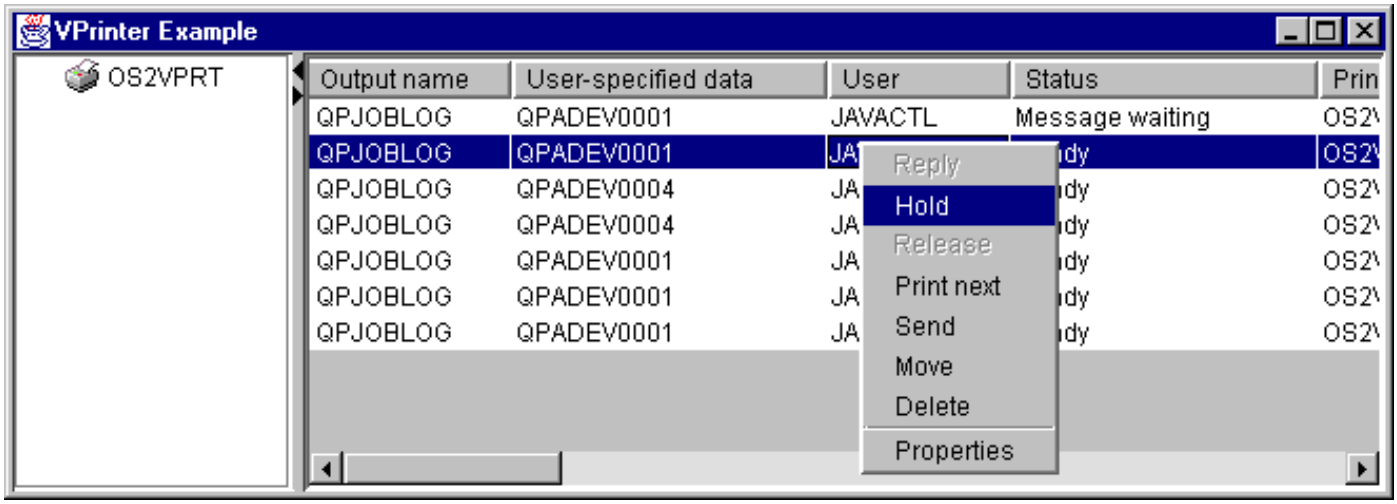
```
                    // Create the VPrinter object.
                    // Assume that "system" is an AS400
                    // object created and initialized
                    // elsewhere.
VPrinter root = new VPrinter (new Printer (system, "MYPRINTER"));
```

```
                    // Create and load an
                    // AS400ExplorerPane object.
AS400ExplorerPane explorerPane = new AS400ExplorerPane (root);
explorerPane.load ();

                    // Add the explorer pane to a frame.
                    // Assume that "frame" is a JFrame
                    // created elsewhere.
frame.getContentPane ().add (explorerPane);
```

例

AS400ExplorerPane を VPrinter オブジェクトと一緒に使用することにより、印刷リソースを表しています。 図 1 は、VPrinter グラフィカル・ユーザー・インターフェース・コンポーネントを示します。

図 1: VPrinter GUI コンポーネント



VPrinterOutput クラス

VPrinterOutput オブジェクトは、 AS400Panes での使用を目的とした、 サーバー上にあるスプール・ファイルのリストを表すリソースです。

VPrinterOutput を使用するためには、 システム・プロパティーを設定する必要があります。 このプロパティーを設定するには、コンストラクターを使用するか、 setSystem() メソッドを使用します。 その後、AS400Pane のコンストラクターまたは setRoot() メソッドを使用し、 ルートとして VPrinterOutput オブジェクトを AS400Pane に「接続」します。

VPrinterOutput オブジェクトには、 AS400Panes で表されているスプール・ファイルのセットを定義するのに役立つ他のプロパティーも含まれています。 setFormTypeFilter() を使用すると、いずれのタイプのフォームを表示するかを指定できます。 setUserDataFilter() を使用すれば、いずれのユーザー・データを表示するかを指定できます。 最後に、 setUserFilter() を使用すれば、 いずれのユーザー・スプール・ファイルを表示するかを指定できます。

AS400Pane および VPrinterOutput オブジェクトは、作成時にはデフォルト状態に初期設定されています。 スプール・ファイルのリストは、作成時はロードされません。 このリストの内容をロードするためには、 呼び出し元がいずれかのオブジェクトに対して load() メソッドを明示的に呼び出す必要があります。 これにより、サーバーとの通信が開始され、リストの内容が収集されます。

実行時には、ユーザーはどのスプール・ファイルまたはスプール・ファイル・リストに対しても、 それを右マウス・ボタン・クリックしてコンテキスト・メニューを表示させることにより、アクションを実行できます。 スプール・ファイル・リストのコンテキスト・メニューは、 プロパティーと呼ばれる項目を組み込むことができ、それによりユーザーは、リストの内容を変更できるフィルター・プロパティーを設定することが可能になります。

スプール・ファイルのコンテキスト・メニューには以下の項目を組み込むことができます。

- 応答 - スプール・ファイルに応答します
- 保留 - スプール・ファイルを保留します
- 解放 - スプール・ファイルを解放します
- 次を印刷 - 次のスプール・ファイルを印刷します
- 送信 - スプール・ファイルを送信します
- 移動 - スプール・ファイルを移動します
- 削除 - スプール・ファイルを削除します
- プロパティー - スプール・ファイルのいろいろなプロパティーを表示するとともに、 ユーザーがそれらを変更できるようにします

当然のことながら、 ユーザーがアクセスできるスプール・ファイルは、 アクセスの

許可を受けているスプール・ファイルだけです。 一方、呼び出し元では、 ペインに対して setAllowActions() メソッドを実行することにより、 ユーザーがアクションを実行できないようにすることができます。

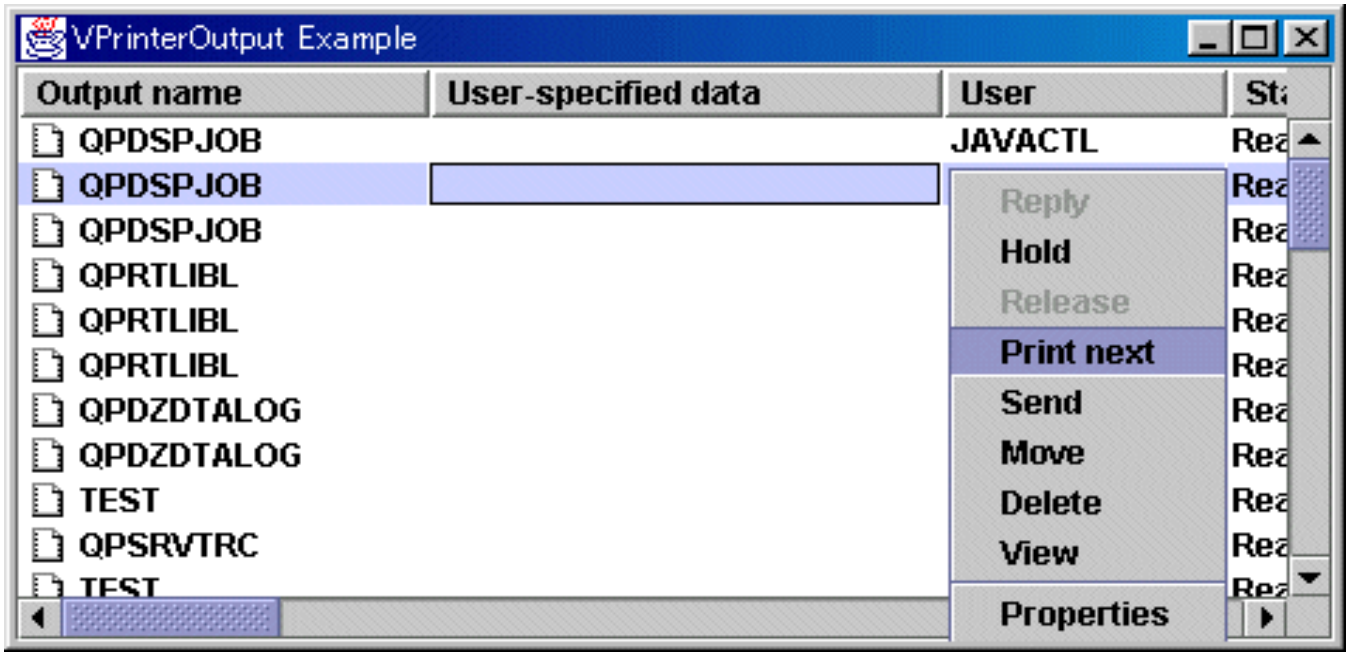
以下の例では、VPrinterOutput を作成し、それを AS400ListPane に表示します。

```
                    // Create the VPrinterOutput object.
                    // Assume that "system" is an AS400
                    // object created and initialized
                    // elsewhere.
VPrinterOutput root = new VPrinterOutput (system);

                    // Create and load an AS400ListPane
                    // object.
AS400ListPane listPane = new AS400ListPane (root);
listPane.load ();

                    // Add the list pane to a frame.
                    // Assume that "frame" is a JFrame
                    // created elsewhere.
frame.getContentPane ().add (listPane);
```

例

印刷リソース VPrinterOutput オブジェクトを使用することにより、スプール・ファイルのリストを表します。 図 1 は、VPrinterOutput グラフィカル・ユーザー・インターフェース・コンポーネントを示します。

図 1: VPrinterOutput GUI コンポーネント


VPrinterOutput Example
Output name
User-specified data
User
Sta
QPDSPJOB
JAVACTL
QPDSPJOB
QPDSPJOB
QPRTLIBL
QPRTLIBL
QPRTLIBL
QPDZDTALOG
QPDZDTALOG
TEST
QPSRVTRC
Reply
Hold
Release
Print next
Send
Move
Delete
View
Properties

SpooledFileViewer クラス

SpooledFileViewer クラスは、 印刷用にスプーリングされている高機能印刷 (AFP) およびシステム・ネットワーク体系文字ストリング (SCS) ファイルを表示するためのウィンドウを作成します。 このクラスにより、図 1 に示されているように、 ほとんどのワード・プロセッシング・プログラムに共通の、「プレビューの表示」機能が追加されます。

スプール・ファイル・ビューアーは、 ファイルを印刷することよりもファイルのレイアウトを正確に表示することのほうが重要な場合、 データを表示するほうが印刷するよりも経済的な場合、 あるいはプリンターを利用できない場合に特に役立ちます。

注: SS1 オプション 8 (AFP 互換フォント) が、 ホスト・サーバーにインストールされている必要があります。

SpooledFileViewer クラスの使用

SpooledFileViewer クラスのインスタンスの作成には、 3 つのコンストラクター・メソッドを使用できます。 SpooledFileViewer() コンストラクターは、 スプール・ファイルが関連付けられていないビューアーを作成する場合に使用できます。 このコンストラクターを使用する場合は、 後で setSpooledFile(SpooledFile) を使ってスプール・ファイルを設定することが必要になります。 SpooledFileViewer(SpooledFile) コンストラクターは、 ページ 1 を初期ビューとする、特定のスプール・ファイル用のビューアーを作成する場合に使用できます。 最後に、SpooledFileViewer(spooledFile, int) コンストラクターは、 指定したページを初期ビューとする、特定のスプール・ファイル用のビューアーを作成する場合に使用できます。 どのコンストラクターを使用する場合でも、ビューアーの作成後にスプール・ファイル・データを実際に取り出すには、 load() を実行しなければなりません。

次に、プログラムは以下のメソッドを使って、スプール・ファイルの個々のページを調べます。

- load FlashPage()
- load Page()
- pageBack()
- pageForward()

ただし、文書の特定のセクションをさらに詳しく調べる必要がある場合は、 以下のメソッドを使って各ページの表示比率を変更すれば、 その文書のページ・

イメージを拡大または縮小することができます。

- fitHeight()
- fitPage()
- fitWidth()
- actualSize()

プログラムは、最後に、入力ストリームをクローズし、 ストリームとのリソース関連を解放する close() メソッドを呼び出します。

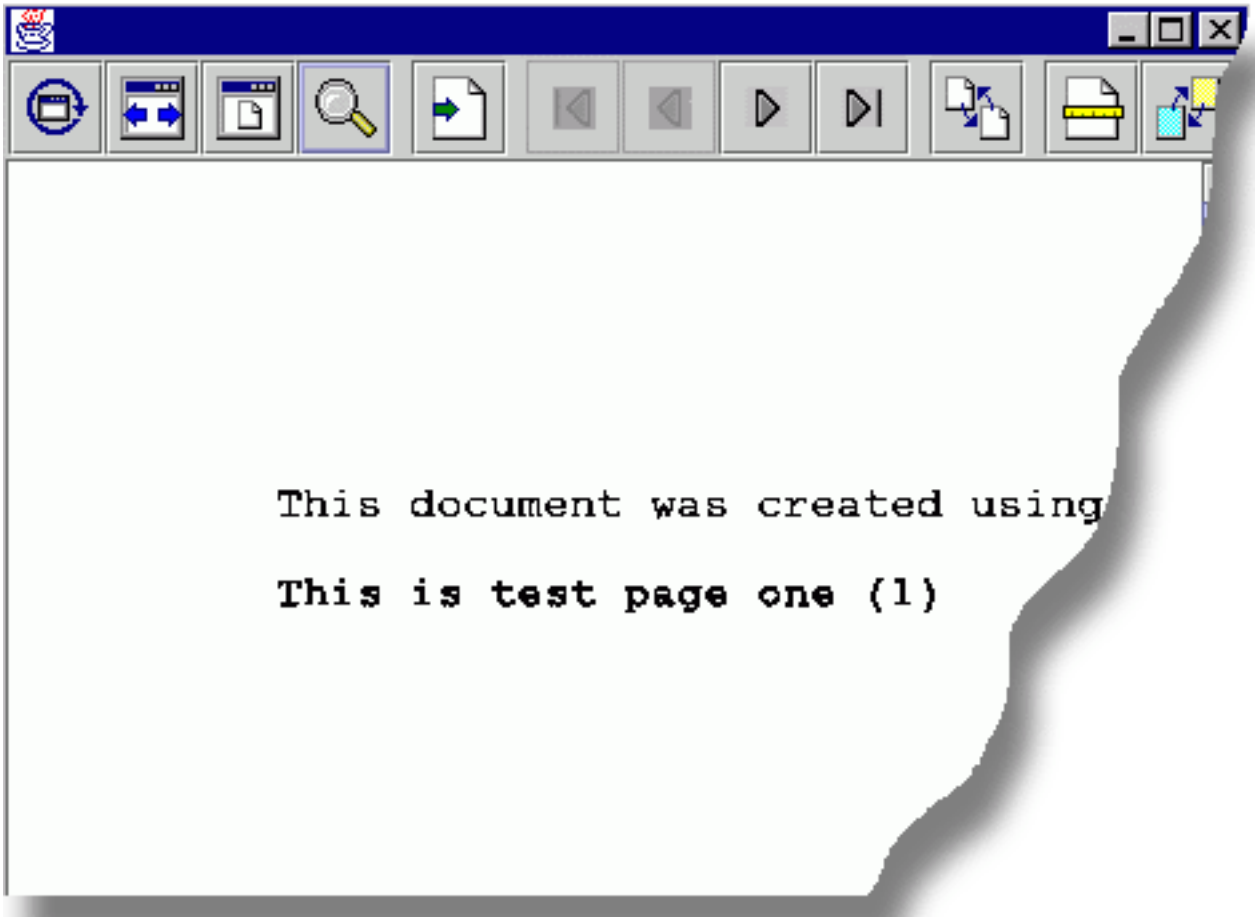
SpooledFileViewer の使用

SpooledFileViewer クラスのインスタンスは、実際には、 AFP または SCS スプール・ファイルを表示およびナビゲートできるビューアーをグラフィカルに表したものです。 たとえば、以下のコードを使用すると、図 1 のスプール・ファイル・ビューアーが作成され、 サーバーで以前に作成されたスプール・ファイルが表示されます。

注: ボタンの機能の説明を見たい時は、 図 1 のイメージ上のボタンを選択するか、 あるいは (ユーザーのブラウザーが JavaScript を使用可能でない場合は)、ツールバーの説明を参照してください。

```
// Assume splf is the spooled file.
// Create the spooled file viewer
SpooledFileViewer splfv = new SpooledFileViewer(splf, 1);
splfv.load();
// Add the spooled file viewer to a frame
JFrame frame = new JFrame("My Window");
frame.getContentPane().add(splfv);
frame.pack();
frame.show();
```

図 1: SpooledFileViewer



SpooledFileViewer ツールバーの説明


「実サイズ (actual size)」ボタンを押すと、 actualSize() が使用され、スプール・ファイルのページのイメージが元のサイズに戻ります。


「幅に合わせる (fit width)」ボタンを押すと、 fitWidth() メソッドが使用され、 スプール・ファイルのページのイメージがビューアーのフレームの左右の両端まで広げられます。


「ページに合わせる (fit page)」ボタンを押すと、 fitPage() メソッドが使用され、 スプール・ファイルのページのイメージがスプール・ファイル・ビューアーのフレーム内で上下左右に最大化されます。


「ズーム」ボタンを使用して、 スプール・ファイルのページのイメージの拡大縮小を行うことができます。 その際、事前設定されたパーセントのいず

れかを選択することもできますし、「ズーム」ボタンを押した後にダイアログ・ボックスに表示されるテキスト・フィールドに 任意のパーセントを入力することもできます。

 「ページへ移動 (go to page)」ボタンを使用すれば、 スプール・ファイル内の特定のページを表示することができます。

 「最初のページ (first page)」ボタンを押すと、 スプール・ファイルの最初のページが表示されます。 最初のページがすでに表示されている場合、 このボタンを使用することはできません。

 「前ページ (previous page)」ボタンを押すと、 現在表示されているページの 1 つ前のページが表示されます。

 「次ページ (next page)」ボタンを押すと、 現在表示されているページの 1 つ後のページが表示されます。

 「最後のページ (last page)」ボタンを押すと、 スプール・ファイルの最後のページが表示されます。 最後のページがすでに表示されている場合、 このボタンを使用することはできません。

 「フラッシュ・ページをロード (load flash page)」 ボタンを押すと、loadFlashPage() メソッドが使用され、 1 つ前に表示されていたページに戻ります。

 「用紙サイズを設定 (set paper size)」ボタンを押すと、 用紙サイズを設定することができます。

 「表示精度を設定 (set viewing fidelity)」 ボタンを押すと、表示精度を設定することができます。

Vaccess ProgramCall クラス

vaccess パッケージ内のプログラム呼び出しコンポーネントを Java プログラムで使用すれば、 サーバー・プログラムを呼び出すボタンやメニュー項目を表すことができます。 入力、出力、および入出力パラメーターの指定には、 ProgramParameter オブジェクトを使用できます。 プログラムを実行すると、出力および入出力パラメーターにはサーバー・プログラムから戻されたデータが入ります。

ProgramCallButton オブジェクトは、 押されるとサーバー・プログラムを呼び出すボタンを表します。 ProgramCallButton クラスは、 Java Foundation Classes (JFC) JButton クラスを拡張して、 すべてのボタンが一貫性のある外観や動作を持つようにします。

同様に、ProgramCallMenuItem オブジェクトは、 選択されるとサーバー・プログラムを呼び出すメニュー項目を表します。 ProgramCallMenuItem クラスも、JFC JMenuItem クラスを拡張して、 すべてのメニュー項目が一貫性のある外観や動作を持つようにします。

Vaccess プログラム呼び出しコンポーネントを使用するためには、 システム・プロパティーとプログラム・プロパティーの両方を設定する必要があります。 これらのプロパティーを設定するには、コンストラクターを使用するか、 setSystem() メソッドと setProgram() メソッドを使用します。

以下の例では、ProgramCallMenuItem を作成します。 実行時にメニュー項目を選択すると、プログラムが呼び出されます。

```
                    // Create the ProgramCallMenuItem
                    // object. Assume that "system" is
                    // an AS400 object created and
                    // initialized elsewhere. The menu
                    // item text says "Select Me", and
                    // there is no icon.
ProgramCallMenuItem menuItem = new ProgramCallMenuItem ("Select Me", null, system);

                    // Create a path name object that
                    // represents program MYPROG in
                    // library MYLIB
QSYSObjectPathName programName = new QSYSObjectPathName("MYLIB", "MYPROG", "PGM");

                    // Set the name of the program.
menuItem.setProgram (programName.getPath());

                    // Add the menu item to a menu.
                    // Assume that the menu was created
                    // elsewhere.
menu.add (menuItem);
```

サーバー・プログラムが実行されると、サーバー・メッセージが戻される場合があります。 サーバー・プログラムがいつ実行されたかを検出するには、addActionCompletedListener() メソッドを使用して ActionCompletedListener をボタンまたはメニュー項目に追加します。 プログラムを実行すると、それらのリスナーすべてに ActionCompletedEvent が向けられます。 リスナーは getMessageList() メソッドを使用して、 プログラムが生成したどのサーバー・メッセージでも検索することができます。

以下の例では、 プログラムが生成したすべてのサーバー・メッセージを処理する ActionCompletedListener を追加します。

```
                    // Add an ActionCompletedListener
                    // that is implemented by using an
                    // anonymous inner class. This is a
```

```
                    // convenient way to specify simple
                    // event listeners.
menuItem.addActionCompletedListener (new ActionCompletedListener ()
{
     public void actionCompleted (ActionCompletedEvent event)
     {
                    // Cast the source of the event to a
                    // ProgramCallMenuItem.
          ProgramCallMenuItem sourceMenuItem = (ProgramCallMenuItem) event.getSource ();

                    // Get the list of server messages
                    // that the program generated.
          AS400Message[] messageList = sourceMenuItem.getMessageList ();

                    // ... Process the message list.
     }
});
```

パラメーター

ProgramParameter オブジェクトは、 Java プログラムとサーバー・プログラムとの間でデータを受け渡しする際に使用します。 入力データは、 setInputData() メソッドで設定されます。 プログラム実行後に、getOutputData() メソッドで出力データを検索します。

各パラメーターはバイト配列です。 バイト配列を Java 形式とサーバー形式との間で変換するのは、Java プログラムの責任です。 データ変換クラスは、データを変換するためのメソッドを提供します。

プログラム呼び出し GUI コンポーネントにパラメーターを追加することもできます。 パラメーターを一度に 1 つずつ追加するには、addParameter() メソッドを使用します。 すべてのパラメーターを一度にまとめて追加するには、setParameterList() メソッドを使用します。

ProgramParameter オブジェクトの使用に関する詳細については、ProgramCall アクセス・クラスを参照してください。

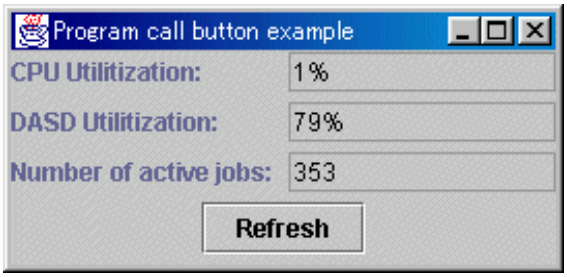
以下の例では、2 つのパラメーターを追加します。

```
                    // The first parameter is a String
                    // name of up to 100 characters.
                    // This is an input parameter.
                    // Assume that "name" is a String
                    // created and initialized elsewhere.
AS400Text parm1Converter = new AS400Text (100, system.getCcsid (), system);
ProgramParameter parm1 = new ProgramParameter (parm1Converter.toBytes (name));
menuItem.addParameter (parm1);

                    // The second parameter is an Integer
                    // output parameter.
AS400Bin4 parm2Converter = new AS400Bin4 ();
ProgramParameter parm2 = new ProgramParameter (parm2Converter.getByteLength ());
menuItem.addParameter (parm2);

                    // ... after the program is called,
                    // get the value returned as the
                    // second parameter.
int result = parm2Converter.toInt (parm2.getOutputData ());
```

例

以下の例は、 アプリケーションでの ProgramCallButton の使用例です。 図 1 は、ProgramCallButton がどのように表示されるかを示しています。

図 1: アプリケーションで ProgramCallButton を使用する



Vaccess レコード・レベルのアクセス・クラス

vaccess パッケージ内のレコード・レベルのアクセス・クラスを Java プログラムで使用すると、 サーバー・ファイルのさまざまなビューを表すことができます。

以下のコンポーネントを使用できます。

- RecordListFormPane は、 サーバー・ファイルからのレコード・リストをフォームとして表します。
- RecordListTablePane は、 サーバー・ファイルからのレコード・リストをテーブルに表します。
- RecordListTableModel は、 特定のテーブルに関連するサーバー・ファイルからのレコード・リストを管理します。

キー順アクセス

レコード・レベルのアクセス GUI コンポーネントを使用して、 サーバー・ファイルにキー順アクセスすることができます。 キー順アクセスとは、 Java プログラムでキーを指定することによってファイルのレコードにアクセスできることを意味します。

キー順アクセスは、レコード・レベルのアクセス GUI コンポーネントごとに同じ働きをします。 順次アクセスではなく、キー順アクセスを指定するには、setKeyed() を使用します。 コンストラクターまたは setKey() メソッドを使用して、キーを指定してください。 キーの指定方法に関する詳細については、 キーの指定を参照してください。

デフォルトでは、 指定されたキーと等しいキーを持つレコードだけが表示されます。 この設定を変更するには、 コンストラクターまたは setSearchType() メソッドを使用して、 searchType プロパティーを指定します。 可能な選択は、以下のとおりです。

- KEY_EQ - 指定されたキーと等しいキーを持つレコードを表示します。
- KEY_GE - 指定されたキーより大きいまたは等しいキーを持つレコードを表示します。
- KEY_GT - 指定されたキーより大きいキーを持つレコードを表示します。
- KEY_LE - 指定されたキーより小さいまたは等しいキーを持つレコードを表示します。
- KEY_LT - 指定されたキーより小さいキーを持つレコードを表示します。

以下の例では、 RecordListTablePane オブジェクトを作成して、 キーよりも小さいまたは等しいレコードをすべて表示します。

```
// Create a key that contains a
// single element, the Integer 5.
```

```
Object[] key = new Object[1];
key[0] = new Integer (5);

                    // Create a RecordListTablePane
                    // object. Assume that "system" is an
                    // AS400 object that is created and
                    // initialized elsewhere. Specify
                    // the key and search type.
RecordListTablePane tablePane = new RecordListTablePane (system,
     "/QSYS.LIB/QGPL.LIB/PARTS.FILE", key, RecordListTablePane.KEY_LE);

                    // Load the file contents.
tablePane.load ();

                    // Add the table pane to a frame.
                    // Assume that "frame" is a JFrame
                    // created elsewhere.
frame.getContentPane ().add (tablePane);
```

RecordListFormPane クラス

RecordListFormPane は、 サーバー・ファイルの内容をフォームとして表します。 このフォームは、特定の時点における 1 つのレコードを表示します。 ユーザーがそのレコード内をスクロール移動したり、 ファイルの内容のビューを最新表示するためのボタンも提供します。

RecordListFormPane を使用するためには、 システム・プロパティーと fileName プロパティーを設定する必要があります。 これらのプロパティーを設定するには、コンストラクターを使用するか、 setSystem() メソッドおよび setFileName() メソッドを使用します。 load() を使用すれば、ファイル内容を検索し、 最初のレコードを表示することができます。 ファイル内容が不要になったら、 close() を呼び出して、ファイルをクローズします。

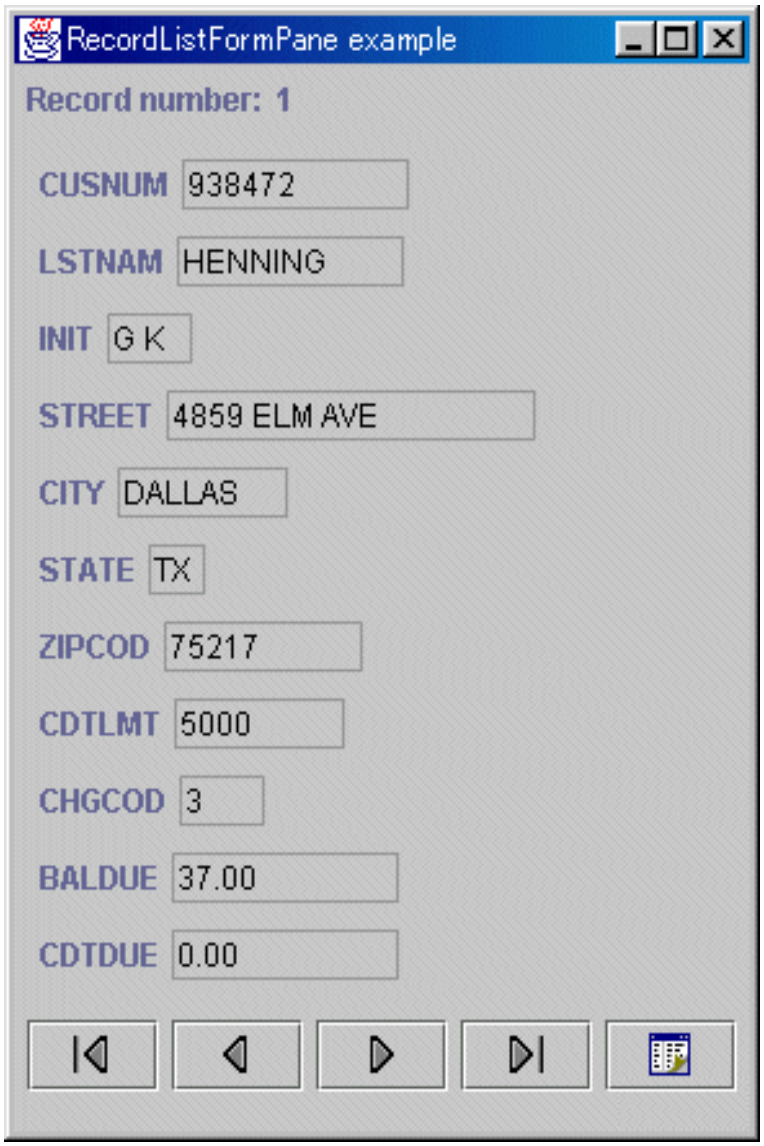
以下の例では、RecordListFormPane オブジェクトを作成し、 それをフレームに追加します。

```
                    // Create a RecordListFormPane
                    // object. Assume that "system" is
                    // an AS400 object that is created
                    // and initialized elsewhere.
    RecordListFormPane formPane = new RecordListFormPane (system,
"/QSYS.LIB/QIWS.LIB/QCUSTCDT.FILE");

                    // Load the file contents.
    formPane.load ();

                    // Add the form pane to a frame.
                    // Assume that "frame" is a JFrame
                    // created elsewhere.
    frame.getContentPane ().add (formPane);
```

例

ファイルの内容を表示する RecordListFormPane を表示しています。 図 1 は、RecordListFormPane グラフィカル・ユーザー・インターフェース・コンポーネントを示します。

図 1: RecordListFormPane GUI コンポーネント


RecordListFormPane example
Record number: 1
CUSNUM 938472
LSTNAM HENNING
INIT G K
STREET 4859 ELM AVE
CITY DALLAS
STATE TX
ZIPCOD 75217
CDTLMT 5000
CHGCOD 3
BALDUE 37.00
CDTDUE 0.00

RecordListTablePane クラス

RecordListTablePane は、サーバー・ファイルの内容をテーブルとして表します。 テーブルの各行はファイルからのレコードを表示し、 テーブルの各列はフィールドを表示します。

RecordListTablePane を使用するためには、 システム・プロパティーと fileName プロパティーを設定する必要があります。 これらのプロパティーを設定するには、コンストラクターを使用するか、 setSystem() メソッドおよび setFileName() メソッドを使用します。 load() を使用すれば、ファイル内容を検索し、 レコードをテーブルに表示することができます。 ファイル内容が不要になったら、 close() を呼び出して、ファイルをクローズします。

以下の例では、RecordListTablePane オブジェクトを作成し、 それをフレームに追加します。

```
                    // Create an RecordListTablePane
                    // object. Assume that "system" is
                    // an AS400 object that is created
                    // and initialized elsewhere.
RecordListTablePane tablePane = new RecordListTablePane (system,
"/QSYS.LIB/QIWS.LIB/QCUSTCDT.FILE");

                    // Load the file contents.
tablePane.load ();

                    // Add the table pane to a frame.
                    // Assume that "frame" is a JFrame
                    // created elsewhere.
frame.getContentPane ().add (tablePane);
```

RecordListTablePane および RecordListTableModel クラス

RecordListTablePane は、 モデル・ビュー・コントローラー・パラダイムを使用して実装されます。 このパラダイムでは、 データとユーザー・インターフェースが別々のクラスに分離されます。 この実装により、 RecordListTableModel が Java Foundation Classes (JFC) の JTable に統合されます。 RecordListTableModel クラスは、 ファイルの内容を検索および管理します。一方、 JTable はそのファイルの内容をグラフィックで表示し、 ユーザーとの対話を処理します。

RecordListTablePane は機能性が高く、 大抵の要件は満たすことができますが、JFC コンポーネントをそれ以上に制御することを呼び出し側が必要とする場合は、 RecordListTableModel を直接使用することにより、 別の GUI コンポーネントとの統合をカスタマイズすることができます。

RecordListTableModel を使用するためには、 システム・プロパティーと fileName プロパティーを設定する必要があります。 これらのプロパティーを設定するには、コンストラクターを使用するか、 setSystem() メソッドおよび setFileName() メソッドを使用します。 load() を使用すれば、ファイル内容を検索します。 ファイル内容が不要になったら、 close() を呼び出して、ファイルをクローズします。

以下の例では、RecordListTableModel オブジェクトを作成し、 それを JTable に表示します。

```
                    // Create a RecordListTableModel
                    // object. Assume that "system" is
                    // an AS400 object that is created
                    // and initialized elsewhere.
RecordListTableModel tableModel = new RecordListTableModel (system,
"/QSYS.LIB/QIWS.LIB/QCUSTCDT.FILE");

                    // Load the file contents.
tableModel.load ();

                    // Create a JTable for the model.
JTable table = new JTable (tableModel);

                    // Add the table to a frame. Assume
                    // that "frame" is a JFrame
                    // created elsewhere.
frame.getContentPane ().add (table);
```

ResourceListPane および ResourceListDetailsPane

ResourceListPane クラスおよび ResourceListDetailsPane クラスを使用して、リソース・リストをグラフィカル・ユーザー・インターフェース (GUI) で表示します。

- ResourceListPane は、リソース・リストの内容をグラフィカルな javax.swing.JList に表示します。リストに表示されるすべての項目は、リソース・リストからのリソース・オブジェクトを表します。
- ResourceListDetailsPane は、リソース・リストの内容を、グラフィカルな javax.swing.JTable に表示します。テーブル内のすべての行は、リソース・リストからのリソース・オブジェクトを表します。

ResourceListDetailsPane のテーブル列は列属性 ID の配列として指定されます。テーブルには、配列のそれぞれの要素ごとに列、それぞれのリソース・オブジェクトごとに行があります。

ポップアップ・メニューは、デフォルトでは、ResourceListPane および ResourceListDetailsPane の両方について使用可能です。

ほとんどのエラーは、例外が出されるのではなく、 com.ibm.as400.vaccess.ErrorEvents として報告されます。エラー状態を診断してリカバリーするために、ErrorEvents を listen します。

例: リソース・リストを GUI で表示する

この例は、システム上に全ユーザーの ResourceList を作成し、それを GUI で表示します (詳細ペイン):

```
// Create the resource list.
AS400 system = new AS400("MYSYSTEM", "MYUSERID", "MYPASSWORD");
RUserList userList = new RUserList(system);

// Create the ResourceListDetailsPane.  In this example,
// there are two columns in the table.  The first column
// contains the icons and names for each user.  The
// second column contains the text description for each
// user.
Object[] columnAttributeIDs = new Object[] { null, RUser.TEXT_DESCRIPTION };
ResourceListDetailsPane detailsPane = new ResourceListDetailsPane();
detailsPane.setResourceList(userList);
detailsPane.setColumnAttributeIDs(columnAttributeIDs);

// Add the ResourceListDetailsPane to a JFrame and show it.
JFrame frame = new JFrame("My Window");
frame.getContentPane().add(detailsPane);
frame.pack();
frame.show();
```

```
// The ResourceListDetailsPane will appear empty until
// we load it.  This gives us control of when the list
// of users is retrieved from the iSeries.
detailsPane.load();
```

システム状況クラス

vaccess パッケージ内のシステム状況コンポーネントを使用すると、 既存の AS400Panes を使用して GUI を作成することができます。 また、Java Foundation Classes (JFC) を使って独自の GUI を作成するためのオプションもあります。 VSystemStatus オブジェクトは、サーバーのシステム状況を表します。 VSystemPool オブジェクトは、 サーバー上のシステム・プールを表します。 VSystemStatusPane は、システム状況情報を表示するビジュアル・ペインを表します。

VSystemStatus クラスを使用すれば、 GUI 環境内でサーバー・セッションの状況に関する情報を取得することができます。

- getSystem() メソッドは、システム状況情報が含まれるサーバーを戻します。
- getText() メソッドは、記述テキストを戻します。
- setSystem() メソッドは、システム状況情報が存在するサーバーを設定します。

上記のメソッドに加えて、 GUI でシステム・プール情報にアクセスして変更を加えることもできます。

VSystemStatus は、 VSystemStatusPane と共に使用します。 VSystemPane は、システム状況およびシステム・プールの両方についての情報が表示されるビジュアル表示ペインです。

VSystemStatusPane クラス

VSystemStatusPane クラスを使用すれば、 Java プログラムでシステム状況およびシステム・プール情報を表示できます。

VSystemStatusPane には、以下のメソッドが含まれます。

- getVSystemStatus(): VSystemStatusPane に VSystemStatus 情報を戻す。
- setAllowModifyAllPools(): システム・プール情報が変更できるかどうかを決定する値を設定する。

以下の例は、VSystemStatusPane クラスを使用する方法を示しています。

```
// Create an as400 object.
AS400 mySystem = new AS400("mySystem.myCompany.com");

// Create a VSystemStatusPane
VSystemStatusPane myPane = new VSystemStatusPane(mySystem);

// Set the value to allow pools to be modified
myPane.setAllowModifyAllPools(true);

//Load the information
myPane.load();
```

システム値 GUI

vaccess パッケージ内のシステム値コンポーネントを Java プログラムで使用すると、 既存の AS400Panes を使用したり、 Java Foundation Classes (JFC) を使って独自のペインを作成したりして、GUI を作成することができます。 VSystemValueList オブジェクトは、サーバー上のシステム値リストを表します。

システム値 GUI コンポーネントを使用するには、コンストラクターを使用するか、 setSystem() メソッドを使用します。

例

以下の例では、AS400Explorer ペインを使ってシステム値 GUI を作成します。

```
//Create an AS400 object
AS400 mySystem = newAS400("mySystem.myCompany.com");
VSystemValueList mySystemValueList = new VSystemValueList(mySystem);
as400Panel=new AS400ExplorerPane((VNode)mySystemValueList);
//Create and load an AS400ExplorerPane object
as400Panel.load();
```

Vaccess ユーザーおよびグループ・クラス

vaccess パッケージ内のユーザーおよびグループ・コンポーネントがあれば、 VUser クラスを使ってサーバー・ユーザーおよびグループのリストを表すことができます。

以下のコンポーネントを使用できます。

- AS400Panes は、 1 つ以上のサーバー・リソースを表示したり操作したりするための GUI コンポーネントです。
- VUserList オブジェクトは、 AS400Panes での使用を目的とした、サーバー・ユーザーおよびグループのリストを表すリソースです。
- VUserAndGroup オブジェクトは、 AS400Panes での使用を目的としたリソースであり、サーバー・ユーザーのグループを表します。 これを使用すると、Java プログラムですべてのユーザーをリストしたり、 すべてのグループをリストしたり、あるいはグループのメンバーではないユーザーをリストしたりすることができます。

AS400Pane と VUserList オブジェクトを合わせて使用すれば、 リストの多くのビューを表示することができます。 また、ユーザーにユーザーやグループを選択させることもできます。

VUserList を使用するには、その前にシステム・プロパティーを設定しなければなりません。 このプロパティーを設定するには、コンストラクターを使用するか、 setSystem() メソッドを使用します。 その後、AS400Pane のコンストラクターまたは setRoot() メソッドを使用し、 ルートとして VUserList オブジェクトを AS400Pane に「接続」します。

VUserList には、 AS400Panes に表示されているユーザーやグループのセットを定義するのに役立つ他のプロパティーも含まれています。

- setUserInfo() メソッドを使用すれば、 表示するユーザーのタイプを指定することができます。
- setGroupInfo() メソッドを使用すれば、グループ名を指定することができます。

VUserAndGroup オブジェクトは、 システム上のユーザーおよびグループについての情報を取得するのに使用できます。 特定のオブジェクトに関する情報を取得するには、情報にアクセスするために、その情報をロードする必要があります。 情報を検出できるサーバーを表示するには、 getSystem メソッドを使用します。

AS400Pane オブジェクトおよび VUserList または VUserAndGroup オブジェクトは、 作成時にはデフォルト状態に初期設定されています。 この時点では、ユーザーおよびグループのリストはロードされていません。 ルート・ディレクトリーの内容をロードするためには、Java プログラムがどちらかのオブジェクトに対して load() メソッドを明示的に呼び出すことにより、 リストの内容を収集するサーバーへの通信を開始しなければなりません。

実行時に、ジョブ、ジョブ・リスト、 またはジョブ・ログ・メッセージを右マウス・ボタンでクリックして、 ショートカット・メニューを表示します。 ショートカット・メニューから「プロパティー」を選択して、 選択したオブジェクトに対するアクションを実行します。

- ユーザー - ユーザー情報のリストを表示します。ユーザー情報には、 記述、ユーザー・クラス、状況、ジョブ記述、出力情報、メッセージ情報、 国別情報、セキュ

リティー情報、グループ情報が含まれます。
- ユーザー・リスト - ユーザー情報およびグループ情報のプロパティーを扱います。リストの内容を変更することもできます。
- ユーザーおよびグループ - ユーザー名や記述などのプロパティーを表示します。

ユーザーがアクセスできるユーザーおよびグループは、 アクセスの許可を受けているユーザーとグループだけです。 さらに、Java プログラムは、 ペインに対して setAllowActions() メソッドを実行することにより、 ユーザーがアクションを実行できないようにすることができます。

以下の例では、 VUserList を作成し、それを AS400DetailsPane に表示します。

```
// Create the VUserList object.
// Assume that "system" is an AS400
// object created and initialized
// elsewhere.
VUserList root = new VUserList (system);

// Create and load an
// AS400DetailsPane object.
AS400DetailsPane detailsPane = new AS400DetailsPane (root);
detailsPane.load ();

// Add the details pane to a frame.
// Assume that "frame" is a JFrame
// created elsewhere.
frame.getContentPane ().add (detailsPane);
```

以下の例では、VUserAndGroup クラスを使用する方法を示します。

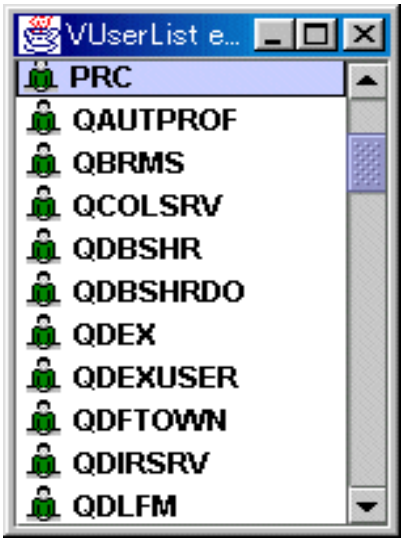
```
// Create the VUserAndGroup object.
// Assume that "system" is an AS400 object created and initialized elsewhere.
VUserAndGroup root = new VUserAndGroup(system);

// Create and Load an AS400ExplorerPane
AS400ExplorerPane explorerPane = new AS400ExplorerPane(root);
explorerPane.load();

// Add the explorer pane to a frame
// Assume that "frame" is a JFrame created elsewhere
frame.getContentPane().add(explorerPane);
```

他の例

AS400ListPane を VUserList オブジェクトと一緒に使用することにより、 システム上のユーザーのリストを表します。

以下のイメージでは、 VUserList グラフィカル・ユーザー・インターフェース・コンポー

ネントを示します。


VUserList e...
PRC
QAUTPROF
QBRMS
QCOLSRV
QDBSHR
QDBSHRDO
QDEX
QDEXUSER
QDFTOWN
QDIRSRV
QDLFM

Graphical Toolbox

Graphical Toolbox は、Java 形式のカスタム・ユーザー・インターフェース・パネルを作成するのに役立つ UI ツールのセットです。 Java のアプリケーション、アプレット、 あるいは iSeries ナビゲーターのプラグインにそのパネルを取り込むことができます。 iSeries や他のソース (ローカル・ファイル・システム中のファイルやネットワーク上のプログラムなど) から取得したデータをこのパネルに含めることができます。

GUI ビルダーは、Java ダイアログ、プロパティー・シート、 およびウィザードを作成するための WYSIWYG 表示形式エディターです。 GUI ビルダーを使用すると、 パネル上でユーザー・インターフェース・コントロールの追加、調整、または編集を行ってから、 そのパネルをプレビューしてレイアウトが望みどおりになっているかどうかを調べることができます。 作成したパネル定義をダイアログ内で使用したり、 プロパティー・シートやウィザードにパネルを挿入したり、 パネルを調整してスプリッター・ペイン、デック・ペイン、 およびタブ形式ペインの形式にしたりすることができます。 GUI ビルダーを使用すれば、 メニュー・バー、ツールバー、およびコンテキスト・メニュー定義を作成することもできます。 パネルには、コンテキストに依存したヘルプも含む、JavaHelp を組み込むことができます。

リソース・スクリプト・コンバーターは、 Windows のリソース・スクリプトを Java プログラムで使用できる XML 表現に変換します。 リソース・スクリプト・コンバーターを使用して、 既存の Windows のダイアログおよびメニューにある Windows リソース・スクリプト (RC ファイル) を処理することができます。 これらの変換済みファイルは、GUI ビルダーで編集できます。 プロパティー・シートおよびウィザードは、 リソース・スクリプト・コンバーターおよび GUI ビルダーを使用して、 RC ファイルから作成できます。

上記 2 つのツールの基礎となっているのは、パネル定義マークアップ言語、 つまり PDML という新しいテクノロジーです。 PDML は、拡張可能マークアップ言語 (XML) に基づいており、 ユーザー・インターフェース要素のレイアウトを記述するための、 プラットフォームに依存しない言語を定義します。 PDML でパネルを定義し終えたら、 Graphical Toolbox に備えられている実行時 API を使用してそれらのパネルを表示できます。 この API は、PDML の解釈およびユーザー・インターフェースの提供を Java Foundation Classes を使って行うことにより、 パネルを表示します。

Graphical Toolbox の利点

コードの減少と時間の節約

Graphical Toolbox を使用すると、Java ベースのユーザー・インターフェースを手早く簡単に作成できます。 GUI ビルダーにより、パネル上の UI 要素のレイアウトを正確に制御できます。 レイアウトは PDML で記述されるので、 ユーザー・インターフェースを定義するための Java コードを開発したり、 コードを再コンパイルして変更を加えたりする必要はありません。 その結果、Java アプリケーションの作成や保守に必要な時間を大幅に減らせます。 リソース・スクリプト・コンバーターにより、多数の Windows パネルを Java に手早く簡単に移行できます。

カスタム・ヘルプ

PDML でユーザー・インターフェースを定義することの利点は他にもあります。 パネルの情報はすべて正式なマークアップ言語に統合されるので、 これらのツールを拡張して開発者のために追加のサービスを実行できます。 たとえば、GUI ビルダーとリソース・スクリプト・コンバーターで、 パネルのオンライン・ヘルプの HTML による骨組みを生成できます。 必要なヘルプ・トピックを指定すると、必要に応じたそのへ

ルプ・トピックが自動的に作成されます。 ヘルプ・トピックのアンカー・タグがヘルプの骨組みに組み込まれているので、 ヘルプの作成者は内容の開発に専念できます。 ユーザーの要求に応じて、 正しいヘルプ・トピックが Graphical Toolbox の実行時環境に自動的に表示されます。

コードへのパネルの自動組み込み

また PDML タグにより、パネル上の個々のコントロールを JavaBean の属性と関連付けることができます。 データを備える bean クラスをパネルに通知し、属性を個々の該当するコントロールに関連付けがなされると、 ツールで bean オブジェクトの Java ソース・コードの骨組みを生成するように要求できます。 実行時に、Graphical Toolbox は、bean と、通知したパネル上のコントロールの間でデータを自動的に転送します。

プラットフォームに依存しない

Graphical Toolbox の実行時環境では、イベント処理、 ユーザー・データ妥当性検査、およびパネルの要素間の共通タイプの対話がサポートされます。 プラットフォーム上での正しいユーザー・インターフェースの外観は、 その使用しているオペレーティング・システムに基づいて自動的に設定されます。 GUI ビルダーによって外観をいろいろと切り替えれば、 さまざまなプラットフォーム上でパネルが表示される様子を評価できます。

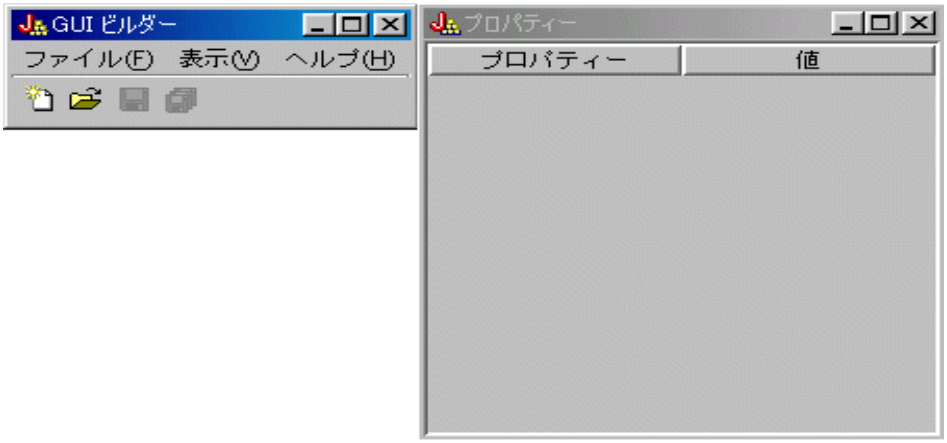
Graphical Toolbox の内部

Graphical Toolbox には 2 つのツールが備えられているので、 ユーザー・インターフェースの作成を自動化する方法は 2 つあることになります。 GUI ビルダーを使用して 1 から新しいパネルを作成することも手早く簡単にできます。また、 リソース・スクリプト・コンバーターを使用して既存の Windows ベースのパネルを Java に変換することもできます。 変換済みファイルは、GUI ビルダーで編集できます。 どちらのツールも国際化対応をサポートしています。

GUI ビルダー

初めて GUI ビルダーを起動する際、図 1 の 2 つのウィンドウが表示されます。

図 1: GUI ビルダーのウィンドウ

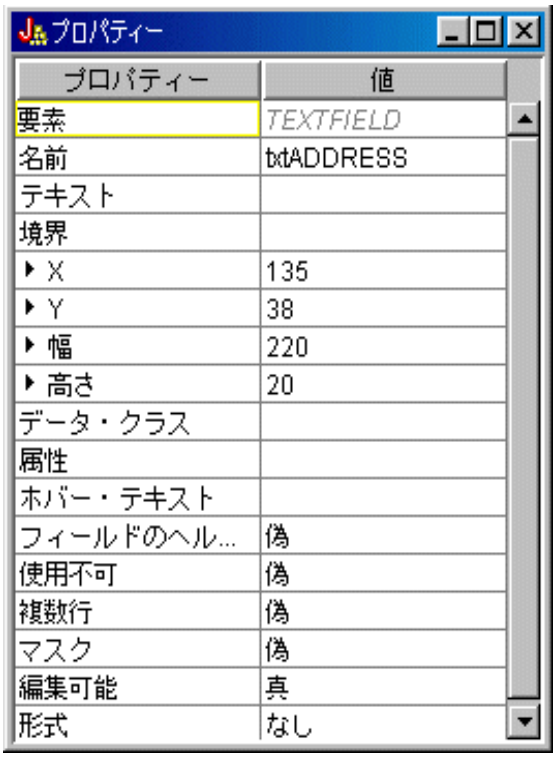


「ファイル・ビルダー」ウィンドウは、PDML ファイルを作成および編集するために使用します。

図 2: 「ファイル・ビルダー」ウィンドウ

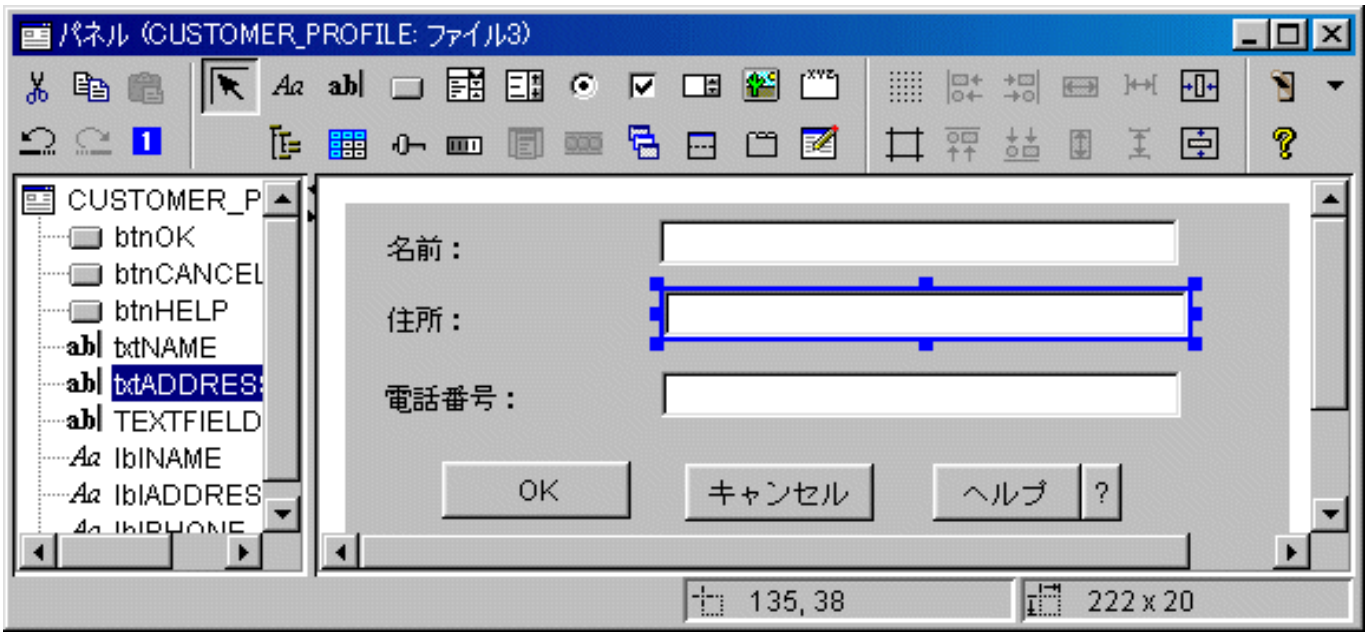
「プロパティー」ウィンドウを使用して、 現在選択されているプロパティーを表示したり変更を加えたりすることができます。

図 3: 「プロパティー」ウィンドウ



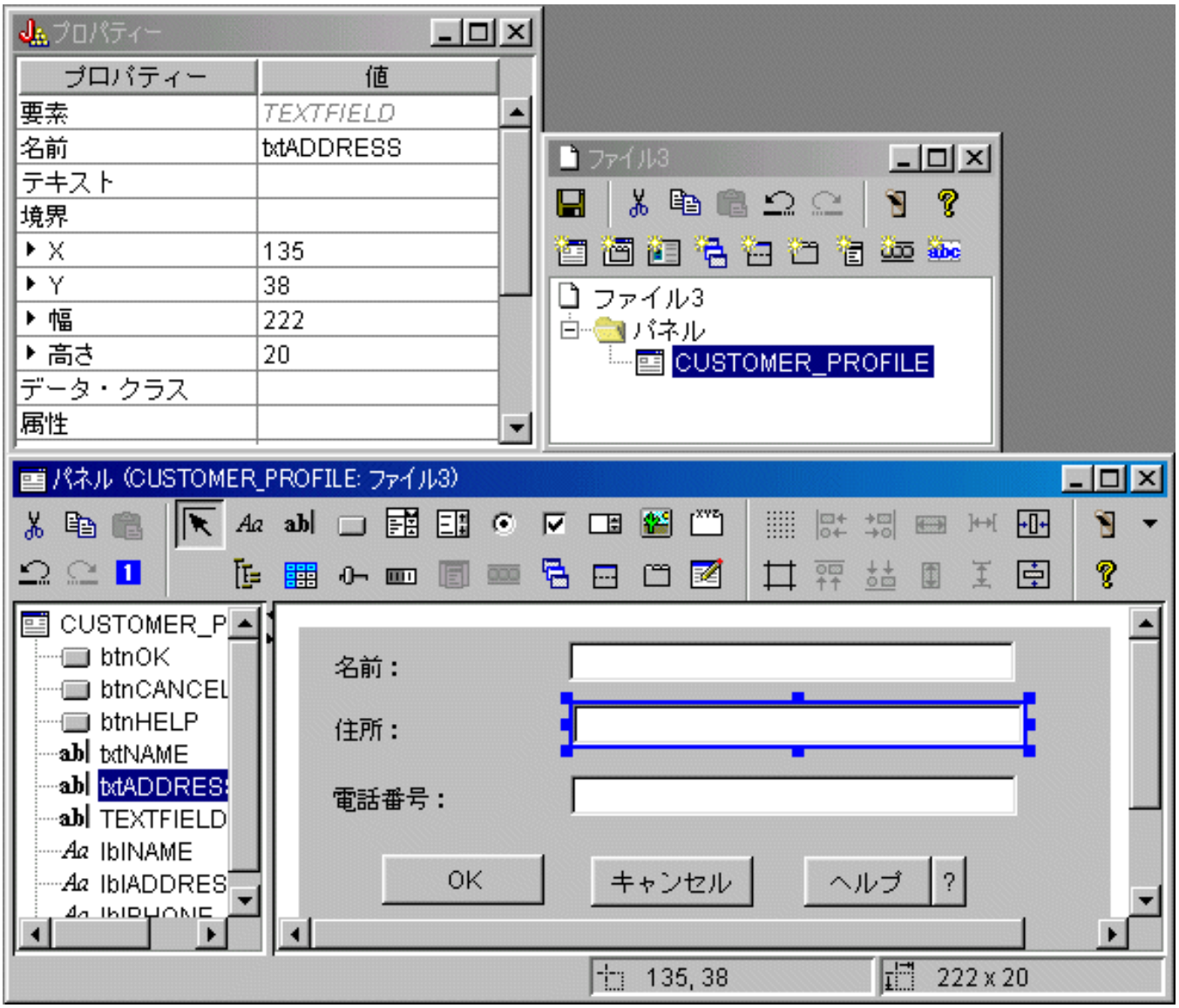
「パネル・ビルダー」ウィンドウを使用して、 グラフィカル・ユーザー・インターフェースのコンポーネントを編集できます。 ツールバーから希望のコンポーネントを選択して、 パネルをクリックして任意の場所にそれを配置します。 またツールバーには、コントロールのグループを位置合わせしたり、 パネルをプレビューしたり、GUI ビルダー機能のオンライン・ヘルプを要求したりする機能も備えられています。 それぞれのアイコンで行えることについての説明は、GUI ビルダーのパネル・ビルダー・ツールバーを参照してください。

図 4: 「パネル・ビルダー」ウィンドウ



編集対象のパネルが、「パネル・ビルダー」ウィンドウに表示されます。 図 5 は、各ウィンドウが協働する方法を示しています。

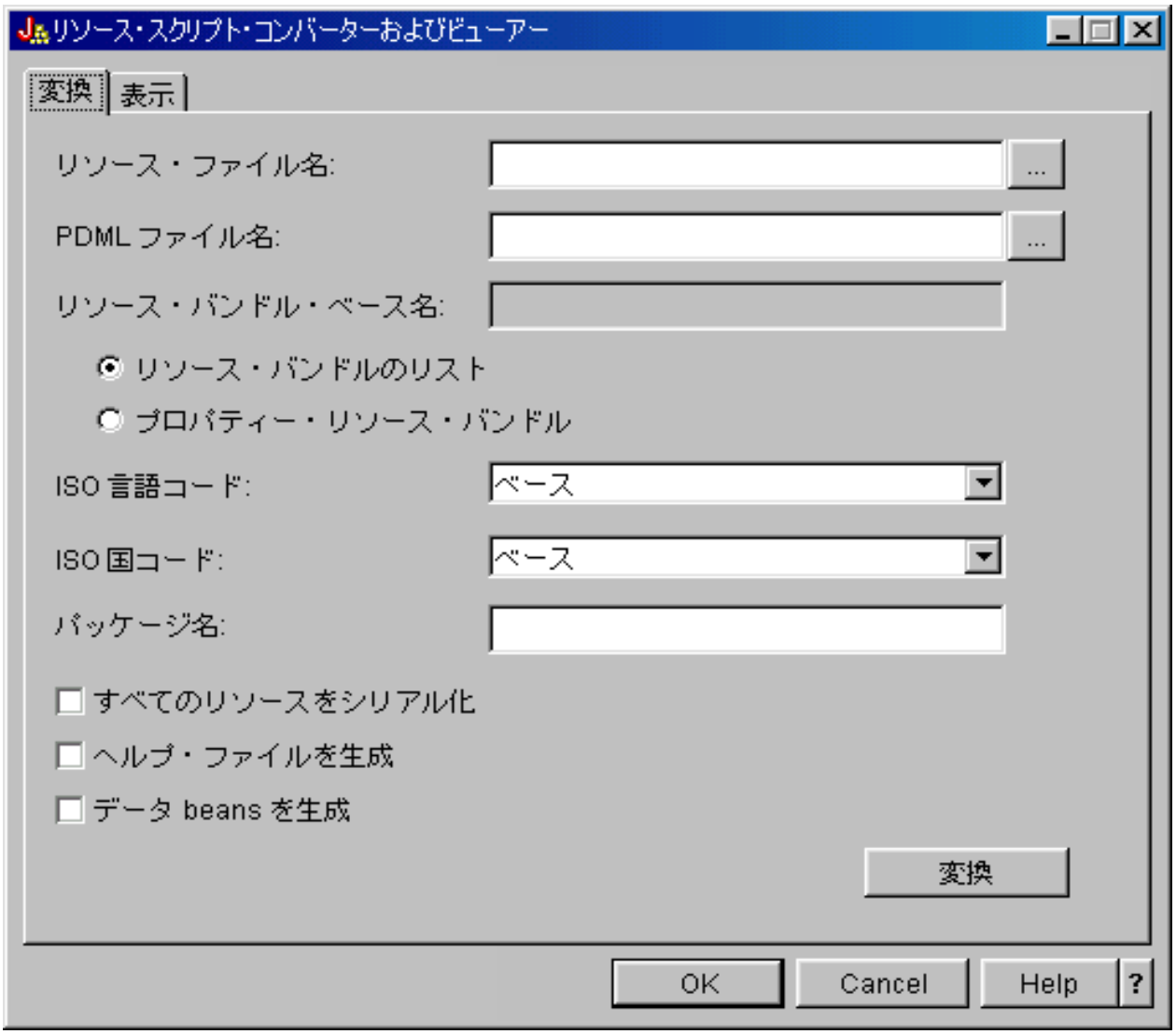
図 5: GUI ビルダーの各ウィンドウが協働する方法の例



リソース・スクリプト・コンバーター

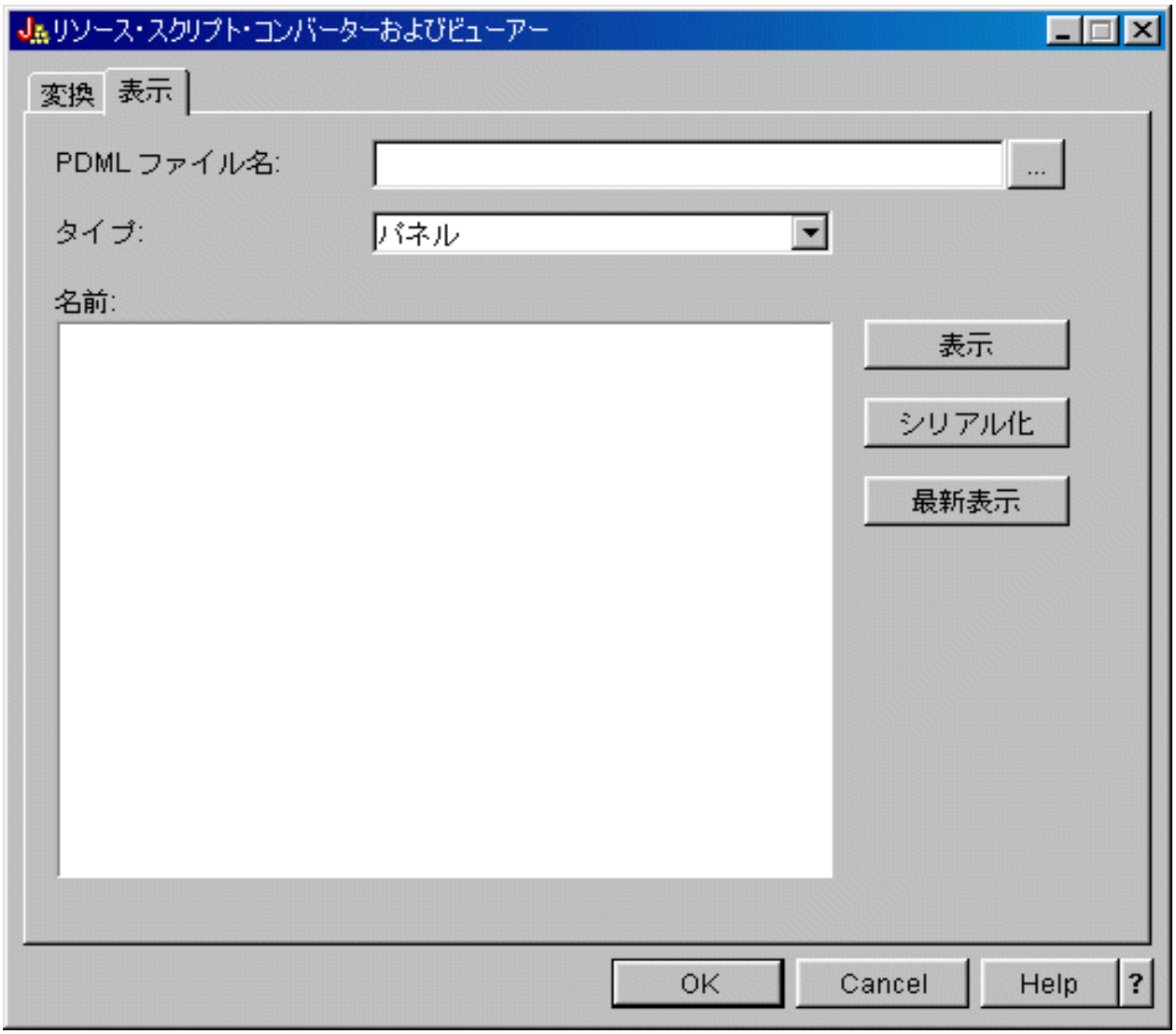
リソース・スクリプト・コンバーターは 2 つのペインから成るタブ形式のダイアログです。「変換」ペインには、PDML に変換する Microsoft または VisualAge for Windows の RC ファイルの名前を指定します。 目的の PDML ファイルの名前を指定して、 変換済みのパネル・ストリングが含まれる Java リソース・バンドルに関連付けることができます。 また、パネルのオンライン・ヘルプの骨組みを生成するよう要求したり、 パネルのデータがあるオブジェクトの Java ソース・コードの骨組みを生成したり、 パネル定義をシリアル化して実行時間のパフォーマンスを改善したりすることができます。 コンバーターのオンライン・ヘルプには、「変換」ペインの個々の入力フィールドに関する詳細な説明があります。

図 6: リソース・スクリプト・コンバーター「変換」ペイン



変換が正常に実行されたら、 「表示」ペインを使用して、 新しく作成された PDML ファイルの内容を表示したり、新しい Java パネルをプレビューしたりできます。 GUI ビルダーを使用して、必要に応じてパネルに多少の調整を加えることができます。 コンバーターは、変換を実行する前に必ず既存の PDML ファイルを検査し、 後でもう一度変換を実行する必要がある場合は変更内容をすべて保存しようとします。

図 7: リソース・スクリプト・コンバーター「表示」ペイン



Graphical Toolbox の入門

Graphical Toolbox についてさらに学ぶには、以下のトピックを利用してください。

- Graphical Toolbox の設定
- ユーザー・インターフェースの作成
- 実行時のパネルの表示
- オンライン・ヘルプ・ファイルの生成
- Graphical Toolbox の例
- ブラウザーで Graphical Toolbox を使用する
- パネル・ビルダー・ツールバー

Graphical Toolbox の設定

Graphical Toolbox は、 JAR ファイルのセットとして出荷されています。 Graphical Toolbox を設定するには、 JAR ファイルをワークステーション上にインストールしてから CLASSPATH 環境変数を設定する必要があります。

また、 ご使用のワークステーションが IBM Toolbox for Java 実行時の要件を満たしているかどうかを確認する必要もあります。

ワークステーションへの Graphical Toolbox のインストール

Graphical Toolbox を使用して Java プログラムを開発するには、 まず、ワークステーションに Graphical Toolbox の JAR ファイルをインストールします。 以下の方式のうち 1 つを使用してください。

JAR ファイルの転送

注: 次のリストでは、JAR ファイルを転送するために使用できるいくつかの方式を示します。 iSeries 上に IBM Toolbox for Java ライセンス・プログラムがインストールされている必要があります。 さらに、 Sun 社の JavaHelp Web サイトから、 JavaHelp、jhall.jar 用の JAR ファイルをダウンロードする必要があります。

- FTP を使用して (バイナリー・モードでファイルを転送してください)
  /QIBM/ProdData/HTTP/Public/jt400/lib ディレクトリー下にある JAR ファイルをワークステーション上のローカル・ディレクトリーにコピーします。
- IBM iSeries Access for Windows を使用してネットワーク・ドライブを割り当てます。
- IBM Toolbox for Java に付属の AS400ToolboxInstaller クラスを使って、 Graphical Toolbox の JAR ファイルをインストールすることもできます。 その場合、パッケージ名として OPNAV を指定します。 詳細については、クライアント・インストールおよびクラスの更新を参照してください。

iSeries Access for Windows を使用した JAR ファイルのインストール

iSeries Access for Windows のインストール時に Graphical Toolbox をインストールすることもできます。 現在 IBM Toolbox for Java は iSeries Access for Windows の一部として出荷されています。 初めて iSeries Access for Windows をインストールする場合、「カスタム・インストール」を選択し、 「インストール」メニューで「IBM Toolbox for Java」コンポーネントを選択します。 iSeries Access for Windows をインストール済みで、IBM Toolbox for Java をインストールしていない場合には、 選択セットアップ

・プログラムを使ってこのコンポーネントをインストールすることができます。

クラスパスの設定

Graphical Toolbox を使用するには、こうした JAR ファイルを CLASSPATH 環境変数に追加する (または、 コマンド行でクラスパス・オプションに使ってこれらを指定する) 必要があります。

たとえば、これらのファイルをワークステーションの C: \gtbox \lib ディレクトリーにコピーした場合、 以下のパス名をクラスパスに追加する必要があります。

```
C:/gtbox/lib/uitools.jar;
C:/gtbox/lib/jui400.jar;
C:/gtbox/lib/data400.jar;
C:/gtbox/lib/util400.jar;
C:/gtbox/lib/x4j400.jar;
C:/gtbox/lib/jhall.jar;
```

iSeries Access for Windows を使用して Graphical Toolbox をインストールした場合には、 iSeries Access for Windows をインストールしたドライブの \Program Files \Ibm \Client Access \jt400 \lib ディレクトリーに、 すべての JAR ファイル (jhall.jar を除く) があります。 iSeries Access for Windows は、 jhall.jar を \Program Files \Ibm \Client Access \jre \lib ディレクトリーにインストールします。 クラスパスのパス名に、これを反映させる必要があります。

JAR ファイルの説明

- uitools.jar GUI ビルダーおよびリソース・スクリプト・コンバーター・ツールが含まれます。
- jui400.jar Graphical Toolbox の実行時の API が含まれます。 Java プログラムは、この API を使用して、ツールを使って構成されたパネルを表示します。 これらのクラスは、アプリケーションを指定して再配布できます。
- data400.jar プログラム呼び出しマークアップ言語 (PCML) の実行時 API が含まれます。 Java プログラムは、この API を使用して、 PCML を使ってパラメーターと戻り値が識別される iSeries プログラムを呼び出します。 これらのクラスは、アプリケーションを指定して再配布できます。
- util400.jar iSeries データのフォーマットおよび iSeries メッセージの処理に使用するユーティリティー・クラスが含まれます。 これらのクラスは、アプリケーションを指定して再配布できます。

- x4j400.jar PDML および PCML 文書を解釈するために API クラスが使用する XML パーサーが含まれます。
- jhall.jar GUI ビルダーで作成するパネル用のオンライン・ヘルプおよびコンテキストに依存したヘルプを表示する、 JavaHelp クラスが含まれます。

注: GUI ビルダーの国際化対応バージョンおよびリソース・スクリプト・コンバーター・ツールを使用できます。 英語バージョン以外を実行するには、インストールする Graphical Toolbox の言語および国あるいは地域に応じた、uitools.jar の正しいバージョンを追加する必要があります。 これらの JAR ファイルは、 iSeries サーバーの /QIBM/ProdData/HTTP/Public/jt400/Mri29xx で使用することができます。 29xx は、言語および国あるいは地域に対応する 4 桁の OS/400 NLV コードです。 (さまざまな MRI29xx ディレクトリー内にある JAR ファイルの名前には、 2 文字の Java 言語コードと国あるいは地域別コードの接尾部が含まれます。) この追加の JAR ファイルは、uitools.jar より前のクラスパスに、 検索順に追加する必要があります。

Graphical Toolbox の使用

Graphical Toolbox のインストールが完了したら、 以下のリンクを順番に参照して、このツールの使用方法を学習してください。

- GUI ビルダーの使用
- リソース・スクリプト・コンバーターの使用

ユーザー・インターフェースの作成

GUI ビルダーの実行

GUI ビルダーを開始するには、 以下のようにして Java インタープリターを起動してください。

```
java com.ibm.as400.ui.tools.GUI Builder [-plaf look and feel]
```

Graphical Toolbox の JAR ファイルを含めるよう CLASSPATH 環境変数を設定しなかった場合は、 classpath オプションを使用して、コマンド行でこれらのファイルを指定する必要があります。 Graphical Toolbox の設定を参照してください。

オプション -plaf *look and feel*

プラットフォームの外観。 このオプションを指定すると、開発を行っている環境のプラットフォームに基づいて設定したデフォルトの外観が一時変更されるので、 パネルをプレビューしてさまざまなプラットフォームでの外観を調べることができます。以下の外観値を指定できます。

- Windows
- Metal
- Motif

Swing 1.1 がサポートするその他の外観値は、 現在 GUI ビルダーによってサポートされていません。

ユーザー・インターフェース・リソースのタイプ

初めて GUI ビルダーを開始するときは、新しい PDML ファイルを作成する必要があります。 GUI ビルダー・ウィンドウのメニュー・バーから、 「ファイル」 --> 「新規ファイル」を選択します。 新しい PDML ファイルを作成し終えたなら、 その中に以下のタイプの UI リソースがどれでも含まれるように定義できます。

パネル

基本的なリソース・タイプ。UI 要素を配置する四角形の区域を記述します。 UI 要素は単純なコントロール (ラジオ・ボタンやテキスト・フィールドなど) で構成することもできますし、 もう少し複雑なサブパネル (『分割ペイン』、 『デック・ペイン』、 および『タブ形式ペイン』の以下の定義を参照) で構成することもできます。 パネルでスタンドアロン型のウィンドウやダイアログのレイアウトを定義したり、 別の UI リソースに含まれるサブパネルの 1 つを定義したりできます。

メニュー

それぞれがテキスト列で表されている (「切り取り (Cut)」、「コピー (Copy)」、および「貼り付け (Paste)」は例外)、１つまたは複数の選択可能なアクションを含むポップアップ・ウィンドウ。 各アクションに対するニーモニックまたはアクセラレーター・キーの定義、 区切り記号および連鎖サブメニューの挿入、 または特殊なチェック項目やラジオ・ボタン・メニュー項目の定義を行うことができます。 メニュー・リソースは、独立型のコンテキスト・メニュー、 またはメニュー・バー内のドロップダウン・メニューとして使用できるほか、 メニュー・リソースそのものをパネル・リソースに関連したメニュー・バーとして定義することもできます。

ツールバー

使用可能な各ユーザー・アクションを表す、一連のプッシュボタンから構成されるウィンドウ。 各ボタンには、テキスト、アイコン、またはその両方を含めることができます。 ツールバーを浮動型として定義して、ユーザーがツールバーをパネルからスタンドアロン型のウィンドウにドラッグできるようにすることも可能です。

プロパティー・シート

タブ形式のパネルと「OK」、「キャンセル」、 および「ヘルプ」ボタンで構成されるスタンドアロン型のウィンドウまたはダイアログ。 個々のタブ形式ウィンドウのレイアウトは、パネル・リソースにより定義します。

ウィザード

事前定義済みの順序で表示される一連のパネルで構成され、 「戻る (Back)」、「次へ (Next)」、「キャンセル (Cancel)」、「終了 (Finish)」、および「ヘルプ (Help)」ボタンのあるスタンドアロン型のウィンドウまたはダイアログ。 ウィザード・ウィンドウ内のパネルの左側にタスクのリストが表示されることもあり、 これによってこのウィザード全体を通じて進行状況がトレースされます。

分割ペイン

スプリッター・バーで分割された 2 つのパネルで構成されるサブペイン。 パネルは横方向と縦方向のどちらにも配置できます。

タブ形式ペイン

タブ形式のコントロールを形成するサブペイン。 このタブ形式のコントロールは、 他のパネル、分割ペイン、またはデック・ペイン内に配置できます。

デック・ペイン

パネルの集合から構成されるサブペイン。 これらの内、同時に表示できるのは１つのパネルだけです。 たとえば、実行時にデック・ペインは指定のユーザー・アクションに応じて、 表示されているパネルを変更する

ことができます。

ストリング・テーブル

ストリング・リソースとそれに関連したリソース ID の集合。

生成されるファイル

パネルの変換可能ストリングは PDML ファイル自体には保管されず、 独立した Java リソース・バンドルに格納されます。 ツールではリソース・バンドルの定義の仕方を、 Java PROPERTIES ファイルか ListResourceBundle サブクラスのどちらにするかを指定できます。 ListResourceBundle サブクラスは変換可能リソースのコンパイル・バージョンで、 Java アプリケーションのパフォーマンスを向上させます。 しかし、保管操作のたびに ListResourceBundle がコンパイルされるので、 GUI ビルダーの保管処理は遅くなります。 したがって、ユーザー・インターフェースの設計が現状のままでよい場合は、 PROPERTIES ファイル (デフォルト設定) で作業を開始するのが最善です。

ツールを使用して、PDML ファイル中に個々のパネルの HTML の骨組みを生成できます。 実行時に、フォーカスがパネルのコントロールのいずれかにある時点でパネルの「ヘルプ」ボタンをクリックするか F1 を押すと、 該当するヘルプ・トピックが表示されます。 ヘルプの内容を、HTML 中の該当する箇所 (<!-- HELPDOC:SEGMENTBEGIN --> タグと <!-- HELPDOC:SEGMENTEND --> タグの有効範囲内) に挿入する必要があります。 詳細については、GUI ビルダーで生成したヘルプ・ドキュメントの編集を参照してください。

JavaBeans (パネルのデータを入力する) のソース・コードの骨組みを生成できます。 GUI ビルダーの「プロパティー」ウィンドウを使用して、 データを入れられるコントロールの DATACLASS プロパティーと ATTRIBUTE プロパティーに記入します。 DATACLASS プロパティーは bean のクラス名を識別し、 ATTRIBUTE プロパティーは bean クラスに実装されている gettor/settor メソッドの名前を指定します。 この情報を PDML ファイルに追加し終えたら、GUI ビルダーを使用して Java ソース・コードの骨組みを生成し、コンパイルできます。 実行時に、該当する gettor/settor メソッドが呼び出され、 パネルのデータが記入されます。

注: gettor/settor メソッドの数とタイプは、 そのメソッドに関連した UI コントロールのタイプに応じて変わります。 個々のコントロールのメソッドのプロトコルは、 DataBean クラスのクラス説明に記述されています。

最後の点として、PDML ファイルの内容をシリアル化できます。 シリアル化すると、ファイル中の UI リソースすべての簡単なバイナリー表示が作成されます。 シリアル化すると、パネルを表示する際に PDML ファイルを解釈する必要がなくなるので、 ユーザー・インターフェースのパフォーマンスは大幅に向

上します。

まとめ: MyPanels.pdml という名前のファイルを作成すると、 選択したツール・オプションに基づいて以下のファイルも作成されます。

- MyPanels.properties (リソース・バンドルを PROPERTIES ファイルとして定義した場合)
- MyPanels.java および MyPanels.class (リソース・バンドルを ListResourceBundle サブクラスとして定義した場合)
- PDML ファイル中に個々のパネルの <panel name>.html (オンライン・ヘルプの骨組みを生成することにした場合)
- DATACLASS プロパティーに指定した個々の固有な bean クラスの <dataclass name>.java および <dataclass name>.class (JavaBeans のソース・コードの骨組みを生成することにした場合)
- PDML ファイルに定義された個々の UI リソースの <resource name>.pdml.ser (内容をシリアル化することにした場合)

注: 条件付き動作機能 (SELECTED/DESELECTED) は、 パネル名が条件付き動作機能が付加されているパネル名と同じである場合には機能しません。 たとえば、FILE2 内の PANEL1 にあるフィールドを参照するフィールドに付加された条件付き動作参照が FILE1 内の PANEL1 にある場合、 その条件付き動作イベントは機能しません。 これを修正するには、 FILE2 内の PANEL1 をリネームしてから、 FILE1 内の条件付き動作イベントを更新してこの変更を反映するようにします。

リソース・スクリプト・コンバーターの実行

リソース・スクリプト・コンバーターを開始するには、 以下のようにして Java インタープリターを起動してください。

```
java com.ibm.as400.ui.tools.PDMLViewer
```

Graphical Toolbox の JAR ファイルを含めるよう CLASSPATH 環境変数を設定しなかった場合は、 classpath オプションを使用して、 コマンド行でこれらのファイルを指定する必要があります。 Graphical Toolbox の設定を参照してください。

以下のコマンドを使用して、 バッチ・モードでリソース・スクリプト・コンバーターを実行することもできます。

```
java com.ibm.as400.ui.tools.RC2XML file [options]
```

file は、処理されるリソース・スクリプト (RC ファイル) の名前です。 オプ

ション

-x *name*

生成される PDML ファイルの名前。 デフォルトは、処理される RC ファイルの名前です。

-p *name*

生成される PROPERTIES ファイルの名前。 デフォルトは PDML ファイルの名前です。

-r *name*

生成される ListResourceBundle サブクラスの名前。 デフォルトは PDML ファイルの名前です。

-package *name*

生成されるリソースが割り当てられるパッケージの名前。 このオプションを指定しないと、パッケージのステートメントは生成されません。

-l *locale*

リソースが生成されるロケール。 ロケールを指定しないと、該当する 2 文字の ISO 言語および国あるいは地域別コードの接尾部として、 生成されるリソース・バンドルの名前が付けられます。

-h

オンライン・ヘルプの HTML の骨組みを生成します。

-d

JavaBeans のソース・コードの骨組みを生成します。

-s

リソースをすべてシリアル化します。

Windows リソースの PDML へのマッピング

RC ファイル中にあるすべてのダイアログ、メニュー、およびストリング・テーブルは、 生成される PDML ファイル中で対応する Graphical Toolbox のリソースに変換されます。 Windows リソース の ID を作成する際に、単純な命名規則に従って、 新しい PDML ファイルにも適用される Windows コントロールの DATACLASS プロパティーと ATTRIBUTE プロパティーを定義することもできます。 これらのプロパティーは、変換の実行時に JavaBeans のソース・

コードの骨組みを生成するのに使用されます。

Windows リソース ID の命名規則は以下のとおりです。

IDCB_<class name>_<attribute>

<class name> は、コントロールの DATACLASS プロパティーとして指定しようとしている bean クラスの完全修飾名で、 <attribute> は、コントロールの ATTRIBUTE プロパティーとして指定しようとしている bean プロパティーの名前です。

たとえば、Windows テキスト・フィールドのリソース ID が IDCB_com_MyCompany_MyPackage_MyBean_SampleAttribute である場合は、 com.MyCompany.MyPackage.MyBean の DATACLASS プロパティーと SampleAttribute の ATTRIBUTE プロパティーが作成されます。 変換の実行時に JavaBeans を生成することにした場合は、 Java ソース・ファイル MyBean.java が作成され、 このファイルにはパッケージ・ステートメント package com.MyCompany.MyPackage と、 SampleAttribute プロパティーの gettor メソッドと settor メソッドが含まれます。

実行時のパネルの表示

Graphical Toolbox では、 PDML を使って定義されているユーザー・インターフェースを表示するために、 Java プログラムが使用できる再分配可能な API が提供されています。 この API は、PDML の解釈およびユーザー・インターフェースの提供を Java Foundation Classes を使って行うことにより、 パネルを表示します。

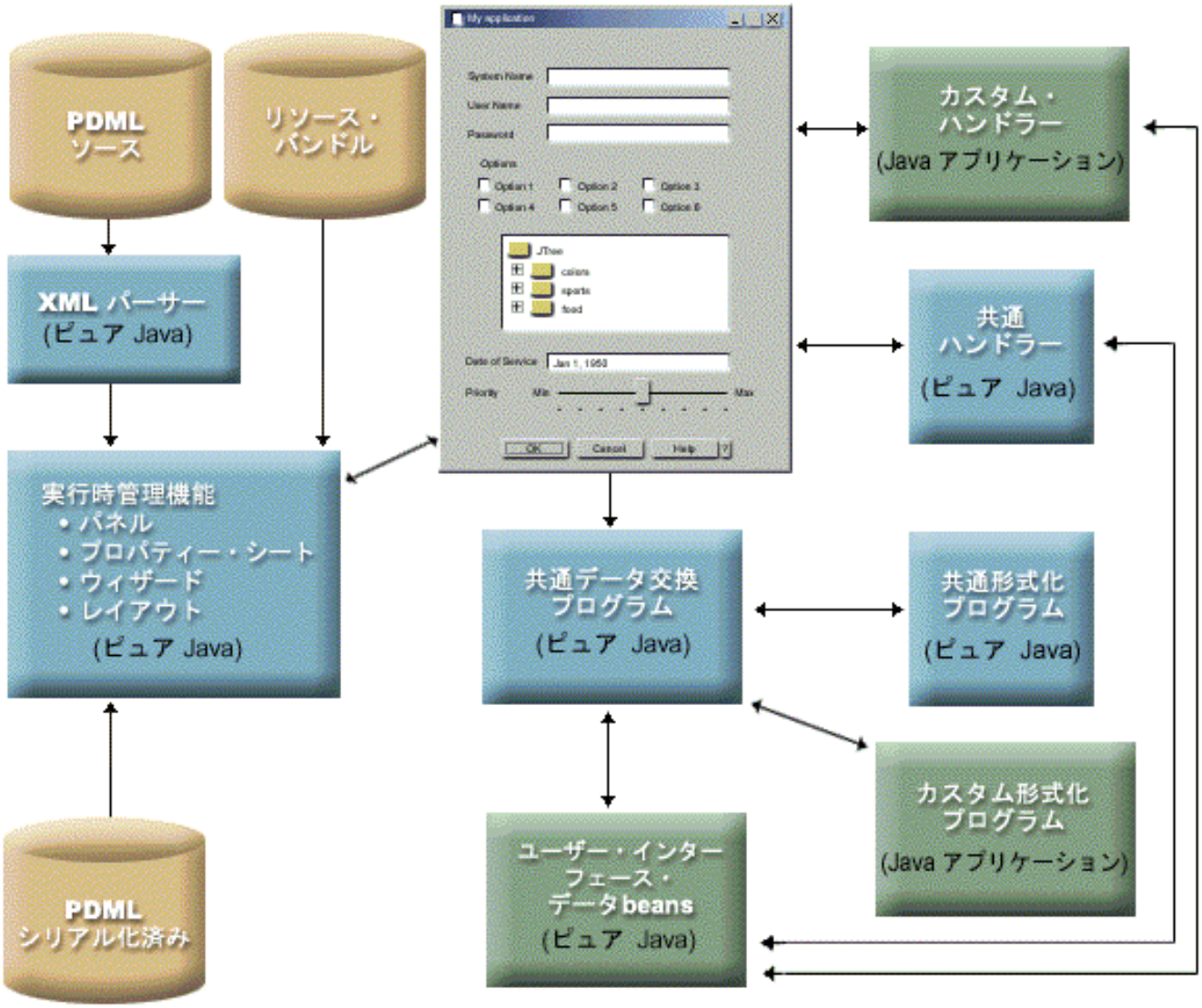
Graphical Toolbox の実行時環境は、以下のサービスを提供します。

- ユーザー・インターフェースの制御と PDML によって識別される JavaBeans との間のすべてのデータ交換処理。
- ユーザー・データ (共通の整数および文字データ・タイプ) の妥当性検査の実行、 およびユーザー設定による妥当性検査を導入できるようにするインターフェースの定義。 データが無効なものとして検出される場合、エラー・メッセージが表示されます。
- コミット、取り消し、およびヘルプ・イベントの処理を標準化する定義、 およびカスタム・イベントを処理するフレームワークの提供。
- PDML で定義された状態情報に基づく、ユーザー・インターフェースの制御間の対話の管理。 (たとえば、特定のラジオ・ボタンを選択すれば、制御のグループを使用禁止にすることができます。)

com.ibm.as400.ui.framework.java パッケージには、 Graphical Toolbox の実行時 API が含まれています。

Graphical Toolbox の実行時環境の要素が、図 1 に表示されています。 Java プログラムは、「実行時管理機能」ボックスにある 1 つかそれ以上のオブジェクトのクライアントです。

図 1: Graphical Toolbox の実行時環境



例

MyPanel が TestPanels.pdml ファイルで定義されていて、 TestPanels.properties プロパティー・ファイルがそのパネル定義に関連付けられているとします。 両方のファイルとも、 com/ourCompany/ourPackage ディレクトリーにあります。 このディレクトリーには、クラスパスで定義されるディレクトリー、 またはクラスパスで定義される ZIP または JAR ファイルの両方からアクセスすることができます。

例: パネルを作成および表示する

以下のコードで、パネルを作成し、表示します。

```
import com.ibm.as400.ui.framework.java.*;

// Create the panel manager. Parameters:
// 1. Resource name of the panel definition
// 2. Name of panel
```

```
// 3. List of DataBeans omitted

PanelManager pm = null;
try {
  pm = new PanelManager("com.ourCompany.ourPackage.TestPanels",
                        "MyPanel", null);
}

catch (DisplayManagerException e) {
  e.displayUserMessage(null);
  System.exit(-1);
}

// Display the panel
pm.setVisible(true);
```

例: ダイアログを作成する

パネルにデータを提供する DataBeans が実装され、その属性が PDML によって識別されると、完全な機能を備えたダイアログを構成するために、以下のコードを使用することができます。

```
import com.ibm.as400.ui.framework.java.*;
import java.awt.Frame;

// Instantiate the objects which supply data to the panel
TestDataBean1 db1 = new TestDataBean1();
TestDataBean2 db2 = new TestDataBean2();

// Initialize the objects
db1.load();
db2.load();

// Set up to pass the objects to the UI framework
DataBean[] dataBeans = { db1, db2 };

// Create the panel manager. Parameters:
// 1. Resource name of the panel definition
// 2. Name of panel
// 3. List of DataBeans
// 4. Owner frame window

Frame owner;
...
PanelManager pm = null;
try {
  pm = new PanelManager("com.ourCompany.ourPackage.TestPanels",
                        "MyPanel", dataBeans, owner);
}

catch (DisplayManagerException e) {
```

```
  e.displayUserMessage(null);
  System.exit(-1);
}

// Display the panel
pm.setVisible(true);
```

例: 動的パネル管理機能を使用する

新規のサービスが既存のパネル管理機能に追加されました。 動的なパネル管理機能では、 実行時のパネルのサイズが動的に変更されます。 動的パネル管理機能を使用した MyPanel の例を以下に示します。

```
import com.ibm.as400.ui.framework.java.*;

// Create the dynamic panel manager. Parameters:
// 1. Resource name of the panel definition
// 2. Name of panel
// 3. List of DataBeans omitted

DynamicPanelManager dpm = null;
try {
  pm = new DynamicPanelManager("com.ourCompany.ourPackage.TestPanels",
                               "MyPanel", null);
}

catch (DisplayManagerException e) {
  e.displayUserMessage(null);
  System.exit(-1);
}

// Display the panel
pm.setVisible(true);
```

このパネル・アプリケーションをインスタンス化すると、 パネルの動的サイズ変更機能を見ることができます。 カーソルを GUI 表示の端に移動すると、サイズ変更の矢印が表示され、 これを使用してパネルのサイズを変更できます。

GUI ビルダーによって生成されたヘルプ文書の編集

各 PDML プロジェクト・ファイルごとに、 GUI ビルダーはヘルプの骨組みを生成してから、それを単一の HTML 文書に入れます。 使用する前に、 この HTML ファイルは、PDML プロジェクトの各ダイアログごとに、 単一トピックの HTML ファイルに分割されます。 これにより、ユーザーは各トピックごとに細分化されたヘルプを利用でき、 管理者は少数の大きなヘルプ・ファイルを管理するだけですみます。

ヘルプ文書は有効な HTML ファイルなので、 任意のブラウザーで表示でき、ほとんどの HTML エディターで編集できます。 ヘルプ文書のセクションを定義するタグは注記内に組み込まれるので、ブラウザーでは表示されません。 注記タグは、ヘルプ文書を以下のようなセクションに分割するのに使用します。

- ヘッダー
- 各ダイアログごとのトピック・セクション
- ヘルプに対応した各コントロールごとのトピック・セクション
- フッター

さらに、フッターの前に追加のトピック・セクションを追加して、 追加情報または共通情報を提供することもできます。 トピック・セクションには、 分割されてヘッダーおよびフッターが作成されるまで、 HTML 本文しかありません。 ヘルプ文書が分割されると、 処理プログラムによってトピック・セクションにヘッダーおよびフッターが追加され、 完全な HTML ファイルが完成します。 ヘルプ文書のヘッダーおよびフッターは、 デフォルトのヘッダーおよびフッターとして使用されます。 ただし、デフォルト・ヘッダーを独自のヘッダーに一時変更することもできます。

ヘルプ文書の内部

以下のセクションでは、ヘルプ文書のパーツを説明します。

ヘッダー

ヘッダー・セクションの最後は、 以下のようなタグで示されます。
<!-- HELPDOC:HEADEREND -->
分割時に個々のトピックすべてのデフォルト・ヘッダーを一時変更したい場合には、 HEADER キーワードを使用して、 組み込む HTML フラグメントの名前を指定します。たとえば、次のようにします。
<!-- HELPDOC:HEADEREND HEADER="defaultheader.html" -->

トピック・セグメント

各トピックは、次のようなタグで囲まれます。
<!-- HELPDOC:SEGMENTBEGIN --> and <!-- HELPDOC:SEGMENTEND -->
SEGMENTBEGIN タグの直後には、 セグメント名を指定するアンカー・タグがあります。 このタグは、ヘルプ文書が分割されるときに作成される HTML 文書のファイル名も提供します。 セグメントの名前は、 パネル識別コード、コントロール識別コード、 および将来のファイル拡張子 (html) を組み合わせたものです。 たとえば、パネルの "MY_PANEL.MY_CONTROL.html" セグメントには、 パネル識別コードと将来のファイル拡張子しかありません。

ヘルプ生成プログラムは、 ヘルプ情報をどこに入れるべきかを示すテキストを文書に書き込みます。

```
<!-- HELPDOC:SEGMENTBEGIN PDMLSYNCH="YES" --><A NAME="MY_PANEL.MY_CONTROL.html"></A>
<H2>My favorite control</H2>
Insert help for "My favorite control" here.
<P><!-- HELPDOC:SEGMENTEND -->
```

アンカー・タグと SEGMENTEND タグの間には、 必要に応じて追加の HTML 2.0 タグを追加でき

ます。

PDMLSYNCH タグは、 セグメントが PDML で定義されたコントロールにどれだけ関連付けられるかを制御します。 PDMLSYCH が "YES" である場合には、 同じ名前のコントロールが PDML で除去されるとヘルプ文書セグメントも除去されます。 PDMLSYNCH="NO" である場合には、 PDML に対応するコントロールが存在してもしなくてもヘルプ文書にそのままトピックが残されます。 この設定は、詳細を示す追加トピックや共通トピックを作成する際に使用されます。

パネル用に生成されたヘルプには、 パネルのヘルプ用に使用可能にされた各コントロールへのリンクがあります。 これらのリンクはローカルのアンカー参照とともに生成されるので、これらを標準ブラウザーで内部リンクとしてテストできます。 ヘルプ文書が分割されると、処理プログラムはこれらの内部リンクの "#" を除去して その結果生成される単一トピック HTML ファイルの外部リンクにします。 トピック内に内部リンクを持つほうがよい場合もあるので、 参照されているファイルに ".html" が組み込まれている場合にのみ処理プログラムは前述の "#" を除去します。

特定のトピックのデフォルト・ヘッダーを一時変更したい場合には、 HEADER キーワードを使用して、組み込む HTML フラグメントの名前を提供します。 たとえば、次のようにします。
<!-- HELPDOC:SEGMENTBEGIN PDMLSYNCH="YES" HEADER="specialheader.html" -->

フッター

ヘルプ文書のフッターは、 次のようなタグで始まります。
<!-- HELPDOC:FOOTERBEGIN -->. 標準フッターは、 </BODY></HTML> です。 このフッターは、各 HTML ファイルに追加されます。

リンクの追加

リンクはすべての外部または内部 URL とその他のセグメントに追加できます。 しかし、以下のようないくつかの規則に従う必要があります。

- 外部 URL は標準的な方法で使用される。 これには、外部 URL への内部リンクが含まれます。
- 同じトピック内の内部リンクは標準的な方法で作成されるが、 ".html" がタグ名の一部であってはならない。 これは、トピックが別々になっている際に、 .html を持つリンクは外部リンクであるとヘルプ文書処理プログラムが見なすからです。 ですから、この場合、先行する "#" は除去されます。
- 他のトピック・セグメントへのリンクは、 内部アンカー参照のように "#" を前に付けて作成する必要がある。
- 他のトピック・セグメントへの内部リンクも作成できる。 処理では先行する "#" が除去されます。

注:

- 実行時には、 PanelManager クラスは PDML ファイルと同じ名前を持つヘルプ・ファイルをサブディレクトリー内で探します。 処理プログラムがヘルプ文書を分割する際には、 このサブディレクトリーがデフォルトで作成され、 分割した結果生成される HTML ファイルがその中に置かれます。
- 処理プログラムは、相対リンクである外部 URL 参照を調整することはありません。 個々のトピック・ファイルからリンクする際、 すべての相対リンクは新しいサブディレクトリーから検索します。 ですから、画像などのリソースは検索される位置に置くか、 パス内で "../" を使用してパネル・ディレクトリーから検索する必要があります。

ビジュアル・エディターを使用した編集

ヘルプの内容は、ほとんどのビジュアル HTML エディターで編集できます。 HELPDOC タグは注記なので、一部のエディターではそれを見分けられない可能性があります。 便利なように、 ヘルプの骨組みの SEGMENTBEGIN タグの直前と SEGMENTEND タグの直後に水平けい線が追加されています。 これらの水平けい線は、ビジュアル・エディターでセグメント全体の位置を明確に示します。 セグメントを移動、コピー、または削除するために選択する際には、 セグメントを囲む水平けい線を選択して、 SEGMENTBEGIN と SEGMENTEND タグも選択範囲にあることを確認してください。 これらの水平けい線は、最終的に生成される個々の HTML ファイルにはコピーされません。

追加トピックの作成

ヘルプ文書内で追加のトピック・セグメントを作成することができます。 多くの場合、他のセグメントをコピーするのがそれを行う最も簡単な方法です。 セグメントをコピーする際には、 SEGMENTBEGIN タグの直前と SEGMENTEND タグの直後の水平けい線をコピーする必要があります。 これにより、将来のビジュアル編集が容易になり、 タグの不一致を避ける助けにもなります。 最善の結果を得るために、 以下のヒントに従ってください。

- アンカーの名前は、 ヘルプ文書が分割される際に生成される単一ファイルの名前である必要がある。 このファイルは ".html" で終わる必要があります。
- ヘルプの骨組みが再生成される際にセグメントが除去されないように、 SEGMENTBEGIN タグの PDMLSYNCH="NO" キーワードを使用する必要がある。
- 新しいトピックへの参照は、 "#" で先行したヘルプ文書の内部リンクとして作成されます。 この "#" は、後の処理でセグメントが単一ファイルに分割されるときに消去されます。

リンクの検査

ほとんどの文章では、 Web ブラウザーで文書を表示してからさまざまなリンクを選択することによってリンクを検査できます。 単一のヘルプ文書内では、リンクはまだ内部形式のままです。

リンクの完了が近づいている場合、 またはヘルプを開発しているアプリケーションでテストしたい場合には、 ヘルプ文書を単一ファイルに分割する必要があります。 そのためには、ヘルプ文書を HTML 化する処理を行います。

ヘルプ文書を編集した後に再生成する必要がある場合、 作成した文章は保存されます。 元のヘルプの骨組みを生成した後に新しいコントロールを追加した場合には、 ヘルプ文書を再生成することができます。 このようなときには、 ヘルプ生成プログラムは新しい骨組みを作成する前にヘルプ文書が既に存在するかどうか検査します。 ヘルプ文書が見つかった場合、 生成プログラムは既存のセグメントを保存してから新しいコントロールを追加します。

Graphical Toolbox の例

Graphical Toolbox 内のツールを独自の UI プログラムに実装する方法を示す例が提供されています。

- パネルの作成および表示: 簡単なパネルを作成する方法を示します。 この例ではその後、 そのパネルを表示する小さな Java アプリケーションの作成方法を示します。 ユーザーがテキスト・フィールドにデータを入力して「閉じる (Close)」ボタンをクリックすると、 アプリケーションによりデータが Java コンソールにエコー表示されます。 この例は全体として、Graphical Toolbox 環境の基本機能および操作を示しています。
- パネルの作成および表示: パネルとプロパティー・ファイルとが同じディレクトリーにある場合の、 パネルの作成および表示方法を示します。
- 完全な機能を備えたダイアログの作成: パネルにデータを提供する DataBeans が実装され、 その属性が PDML によって識別された後で、 完全な機能を備えたダイアログを構成する方法を示します。
- 動的なパネル管理機能を使用したパネルのサイズ設定: 動的なパネル管理機能により、 実行時のパネルのサイズを動的に変更します。
- 編集可能コンボ・ボックス: 編集可能コンボ・ボックス用のデータ bean コーディングの例を示します。

以下の例は、次のものを作成するために GUI ビルダーがどのように役立つかを示しています。

- パネル: サンプル・パネル、およびそのパネルを実行するデータ bean コードの作成方法を示します。
- デック・ペイン: デック・ペインの作成方法、 および完成したデック・ペインがどのようなものかを示します。
- プロパティー・シート: プロパティー・シートの作成方法、 および完成したプロパティー・シートがどのようなものかを示します。
- 分割ペイン: 分割ペインの作成方法、 および完成した分割ペインがどのようなものかを示します。
- タブ形式ペイン: タブ形式ペインの作成方法、 および完成したタブ形式ペインがどのようなものかを示します。
- ウィザード: ウィザードの作成方法、 および完成した製品がどのようなものかを示します。
- ツールバー: ツールバーの作成方法、 および完成したツールバーがどのようなものかを示します。
- メニュー・バー: メニュー・バーの作成方法、 および完成したメニュー・バーがどのようなものかを示します。

- ヘルプ: ヘルプ文書の作成方法、 およびヘルプ文書をトピック・ページに分割する方法を示します。 GUI ビルダーによって生成されたヘルプ文書の編集も参照してください。
- サンプル: パネル、プロパティー・シート、ウィザード、 選択/選択解除オプション、およびメニュー・オプションを含む、 PDML プログラムの全体がどのようなものかを示します。

以下の特記事項は、IBM Toolbox for Java のすべての例に適用されます。

コードの特記事項情報

IBM は、お客様に、すべてのプログラミング・コード・サンプルを使用することができる非独占的な使用権を許諾します。 お客様は、このサンプル・コードから、 お客様独自の特別のニーズに合わせた類似のプログラムを作成することができます。

すべてのサンプル・コードは、例として示す目的でのみ、IBM により提供されます。 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。 したがって 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。

ここに含まれるすべてのプログラムは、現存するままの状態で提供され、いかなる保証条件も適用されません。 第三者の権利の不侵害、商品性、特定目的適合性に関する黙示の保証の適用も一切ありません。

ブラウザーで Graphical Toolbox を使用する

Graphical Toolbox を使用すると、Web ブラウザー上で稼働する Java アプレットのパネルを構築することができます。 このセクションでは、Graphical Toolbox の例をブラウザー上にて使用できるシンプルなパネルに変換する方法を説明します。 サポートされている最低限のブラウザー・レベルは、Netscape 4.05 および Internet Explorer 4.0 です。 ブラウザーごとの特異性にとらわれないようにするには、 Sun 社の Java プラグインを使ってアプレットを実行することをお勧めします。 それ以外の場合は、Netscape には署名付き JAR ファイルを、 Internet Explorer には別々の署名付き CAB ファイルをそれぞれ作成する必要があります。

アプレットの作成

アプレットでパネルを表示するコードは、 Java アプリケーションの例で使われているコードとほとんど同じですが、まず、 そのコードを JApplet サブクラスの init メソッドに再パッケージする必要があります。 また、そのパネルの PDML 定義で指定されたディメンションに、アプレット・パネルが分類されるようにするため、コードをいくつか追加する必要もあります。 以下に示すのは、サンプルのアプレット SampleApplet.java のソース・コードです。

```
import com.ibm.as400.ui.framework.java.*;

import javax.swing.*;
import java.awt.*;
import java.applet.*;
import java.util.*;

public class SampleApplet extends JApplet
{
    // The following are needed to maintain the panel's size
    private PanelManager        m_pm;
    private Dimension           m_panelSize;

    // Define an exception to throw in case something goes wrong
    class SampleAppletException extends RuntimeException {}

    public void init()
    {
        System.out.println("In init!");

        // Trace applet parameters
        System.out.println("SampleApplet code base=" + getCodeBase());
        System.out.println("SampleApplet document base=" + getDocumentBase());

        // Do a check to make sure we're running a Java virtual machine that's compatible with Swing 1.1
```

```
    if (System.getProperty("java.version").compareTo("1.1.5") < 0)
        throw new IllegalStateException("SampleApplet cannot run on Java VM version " +
                                        System.getProperty("java.version") + " - requires 1.1.5 or higher");

    // Instantiate the bean object that supplies data to the panel
    SampleBean bean = new SampleBean();

    // Initialize the object
    bean.load();

    // Set up to pass the bean to the panel manager
    DataBean[] beans = { bean };

    // Update the status bar
    showStatus("Loading the panel definition...");

    // Create the panel manager. Parameters:
    // 1. PDML file as a resource name
    // 2. Name of panel to display
    // 3. List of data objects that supply panel data
    // 4. The content pane of the applet

    try { m_pm = new PanelManager("MyGUI", "PANEL_1", beans, getContentPane()); }
    catch (DisplayManagerException e)
    {
        // Something didn't work, so display a message and exit
        e.displayUserMessage(null);
        throw new SampleAppletException();
    }

    // Identify the directory where the online help resides
    m_pm.setHelpPath("http://MyDomain/MyDirectory/");

    // Display the panel
    m_pm.setVisible(true);
}

public void start()
{
    System.out.println("In start!");
```

```
        // Size the panel to its predefined size
        m_panelSize = m_pm.getPreferredSize();
        if (m_panelSize != null)
        {
            System.out.println("Resizing to " + m_panelSize);
            resize(m_panelSize);
        }
    else
            System.err.println("Error: getPreferredSize returned null");
    }

    public void stop()
    {
        System.out.println("In stop!");
    }

    public void destroy()
    {
        System.out.println("In destroy!");
    }

    public void paint(Graphics g)
    {
        // Call the parent first
        super.paint(g);

        // Preserve the panel's predefined size on a repaint
        if (m_panelSize != null)
            resize(m_panelSize);
    }
}
```

アプレットのコンテンツ・ペインは、レイアウト用のコンテナーとして、 Graphical Toolbox に渡されます。 start メソッドで、アプレットのペインを正しいサイズに調整し、 そのパネル・サイズを保管するために、ブラウザー・ウィンドウのサイズ変更時に paint メソッドをオーバーライドします。

Graphical Toolbox をブラウザーで実行している時には、 JAR ファイルからパネルのオンライン・ヘルプ用 HTML ファイルにアクセスすることはできません。 それらのファイルは、アプレットが入っているディレクトリーに独立したファイルとして存在している必要があります。 PanelManager.setHelpPath を呼び出すと、 このディレクトリーが Graphical Toolbox に識別されるため、ヘルプ・ファイルを配置することができます。

HTML タグ

正しい水準の Java 実行環境を提供するために Sun 社の Java プラグインの使用をお勧めしましたが、 その場合は、Graphical Toolbox のアプレットを識別する HTML ファイルが期待どおりに動作しません。 しかし、幸いなことに、他のアプレット用にわずかな変更を加えるだけで、 同じ HTML テンプレートを再利用することができます。 マークアップは、Netscape Navigator と Internet Explorer の両方で解釈されるように設計されており、 ご使用のマシンにインストールされていない場合には、 Sun 社の Web サイトから Java プラグインのダウンロードを行うプロンプトが生成されます。 Java プラグインの動作の詳細については、 Java プラグイン HTML 仕様書を参照してください。

以下に示すのは、MyGUI.html ファイルにある、サンプル・アプレットの HTML です。

```
<html>

<head>
<title>Graphical Toolbox Demo</title>
</head>

<body>
<h1>Graphical Toolbox Demo Using Java(TM) Plug-in</h1>
<p>

<!-- BEGIN JAVA(TM) PLUG-IN APPLET TAGS -->

<!-- The following tags use a special syntax which allows both Netscape and Internet Explorer to load   -->
<!-- the Java Plug-in and run the applet in the Plug-in's JRE.  Do not modify this syntax.               -->
<!-- For more information see http://java.sun.com/products/jfc/tsc/swingdoc-current/java_plug_in.html.  -->

<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
        width="400"
        height="200"
        align="left"
codebase="http://java.sun.com/products/plugin/1.1.3/jinstall-113-win32.cab#Version=1,1,3,0">
    <PARAM name="code"     value="SampleApplet">
    <PARAM name="codebase" value="http://www.mycompany.com/~auser/applets/">
    <PARAM name="archive"  value="MyGUI.jar,jui400.jar,util400.jar,x4j400.jar">
    <PARAM name="type"     value="application/x-java-applet;version=1.1">

    <COMMENT>
    <EMBED type="application/x-java-applet;version=1.1"
           width="400"
           height=200"
```

```
            align="left"
            code="SampleApplet"
            codebase="http://www.mycompany.com/~auser/applets/"
            archive="MyGUI.jar,jui400.jar,util400.jar,x4j400.jar"
            pluginspage="http://java.sun.com/products/plugin/1.1.3/plugin-install.html">
          <NOEMBED>
      </COMMENT>
          No support for JDK 1.1 applets found!
          </NOEMBED>
      </EMBED>
</OBJECT>

<!-- END JAVA(TM) PLUG-IN APPLET TAGS -->

<p>
</body>
</html>
```

バージョン情報が 1.1.3 に設定されている必要があります。

注: この例では、XML パーサー の JAR ファイルである、 x4j400.jar を Web サーバーに格納する選択をしました。 このことは、PDML ファイルをアプレット・インストールの一部として組み込む時にのみ必要となります。 パフォーマンス上の理由で、通常はパネル定義を *シリアル化* し、 Graphical Toolbox が PDML を実行時に解釈する必要がないようにします。 こうすることにより、パネルの圧縮されたバイナリー表記を作成するので、 ユーザー・インターフェースのパフォーマンスが大いに向上します。 詳しくは、ツールにより生成されるファイルの説明を参照してください。

アプレットのインストールおよび実行

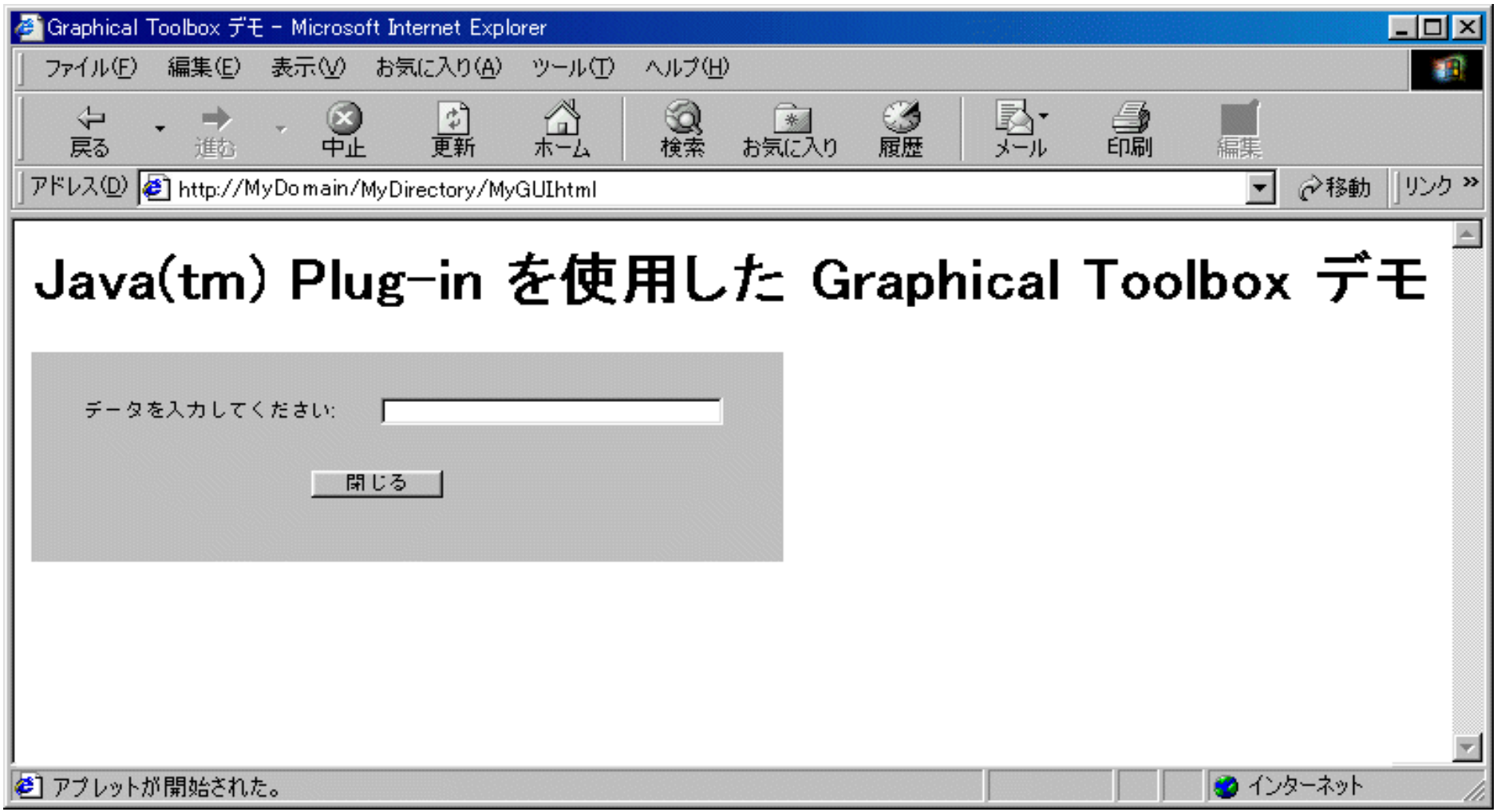
以下のステップを実行して、指定した Web サーバーにアプレットをインストールします。

- SampleApplet.java をコンパイルします。
- アプレットのバイナリーを入れるため、MyGUI.jar という名前の JAR ファイルを作成します。 このバイナリーには、SampleApplet.java および SampleBean.java のコンパイル時に作成されるクラス・ファイル、PDML ファイル (MyGUI.pdml)、 およびリソース・バンドル (MyGUI.properties) が含まれます。
- 新しい JAR ファイルを、Web サーバー上で選択するディレクトリーにコピーします。 オンライン・ヘルプを含む HTML ファイルを、サーバーのディレクトリーにコピーします。
- Graphical Toolbox の JAR ファイルを、サーバーのディレクトリーにコピーします。
- 最後に、組み込みアプレットを含む HTML ファイル (MyGUI.html) を、 サーバーのディレクトリーにコピーします。

ヒント: アプレットをテストするとき、 ワークステーションの CLASSPATH 環境変数から Graphical Toolbox の JAR を確実に削除する必要があります。 この値が削除されていないと、アプレットのリソースをサーバー上に配置できないという趣旨のエラー・メッセージが表示されます。

これでアプレットを実行する準備ができました。 サーバー上にある MyGUI.html を Web ブラウザーに表示します。 まだ Java プラグインがインストールされていない場合、 それをインストールするかどうかを尋ねてきます。 プラグインをインストールしてアプレットを開始すると、ブラウザーの表示は図 1 のようになります。

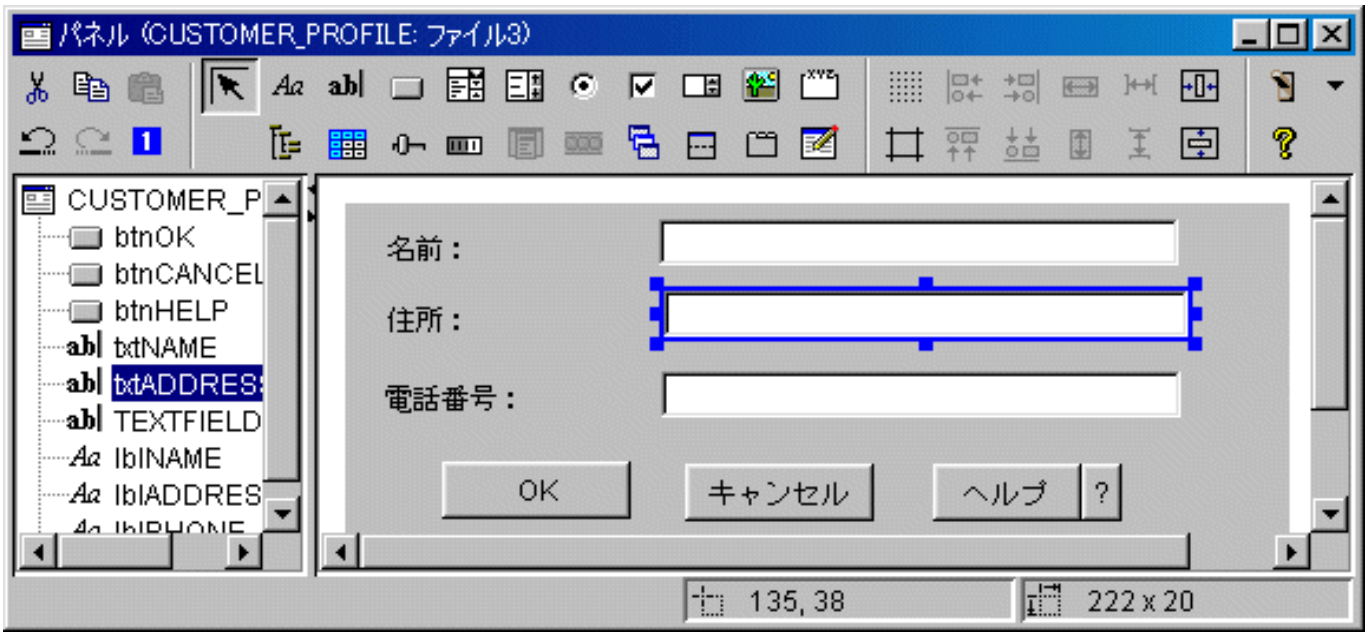
図 1: ブラウザーでのサンプル・アプレットの実行



GUI ビルダーの「パネル・ビルダー」ツールバー

図 1 は、GUI ビルダーの「パネル・ビルダー」ウィンドウを示しています。 図 1 に続いて、パネル・ビルダーの各ツール・アイコンを示し、その機能を説明するリストがあります。

図 1: GUI ビルダーの「パネル」ウィンドウ



「ポインター」をクリックすると、 パネル上のコンポーネントを移動したりサイズを変更したりできます。

「ラベル」をクリックすると、 パネル上に静的ラベルを挿入できます。

「テキスト (Text)」をクリックすると、 パネル上にテキスト・ボックスを挿入できます。

「ボタン (Button)」をクリックすると、 パネル上にボタンを挿入できます。

「コンボ・ボックス (Combo Box)」をクリックすると、 パネル上にドロップダウン・リスト・ボタンを挿入できます。

「リスト・ボックス (List Box)」をクリックすると、 パネル上にリスト・ボタンを挿入できます。

「ラジオ・ボタン (Radio Button)」をクリックすると、 パネル上にラジオ・ボタンを挿入できます。

「チェック・ボックス (Checkbox)」をクリックすると、 パネル上にチェック・ボックスを挿入できます。

「スピナー (Spinner)」をクリックすると、 パネル上にスピナーを挿入できます。

「イメージ」をクリックすると、 パネル上にイメージを挿入できます。

「メニュー・バー」をクリックすると、 パネル上にメニュー・バーを挿入できます。

 「グループ・ボックス (Group Box)」をクリックすると、 パネル上にラベル付きグループ・ボックスを挿入できます。

 「ツリー」をクリックすると、 パネル上に階層ツリーを挿入できます。

 「テーブル」をクリックすると、 パネル上にテーブルを挿入できます。

 「スライダー (Slider)」をクリックすると、 パネル上に調整可能なスライダーを挿入できます。

 「進行状況バー (Progress Bar)」をクリックすると、 パネル上に進行状況バーを挿入できます。

 「デック・ペイン (Deck Pane)」をクリックすると、 パネル上にデック・ペインを挿入できます。デック・ペインには多数のパネルが入っています。 複数のパネルを選択できますが、完全に表示されるのは選択した 1 つのパネルだけです。

 「分割ペイン (Split Pane)」をクリックすると、 パネル上に分割ペインを挿入できます。分割ペインは、2 つの水平または垂直ペインに分割された 1 つのペインです。

 「タブ形式ペイン (Tabbed Pane)」をクリックすると、 パネル上にタブ形式ペインを挿入できます。タブ形式ペインには多数のパネルが入っており、上部にタブが付いています。 ユーザーはタブをクリックしてパネルのコンテンツを表示します。 パネルのタイトルが、タブのテキストとして使用されます。

 「カスタム (Custom)」をクリックすると、 パネル上にカスタム定義のユーザー・インターフェース・コンポーネントを挿入できます。

 「ツールバー (Toolbar)」をクリックすると、 パネル上にツールバーを挿入できます。

 「格子線のトグル (Toggle Grid)」をクリックすると、 パネル上で格子線が使用可能になります。

 「上部位置合わせ (Align Top)」をクリックすると、 特定のコンポーネントや主要なコンポーネントの上端でパネルの複数のコンポーネントを位置合わせします。

 「下部位置合わせ (Align Bottom)」をクリックすると、 特定のコンポーネントや主要なコンポーネントの下端でパネルの複数のコンポーネントを位置合わせします。

 「高さ均等化 (Equalize Height)」をクリックすると、 特定のコンポーネントや主要なコンポーネントの高さに合わせてパネルの複数のコンポーネントの高さを均等化できます。

 「縦合わせ (Center Vertically)」をクリックすると、 選択したコンポーネントをパネルに対して中央で縦合わせします。

 「マージンのトグル (Toggle Margins)」をクリックすると、 パネルのマージンが表示されます。

 「左寄せ (Align Left)」をクリックすると、 特定のコンポーネントや主要なコンポーネントの左端でパネルの複数のコンポーネントを位置合わせします。

 「右寄せ (Align Right)」をクリックすると、 特定のコンポーネントや主要なコンポーネントの右端でパネルの複数のコンポーネントを位置合わせします。

 「幅均等化 (Equalize Width)」をクリックすると、 特定のコンポーネントや主要なコンポーネントの高さに合わせてパネルの複数のコンポーネントの幅を均等化できます。

 「横合わせ (Center Horizontally)」をクリックすると、 選択したコンポーネントをパネルに対して中央で横合わせします。

 「切り取り (Cut)」をクリックすると、 パネル・コンポーネントを切り取ります。

 「コピー・ボタン (Copy button)」をクリックすると、 パネル・コンポーネントをコピーできます。

 「貼り付け (Paste)」をクリックすると、 異なるパネルまたはフィルター間でパネル・コンポーネントを貼り付けることができます。

 「取り消し (Undo)」をクリックすると、 最後のアクションを取り消すことができます。

 「再実行 (Redo)」をクリックすると、 最後のアクションを再実行できます。

 「タブ順序 (Tab Order)」をクリックすると、 ユーザーが Tab キーを押してパネル内を移動する場合に、 それぞれのパネル・コンポーネントの選択順序を制御できます。

 「プレビュー (Preview)」をクリックすると、 パネルの外観のプレビューを表示できます。

 「ヘルプ」をクリックすると、 Graphical Toolbox の詳細を表示できます。

IBM Toolbox for Java beans

JavaBeans(TM)は、Java で作成される再利用可能なソフトウェア・コンポーネントです。 このコンポーネントは、高度に定義された機能単位を提供する 1 つのプログラム・コードです。 この機能単位は、ウィンドウ上のボタンのラベルと同じように小さくすることも、 またはアプリケーション全体と同じように大きくすることもできます。

JavaBeans は、ビジュアルまたは非ビジュアルのどちらのコンポーネントにもすることができます。 非ビジュアル JavaBeans の場合も、アイコンや名前などのビジュアル表現法があるので、 ビジュアル操作は可能です。

IBM Toolbox for Java のすべてのパブリック・クラスも、JavaBeans です。 このクラスは、Javasoft JavaBean 標準に合わせて作成されており、 再利用可能なコンポーネントとして機能します。 IBM Toolbox for Java bean 用のプロパティーとメソッドは、 クラスのプロパティーおよびメソッドと同じです。

JavaBeans は、アプリケーション・プログラムで使ったり、 IBM VisualAge for Java 製品などのビルダー・ツールでビジュアルに処理したりできます。

例

- 例: IBM Toolbox for Java bean の例は、 ご使用のプログラムで JavaBeans を使用する 1 つの方法を示します。
- 例: Visual bean ビルダーのコード例では、 IBM Visual Age for Java などの Visual bean ビルダーを使って JavaBeans からプログラムを作成する 1 つの方法を示します。

プログラム呼び出しマークアップ言語

概説

プログラム呼び出しマークアップ言語 (PCML) とは、 サーバー・プログラムの呼び出しに便利なタグ言語であり、より少ない量の Java コードですみます。 PCML は、サーバー・プログラムの入出力パラメーターを使用するためのタグ構文である拡張可能マークアップ言語 (XML) に基づいています。 PCML を使用すると、Java アプリケーションによって呼び出される サーバー・プログラムを完全に記述するタグを定義することができます。 XML の詳細については、XML 参照のセクションを参照してください。

PCML を使用する大きなメリットの 1 つは、作成するコードが少なくてすむことです。 通常は、サーバーと IBM Toolbox for Java オブジェクトの間でデータを接続、検索、 および変換するための余分なコードが必要です。 しかし、PCML を使用すると、IBM Toolbox for Java クラスでのサーバーに対する呼び出しが自動的に処理されます。 PCML クラス・オブジェクトは、PCML タグから生成されます。 これは、アプリケーションからサーバー・プログラムを呼び出すために書き込む必要があるコードの量を最小限に抑えるのに役立ちます。

プラットフォーム要件

PCML は、サーバー・プログラム・オブジェクトに対する分散プログラム呼び出しを Java プラットフォームから行うよう設計されていますが、 サーバー環境内からサーバー・プログラムを呼び出すために PCML を使うことも可能です。

詳細情報を含むトピック

PCML の使用法については、以下のトピックを参照してください。

- PCML のヘルプを使ったプログラムの呼び出し
- PCML のタグを使ったプログラム呼び出しの構築
- PCML のコード例

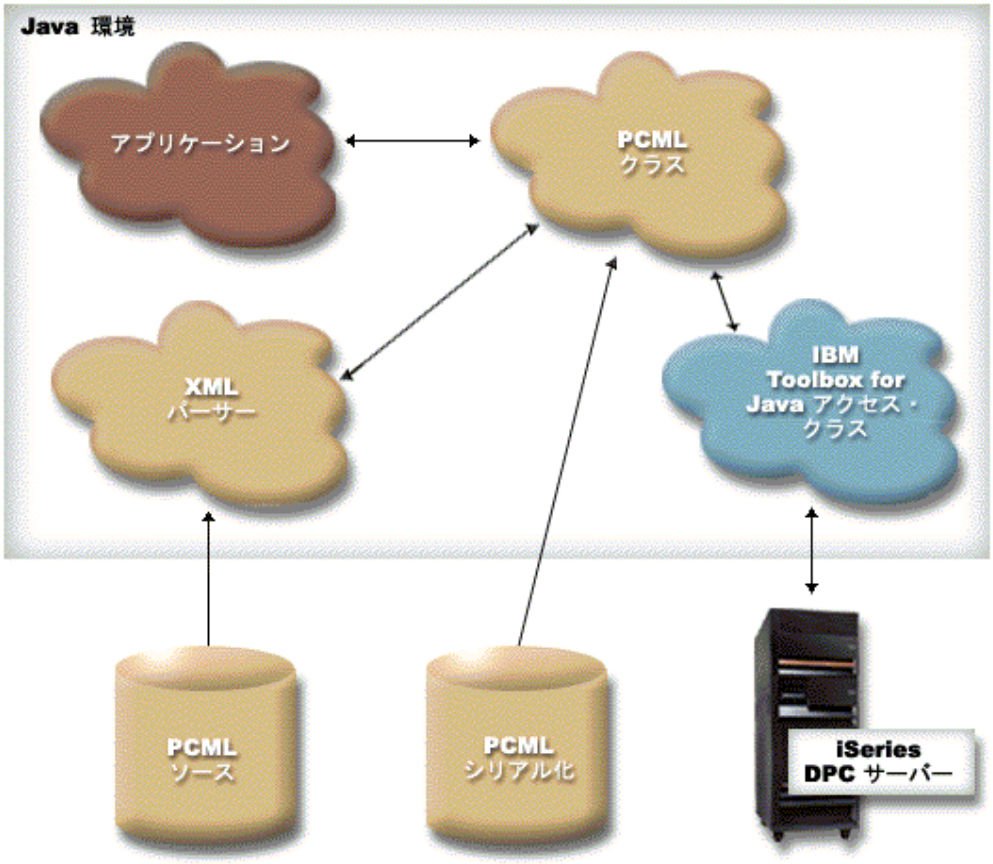
PCML を使用して iSeries プログラム呼び出しを作成する

PCML を使用して iSeries プログラム呼び出しを作成するには、 Java アプリケーションおよび PCML ソース・ファイルを作成して開始する必要があります。

1 つかそれ以上の PCML ソース・ファイルを設計プロセスに基づいて書き、 Java アプリケーションによって呼び出される iSeries プログラムに対するインターフェースを記述する必要があります。 この言語の詳細については、PCML の構文を参照してください。

それから、Java アプリケーションは、PCML クラス (このケースでは、 ProgramCallDocument クラス) と対話します。 ProgramCallDocument クラスは PCML ソース・ファイルを使用して、 Java アプリケーションと iSeries プログラムの間で情報を受け渡します。 図 1 では、Java アプリケーションがどのように PCML クラスと対話するかが図解されています。

図 1: PCML を使用したサーバーに対するプログラム呼び出しの作成



アプリケーションが ProgramCallDocument を作成すると、 IBM XML パーサーは PCML ソース・ファイルの読み取りおよび解析を行います。

ProgramCallDocument クラスが作成された後で、 アプリケーション・プログラムは ProgramCallDocument クラスのメソッドを使用し、 iSeries 分散プログラム呼び出し (DPC) サーバーを介して、 サーバーにある必要な情報を検索します。

実行時のパフォーマンスを向上させるため、 プロダクトの作成中に ProgramCallDocument クラスをシリアル化することができます。 そうすると、ProgramCallDocument はシリアル化済みファイルを使って作成されます。 この場合、実行時に IBM XML パーサーは使用されません。 シリアル化済み PCML ファイルの使用を参照してください。

PCML ソース・ファイルの使用

Java アプリケーションは、PCML ソース・ファイルへの参照を使って ProgramCallDocument オブジェクトを作成することにより、PCML を使用します。 ProgramCallDocument オブジェクトは、 PCML ソース・ファイルを Java リソースと見なします。

»Java アプリケーションは、 Java CLASSPATH あるいは ProgramCallDocument setPath() メソッドのいずれかを使用して、PCML ソース・ファイルを見つけます。 Java アプリケーション・プログラムが、実行時にパスを PCML ファイルに設定する必要がある時には、 setPath() メソッドを使用してください。«

以下の Java コードが、ProgramCallDocument オブジェクトを構成します。

```
AS400 as400 = new AS400();
ProgramCallDocument pcmlDoc = new ProgramCallDocument(as400, "myPcmlDoc");
```

ProgramCallDocument オブジェクトは、myPcmlDoc.pcml というファイル内の PCML ソースは検索しません。 .pcml という拡張子は、コンストラクターで指定されていないことに注意してください。

Java パッケージ にある Java アプリケーションを開発している場合、 PCML リソースの名前をパッケージ修飾することができます。

```
AS400 as400 = new AS400();
ProgramCallDocument pcmlDoc = new ProgramCallDocument(as400, "com.company.package.myPcmlDoc");
```

シリアル化済み PCML ファイルの使用

実行時のパフォーマンスを上げるため、シリアル化済み PCML ファイルを使用することができます。 シリアル化済み PCML ファイルには、PCML の代わりをするシリアル化済み Java オブジェクトが含まれます。 シリアル化済みオブジェクトは、 前述のようにしてソース・ファイルから ProgramCallDocument を構成するときに作成したオブジェクトと同じものです。

シリアル化済み PCML ファイルを使用すると、 IBM XML パーサーが実行時に PCML タグを処理する必要がないため、パフォーマンスが向上します。

PCML は、以下のいずれかの方法でシリアル化することができます。

- コマンド行で次のように入力します。

  ```
  java com.ibm.as400.data.ProgramCallDocument -serialize mypcml
  ```

  この方法は、バッチ処理でアプリケーションを構築する場合に便利です。
- Java プログラムの中に、次の行を追加します。

  ```
  ProgramCallDocument pcmlDoc; // Initialized elsewhere
  pcmlDoc.serialize();
  ```

PCML が myDoc.pcml という名前のソース・ファイルにある場合、 シリアル化すると myDoc.pcml.ser という名前のファイルになります。

PCML ソース・ファイルとシリアル化済み PCML ファイルの比較

以下のコードを考慮に入れて ProgramCallDocument を構成します。

```
AS400 as400 = new AS400();
ProgramCallDocument pcmlDoc = new ProgramCallDocument(as400, "com.mycompany.mypackage.myPcmlDoc");
```

ProgramCallDocument のコンストラクターは、 Java CLASSPATH にある com.mycompany.mypackage パッケージ内の myPcmlDoc.pcml.ser という名前のシリアル化済み PCML ファイルをまず検出しようとします。 シリアル化済み PCML ファイルが存在しない場合、 コンストラクターは次に Java CLASSPATH にある com.mycompany.mypackage パッケージ内の myPcmlDoc.pcml という名前の PCML ソース・ファイルの検出を試みます。 PCML ソース・ファイルが存在しない場合には、例外が発生します。

修飾名

Java アプリケーションは、ProgramCallDocument.setValue() を使用して、 呼び出されている iSeries プログラムの入力値を設定します。 同様にアプリケーションは、ProgramCallDocument.getValue() を使用して、 iSeries プログラムからの出力値を検索します。

ProgramCallDocument クラスからの値にアクセスする場合には、 文書要素または <data> タグの完全修飾名を指定する必要があります。 修飾名は、含まれるタグの名前すべてと、 ピリオドで区切られたそれぞれの名前を連結したものです。

たとえば、以下の PCML ソースが与えられる場合、 "nbrPolygons" 項目の修飾名は "polytest.parm1.nbrPolygons" です。 複数の多角形の中の 1 つにある 1 点を表す "x" 値にアクセスするための修飾名は、 "polytest.parm1.polygon.point.x" です。

修飾名を作成するのに必要な要素のどれか 1 つに名前が付けられていない場合、 その要素のすべての子孫には、修飾名

が付けられません。 修飾名がない要素は、Java プログラムからアクセスすることができません。

```
<pcml version="1.0">
  <program name="polytest" path="/QSYS.lib/MYLIB.lib/POLYTEST.pgm">
    <!-- Parameter 1 contains a count of polygons along with an array of polygons -->
    <struct name="parm1" usage="inputoutput">
      <data name="nbrPolygons" type="int" length="4" init="5" />
      <!-- Each polygon contains a count of the number of points along with an array of points -->
      <struct name="polygon" count="nbrPolygons">
        <data name="nbrPoints" type="int" length="4" init="3" />
        <struct name="point" count="nbrPoints" >
          <data name="x" type="int" length="4" init="100" />
          <data name="y" type="int" length="4" init="200" />
        </struct>
      </struct>
    </struct>
  </program>
</pcml>
```

配列内のデータへのアクセス

すべての <data> または <struct> 要素は、 count 属性を使用することによって配列として定義することができます。 また、<data> または <struct> 要素は、 配列として定義されている別の <struct> 要素の中に含めることができます。

さらに、<data> または <struct> 要素は、 含まれている複数の要素に count 属性が指定されていれば、 多次元配列の中に置くことができます。

アプリケーションが、配列としてまたは配列内に定義されている値を設定または取得するには、 配列の各次元ごとに配列指標を指定する必要があります。 配列指標は、int 値の配列として渡されます。 前述の多角形の配列が与えられる場合、その多角形に関する情報を検索するために、 以下の Java コードを使うことができます。

```
    ProgramCallDocument polytest; // Initialized elsewhere
    Integer nbrPolygons, nbrPoints, pointX, pointY;
    nbrPolygons = (Integer) polytest.getValue("polytest.parm1.nbrPolygons");
    System.out.println("Number of polygons:" + nbrPolygons);
    indices = new int[2];
    for (int polygon = 0; polygon < nbrPolygons.intValue(); polygon++)
    {
        indices[0] = polygon;
        nbrPoints = (Integer) polytest.getValue("polytest.parm1.polygon.nbrPoints", indices );
        System.out.println("  Number of points:" + nbrPoints);

        for (int point = 0; point < nbrPoints.intValue(); point++)
        {
            indices[1] = point;
            pointX = (Integer) polytest.getValue("polytest.parm1.polygon.point.x", indices );
            pointY = (Integer) polytest.getValue("polytest.parm1.polygon.point.y", indices );
            System.out.println("    X:" + pointX + " Y:" + pointY);
        }
    }
```

デバッグ

PCML を使用して複雑なデータ構造を伴うプログラムを呼び出す場合、 ProgramCallDocument クラスからの例外になるような PCML エラーが生じやすくなります。 データのオフセットおよび長さの不正確な記述が関係するエラーの場合、こうした例外のデバッグは困難です。

» Trace クラスから以下のメソッドを使用して、PCML のトレースをオンにします。

```
Trace.setTraceOn(true);       // Turn on tracing function.
Trace.setTracePcmlOn(true);   // Turn on PCML tracing.
```

注: PcmlMessageLog クラス内のすべての public メソッドは、 トレースを含めて、V5R2 では使用すべきでありません。

トレースの setFileName() メソッドは、 以下のタイプの情報を特定のログ・ファイル、あるいはデフォルトでは System.out に送信することを可能にします。 «

- Java アプリケーションと iSeries プログラム間で転送中の 16 進データのダンプ。 これは、文字データが EBCDIC に変換され、整数がビッグ・エンディアンに変換された後、 プログラムの入力パラメーターを表示します。 また、出力パラメーターが Java 環境に変換される前にも、それらを表示します。

  このデータは、左側に 16 進数字、右側に文字の解釈を伴う典型的な 16 進ダンプ形式で表示されます。 以下に示すのは、このダンプ形式の例です。

```
qgyolobj[6]
Offset : 0....... 4....... 8....... C....... 0....... 4....... 8....... C.......
0...4...8...C...0...4...8...C...
     0 : 5CE4E2D9 D7D9C640 4040                                             **USRPRF
*
```

  上記の例では、ダンプは、7 番目のパラメーターの 10 バイトが "*USRPRF" に対するデータ・セットであることを示します。

- 出力パラメーターの場合、以下に示す 16 進ダンプは、 データが文書でどのように解釈されているかを説明します。

```
/QSYS.lib/QGY.lib/QGYOLOBJ.pgm[2]
Offset : 0....... 4....... 8....... C....... 0....... 4....... 8....... C.......
0...4...8...C...0...4...8...C...
     0 : 0000000A 0000000A 00000001 00000068 D7F0F9F9 F0F1F1F5 F1F4F2F6 F2F5F400
*.................P09901151426254.*
    20 : 00000410 00000001 00000000 00000000 00000000 00000000 00000000 00000000
*................................*
    40 : 00000000 00000000 00000000 00000000                                 *................
*
Reading data -- Offset: 0   Length: 4   Name: "qgyolobj.listInfo.totalRcds"   Byte data: 0000000A
Reading data -- Offset: 4   Length: 4   Name: "qgyolobj.listInfo.rcdsReturned"    Byte data: 0000000A
Reading data -- Offset: 8   Length: 4   Name: "qgyolobj.listInfo.rqsHandle"   Byte data: 00000001
Reading data -- Offset: c   Length: 4   Name: "qgyolobj.listInfo.rcdLength"   Byte data: 00000068
Reading data -- Offset: 10  Length: 1   Name: "qgyolobj.listInfo.infoComplete"    Byte data: D7
Reading data -- Offset: 11  Length: 7   Name: "qgyolobj.listInfo.dateCreated" Byte data:
F0F9F9F0F1F1F5
Reading data -- Offset: 18  Length: 6   Name: "qgyolobj.listInfo.timeCreated" Byte data: F1F4F2F6F2F5
Reading data -- Offset: 1e  Length: 1   Name: "qgyolobj.listInfo.listStatus"  Byte data: F4
Reading data -- Offset: 1f  Length: 1   Name: "qgyolobj.listInfo.[8]" Byte data: 00
Reading data -- Offset: 20  Length: 4   Name: "qgyolobj.listInfo.length0fInfo"    Byte data: 00000410
Reading data -- Offset: 24  Length: 4   Name: "qgyolobj.listInfo.firstRecord" Byte data: 00000001
Reading data -- Offset: 28  Length: 40  Name: "qgyolobj.listInfo.[11]"    Byte data:
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

  上記のメッセージは、 iSeries プログラムから来る出力データが PCML ソースと一致しないケースを診断する場合に大変有用です。 このことは、動的長さおよびオフセットを使用していると容易に生じ得ます。

PCML の構文

PCML は、それぞれに独自の属性タグがある、以下のタグで構成されます。

- program タグは、１つのプログラムを記述するコードを開始および終了します。
- struct タグは、プログラムに対する引き数として、 または別の名前付き構造体の内側にあるフィールドとして指定できる名前付き構造体を定義します。 struct タグには、構造体にある各フィールドの data タグまたは struct タグが含まれます。
- data タグは、 プログラムまたは構造体の内側にあるフィールドを定義します。

以下の例では、１つのデータ・カテゴリーといくつかの分離データを伴う１つのプログラムを PCML 構文で記述しています。

```
<program>

    <struct>
        <data> </data>
    </struct>

    <data> </data>

</program>
```

PCML program タグ

PCML program タグは、以下の要素を使って拡張することができます。

```
<program name="name"
    [ entrypoint="entry-point-name" ]
» [ epccsid="ccsid" ]«
    [ path="path-name" ]
    [ parseorder="name-list" ]
    [ returnvalue="{ void | integer }" ]
    [ threadsafe="{ true | false }" ]>
</program>
```

以下の表では、program タグ属性をリストします。 各エントリーには、属性名、可能な有効値、および属性の説明が含まれています。

属性	値	説明
entrypoint=	*entry-point-name*	プログラム呼び出しのターゲットになっている、 サービス・プログラム・オブジェクト内のエントリー・ポイントの名前を指定します。
» epccsid=	*ccsid*	サービス・プログラム内の入り口点の CCSID を指定します。 詳しくは、 ServiceProgramCall javadoc にある、 サービス・プログラム・エントリーの注記を参照してください。 «
name=	*name*	プログラムの名前を指定します。
path=	*path-name*	プログラム・オブジェクトへのパスを指定します。 デフォルト値では、このプログラムは QSYS ライブラリーにあると見なされます。 このパスは、 *PGM または *SRVPGM オブジェクトへの有効な IFS パス名である必要があります。 *SRVPGM オブジェクトを呼び出す場合、 呼び出されるエントリー・ポイントの名前を示すエントリー・ポイントの属性を指定する必要があります。 エントリー・ポイントの属性が指定され

		ない場合、 この属性のデフォルト値は QSYS ライブラリーの *PGM オブジェクトであると見なされます。 エントリー・ポイントの属性が指定されると、 この属性のデフォルト値は QSYS ライブラリーの *SRVPGM オブジェクトであると見なされます。 パス名はすべて大文字で指定する必要があります。 » アプリケーションが実行時にパスを設定しなければならない時、 たとえば、 ユーザーがインストールのためにどのライブラリーを使用するかを指定する時には、 path 属性を使用しないでください。 この場合には、 ProgramCallDocument.setPath() メソッドを使用してください。 «
parseorder=	*name-list*	出力パラメーターが処理される順番を指定します。 指定する値は、パラメーターが処理される順番にパラメーター名をブランクで区切ったリストです。 リストにある名前は、<program> に所属するタグの name 属性で指定した名前と同一でなければなりません。 デフォルト値では、文書内でタグが現れる順番に出力パラメーターが処理されます。 プログラムには、それ以前のパラメーターの情報を記述する 1 つのパラメーターの中に情報を戻すものもあります。 たとえば、プログラムが最初のパラメーターに構造の配列を戻し、 2 番目のパラメーターに配列内の要素の数を戻すとします。 この場合、最初のパラメーター内の処理する構造の数を決定するためには、 2 番目のパラメーターを ProgramCallDocument に応じた順番で処理する必要があります。

returnvalue=	*void* プログラムは値を戻しません。 *integer* プログラムは 4 バイトの符号付き整数を戻します。	サービス・プログラム呼び出しから値が戻される場合、 値のタイプを指定します。 この属性は、*PGM オブジェクト呼び出しでは許可されていません。
threadsafe=	*true* プログラムはスレッド・セーフであると見なされます。 *false* プログラムはスレッド・セーフであると見なされません。	同一サーバー上にある Java プログラムと iSeries プログラムを呼び出す場合、このプロパティーを使用して、 Java プログラムと同じジョブおよび同じスレッド上にある iSeries プログラムを呼び出すかどうかを指定します。 使用中のプログラムがスレッド・セーフであることがわかっている場合、 プロパティーを *true* に設定したほうがパフォーマンスが向上します。 環境を保護しておくために、デフォルトではプログラムを別々のサーバー・ジョブで呼び出します。 デフォルト値は *false* です。

PCML struct タグ

PCML struct タグは、以下の要素を使って拡張することができます。

```
<struct name="name"
    [ count="{number | data-name }"]
    [ maxvrm="version-string" ]
    [ minvrm="version-string" ]
    [ offset="{number | data-name }" ]
    [ offsetfrom="{number | data-name | struct-name }" ]
    [ outputsize="{number | data-name }" ]
    [ usage="{ inherit | input | output | inputoutput }" ]>
</struct>
```

以下の表では、struct タグ属性をリストします。 各エントリーには、属性名、可能な有効値、および属性の説明が含まれています。

属性	値	説明
name=	*name*	<struct> 要素の名前を指定します。
count=	*number* *number* は、固定されていて変更できないサイズ配列を定義します。 *data-name* *data-name* は、 PCML 文書内の <data> 要素 (実行時の配列内の要素数が入る) の名前を定義します。 *data-name* には、現行要素の完全修飾名または相対名を指定できます。どちらの場合でも、 その名前は type="*int*" で定義された <data> の要素を参照する必要があります。 相対名の解決方法に関する詳細は、相対名の解決を参照してください。	要素が配列であると指定し、配列内にある項目数を識別します。 この属性を省略すると、配列として定義されている別の要素に含まれていても、 その要素は配列として定義されません。

maxvrm=	*version-string*	その要素が存在する最高水準の OS/400 のバージョンを指定します。 OS/400 のバージョンが、この属性で指定されたバージョンより高水準の場合、 この要素とこの要素の子要素 (存在していれば) は、プログラムの呼び出し中には処理されません。 maxvrm 属性は、OS/400 のリリース間で異なるプログラム・インターフェースを定義するのに役立ちます。 バージョン・ストリングの構文は、"VvRrMm" にする必要があります。 大文字の "V"、"R"、および "M" はリテラル文字で、 "v"、"r"、および "m" は、それぞれバージョン、リリース、 およびモディフィケーション・レベルを表す 1 つかそれ以上の数字です。 "v" の値は、1 ～ 255 までででなければなりません。 "r" および "m" の値は、0 ～ 255 までででなければなりません。
minvrm=	*version-string*	この要素が存在する最低水準の OS/400 のバージョンを指定します。 OS/400 のバージョンが、この属性で指定されたバージョンより低水準の場合、 この要素とこの要素の子要素 (存在していれば) は、プログラムの呼び出し中には処理されません。 この属性は、OS/400 のリリース間で異なるプログラム・インターフェースを定義するのに役立ちます。 バージョン・ストリングの構文は、"VvRrMm" にする必要があります。 大文字の "V"、"R"、および "M" はリテラル文字で、 "v"、"r"、および "m" は、それぞれバージョン、リリース、 およびモディフィケーション・レベルを表す 1 つかそれ以上の数字です。 "v" の値は、1 ～ 255 までででなければなりません。 "r" および "m" の値は、0 ～ 255 までででなければなりません。

offset=	*number* *number* は、固定されていて変更できないオフセットを定義します。 *data-name* *data-name* は、PCML 文書内の <data> 要素 (要素に対する実行時のオフセットが入る) の名前を定義します。 data-name には、現行要素の完全修飾名または相対名を指定できます。どちらの場合でも、 その名前は type="*int*" で定義された <data> の要素を参照する必要があります。 相対名の解決方法に関する詳細は、相対名の解決を参照してください。	出力パラメーターにある <struct> 要素に対するオフセットを指定します。 プログラムの中には、固定された構造体と、 その後に 1 つかそれ以上の可変長フィールドまたは構造体が付いた情報を戻すものもあります。 この場合、可変長要素のロケーションは、通常パラメーター内のオフセットまたは変位として指定されます。 offset 属性は、 この <struct> 要素に対するオフセットを記述するのに使われます。 offset 属性は、offsetfrom 属性と一緒に使用します。 offsetfrom 属性を指定しないと、 offset 属性に指定したオフセットの基本ロケーションは、この要素の親になります。offset および offsetfrom 属性の使用法に関する詳しい情報は、 オフセットの指定を参照してください。 offset および offsetfrom 属性は、 プログラムからの出力データを処理するためにのみ使用します。 こうした属性が、入力データのオフセットまたは変位を制御することはありません。 この属性を省略すると、この要素のデータのロケーション (存在していれば) は、 パラメーター内の先行する要素のすぐ後にきます。

offsetfrom=	*number* *number* は、固定されていて変更できない基本ロケーションを定義します。 *number* は通常、このオフセットがパラメーターの先頭からの絶対オフセットであることを示す、 number="0" を指定するために使用します。 *data-name* *data-name* は、 オフセットの基本ロケーションとして使われる <data> 要素の名前を定義します。 指定する要素名は、この要素の親または親元でなければなりません。 offset 属性からの値は、 この属性で指定する要素のロケーションに対する相対位置になります。 *data-name* には、現行要素の完全修飾名または相対名を指定できます。 どちらの場合でも、その名前はこの要素の親元を参照する必要があります。 相対名の解決方法に関する詳細は、相対名の解決を参照してください。 *struct-name* *struct-name* は、 オフセットの基本ロケーションとして使われる <struct> 要素の名前を定義します。 指定する要素名は、この要素の親または親元でなければなりません。 offset 属性からの値は、 この属性で指定する要素のロケーションに対する相対位置になります。 *struct-name* には、現行要素の完全修飾名または相対名を指定できます。 どちらの場合でも、その名前はこの要素の親元を参照する必要があります。 相対名の解決方法に関する詳細は、相対名の解決を参照してください。	offset 属性の相対位置からの基本ロケーションを指定します。 offsetfrom 属性を指定しないと、 offset 属性に指定したオフセットの基本ロケーションは、 この要素の親になります。offset および offsetfrom 属性の使用法に関する詳しい情報は、オフセットの指定を参照してください。 offset および offsetfrom 属性は、 プログラムからの出力データを処理するためにのみ使用します。 こうした属性が、入力データのオフセットまたは変位を制御することはありません。

outputsize=	*number* *number* は、予約するために固定されていて変更できないバイト数を定義します。 *data-name* *data-name* は、 PCML 文書内の <data> 要素 (実行時に出力データ用に予約するバイト数) の名前を定義します。 *data-name* には、現行要素の完全修飾名または相対名を指定できます。どちらの場合でも、 その名前は type=*"int"* で定義された <data> の要素を参照する必要があります。 相対名の解決方法に関する詳細は、相対名の解決を参照してください。	要素の出力データ用に予約するバイト数を指定します。 出力パラメーターが可変長である場合、サーバー・プログラムから戻されるデータ用に予約するバイト数を指定するため、 outputsize 属性が必要です。 outputsize 属性は、すべての可変長フィールドまたは可変サイズ配列で指定することができます。 あるいは、1 つかそれ以上の可変長フィールドを含むパラメーター全体に指定することができます。 outputsize は必要ありません。また、サイズを固定したパラメーターには指定しないでください。 この属性に指定した値は、要素 (子要素をすべて含む) の合計サイズとして使われます。 それで、outputsize 属性は、要素の子または子孫では無視されます。 この要素を省略すると、出力データに予約するバイト数は、 <struct> 要素のすべての子要素に予約するバイト数を追加することにより、 実行時に決まります。
usage=	*inherit*	使用法は、親要素から継承されます。 構造体に親がない場合には、使用法は inputoutput であると見なされます。
	input	構造体は、ホスト・プログラムへの入力値です。 文字および数字タイプの場合、ふさわしく変換されます。
	output	構造体は、ホスト・プログラムからの出力値です。 文字および数字タイプの場合、ふさわしく変換されます。
	inputoutput	構造体は入力値と出力値の両方です。

オフセットの指定

プログラムの中には、固定された構造体と、 その後に 1 つかそれ以上の可変長フィールドまたは構造体が付いた情報を戻すものもあります。 この場合、可変長要素のロケーションは、通常パラメーター内のオフセットまたは変位として指定されます。

オフセットとは、パラメーターの先頭から、フィールドまたは構造体の先頭までの距離 (バイト) です。 変位とは、ある構造体の先頭から別の構造体の先頭までの距離 (バイト) です。

オフセットの場合、この距離はパラメーターの先頭から始まるため、 offsetfrom="0" と指定する必要があります。 以下に示すのは、パラメーターの先頭から始まるオフセットの例です。

```
<pcml version="1.0">
  <program name="myprog" path="/QSYS.lib/MYLIB.lib/MYPROG.pgm">
    <!-- receiver variable contains a path -->
    <struct name="receiver" usage="output" outputsize="2048">
      <data name="pathType"         type="int"  length="4" />
      <data name="offsetToPathName" type="int"  length="4" />
      <data name="lengthOfPathName" type="int"  length="4" />
      <data name="pathName"         type="char" length="lengthOfPathName"
              offset="offsetToPathName"  offsetfrom="0"/>
    </struct>
  </program>
</pcml>
```

変位の場合、この距離は別の構造体の先頭から始まるため、 オフセットに対して相対位置にある構造体の名前を指定します。 以下に示すのは、名前付き構造体の先頭から始まる変位の例です。

```
<pcml ="1.0">
  <program name="myprog" path="/QSYS.lib/MYLIB.lib/MYPROG.pgm">
    <!-- receiver variable contains an object -->
    <struct name="receiver" usage="output" >
      <data name="objectName"       type="char"  length="10" />
      <data name="libraryName"      type="char"  length="10" />
      <data name="objectType"       type="char"  length="10" />
      <struct name="pathInfo" usage="output" outputsize="2048" >
        <data name="pathType"         type="int"  length="4" />
        <data name="offsetToPathName" type="int"  length="4" />
        <data name="lengthOfPathName" type="int"  length="4" />
        <data name="pathName"         type="char" length="lengthOfPathName"
                offset="offsetToPathName"  offsetfrom="pathInfo"/>
```

```
      </struct>
    </struct>
  </program>
</pcml>
```

PCML data タグ

PCML data タグには、以下の属性を含めることができます。 ブラケット [] で囲まれた属性は、その属性が任意選択であることを示します。 任意選択属性を指定する場合には、ソースにブラケットを含めないでください。 選択項目のリストに示しているように、属性値の中には、 ブレース {} (垂直バー | で区切られた選択可能な項目付き) で囲まれているものもあります。 こうした属性の 1 つを指定する場合には、ソースにはブレースを含めず、 表示される選択項目の 1 つだけを指定してください。

```
<data type="{ char | int | packed | zoned | float | byte | struct }"
    [ bidistringtype="{ ST4 | ST5 | ST6 | ST7 | ST8 | ST9 | ST10 | ST11 | DEFAULT }"]
    [ ccsid="{ number | data-name }" ]
»   [ chartype="{ onebyte | twobyte }"]«
    [ count="{ number | data-name }" ]
    [ init="string" ]
    [ length="{ number | data-name }" ]
    [ maxvrm="version-string" ]
    [ minvrm="version-string" ]
    [ name="name" ]
    [ offset="{ number | data-name }" ]
    [ offsetfrom="{ number | data-name | struct-name }" ]
    [ outputsize="{ number | data-name | struct-name }" ]
    [ passby= "{ reference | value }" ]
    [ precision="number" ]
    [ struct="struct-name" ]
»   [ trim="{ right | left | both | none }" ]«
    [ usage="{ inherit | input | output | inputoutput }" ]>
</data>
```

以下の表では、data タグ属性をリストします。 各エントリーには、属性名、可能な有効値、および属性の説明が含まれています。

属性	値	説明
type=	*char* *char* は文字値を示しています。*char* データ値は、*java.lang.String* として戻されます。 詳細については、 長さの *char* 値を参照してください。 *int* *int* は整数値です。*int* データ値は、*java.lang.Long* として戻されます。 詳細については、 長さおよび精度の *int* 値を参照してください。 *packed* *packed* は、パック 10 進数値です。*packed* データ値は、*java.math.BigDecimal* として戻されます。 詳細については、 長さおよび精度の *packed* 値を参照してください。	使われているデータのタイプ (文字、整数、パック、ゾーン、バイト、または構造体) を示します。 データ・タイプが異なれば、長さおよび精度の属性値も異なります。 詳細については、長さおよび精度の値を参照してください。

	zoned *zoned* は、ゾーン 10 進数値です。*zoned* データ値は、*java.math.BigDecimal* として戻されます。 詳細については、 長さおよび精度の *zoned* 値を参照してください。 *float* *float* は浮動小数点値です。length 属性は、バイト数 ("4" または "8") を指定します。 4 バイト整数は、*java.lang.Float* として戻されます。 8 バイト整数は、*java.lang.Double* として戻されます。 詳細については、 長さの *float* 値を参照してください。 *byte* *byte* はバイト値です。データの変換が行われることはありません。 *byte* データ値は、*byte* 値の配列 (*byte[]*) として戻されます。 詳細については、 長さの *byte* 値を参照してください。 *struct* *struct* は <struct> 要素の名前を指定します。 *struct* を使用すると、1 度構造体を定義しておけば、 その構造体を文書内で何度でも再使用することができます。 type=*"struct"* とすると、 指定した構造体が文書のこの位置に表示されたかのようになります。 *struct* では、長さの値を定めることはできません。また、精度の値を持ちません。	
bidistringtype=	*DEFAULT* DEFAULT は、非両方向データ (LTR) のデフォルトのストリング・タイプです。 *ST4* *ST4* は、ストリング・タイプ 4 です。 *ST5* *ST5* は、ストリング・タイプ 5 です。 *ST6* *ST6* は、ストリング・タイプ 6 です。 *ST7* *ST7* は、ストリング・タイプ 7 です。 *ST8* *ST8* は、ストリング・タイプ 8 です。 *ST9* *ST9* は、ストリング・タイプ 9 です。	両方向ストリング・タイプを type=*"char"* が付いた <data> 要素に指定します。 この属性を省略すると、この要素のストリング・タイプは、CCSID によって暗黙指定されます (明示的に指定されるか、 ホスト環境のデフォルトの CCSID を取ります)。 ストリング・タイプは、BidiStringType クラスの javadoc で定義されます。

	ST10 *ST10* は、ストリング・タイプ 10 です。 *ST11* *ST11* は、ストリング・タイプ 11 です。	
ccsid=	*number* *number* は、固定されていて変更できない CCSID を定義します。 *data-name* *data-name* は、文字データの CCSID (実行時) を含む名前を定義します。 *data-name* には、現行要素の完全修飾名または相対名を指定できます。どちらの場合でも、 その名前は type=*"int"* で定義された <data> の要素を参照する必要があります。 相対名の解決方法に関する詳細は、相対名の解決を参照してください。	<data> 要素の文字データに応じたホストのコード化文字セット識別子 (CCSID) を指定します。 ccsid 属性は、 type=*"char"* が付いた <data> 要素にのみ指定することができます。 この属性を省略すると、この要素の文字データは、ホスト環境のデフォルトの CCSID であると見なされます。
»chartype=	*onebyte* ここで *onebyte* は、各文字のサイズを指定します。 *twobyte* ここで *twobyte* は、各文字のサイズを指定します。 *chartype* を使用している時は、length=*"number"* 属性はバイト数ではなく、文字数を指定します。	各文字のサイズを指定します。«
count=	*number* *number* は、固定されていて変更できない、サイズ配列内の要素数を定義します。 *data-name* *data-name* は、 PCML 文書内の <data> 要素 (実行時の配列内の要素数が入る) の名前を定義します。 *data-name* には、現行要素の完全修飾名または相対名を指定できます。どちらの場合でも、 その名前は type=*"int"* で定義された <data> の要素を参照する必要があります。 相対名の解決方法に関する詳細は、相対名の解決を参照してください。	要素が配列であると指定し、配列内にある項目数を識別します。 *count* 属性を省略すると、配列として定義されている別の要素に含まれていても、 その要素は配列として定義されません。

init=	*string*	<data> 要素の初期値を指定します。 <data> 要素が usage=*"input"* または usage=*"inputoutput"* 付きで使用され、 初期値がアプリケーション・プログラムによって明示的に設定されていない場合、 *init* 値が使われます。 指定された初期値は、スカラー値の初期設定に使われます。 要素が配列として定義されるか、または配列として定義された構造体に含まれる場合、 指定された初期値は配列内のすべての項目の初期値として使用されます。
length=	»*number* ここで、*number* はデータが必要とするバイト数を定義します。 しかし、*chartype* 属性を使用する時は、*number* はバイト数ではなく、文字数を指定します。 « *data-name* *data-name* は、PCML 文書内の <data> (実行時の長さが入る) の名前を定義します。 *data-name* は、type=*"char"* または type=*"byte"* が付いた <data> 要素にのみ指定することができます。 *data-name* には、現行要素の完全修飾名または相対名を指定できます。 どちらの場合でも、 その名前は type=*"int"* で定義された <data> の要素を参照する必要があります。 相対名の解決方法に関する詳細は、相対名の解決を参照してください。	データ要素の長さを指定します。 この属性値の使用法は、データ・タイプによって変わります。 詳細については、長さおよび精度の値を参照してください。
maxvrm=	*version-string*	この要素が存在する最高水準の iSeries のバージョンを指定します。 iSeries のバージョンが、 この属性で指定されたバージョンより高水準の場合、 この要素とこの要素の子要素 (存在していれば) は、プログラムの呼び出し中には処理されません。 この属性は、iSeries のリリース間で異なるプログラム・インターフェースを定義するのに役立ちます。 バージョン・ストリングの構文は、"VvRrMm" にする必要があります。 大文字の "V"、"R"、および "M" はリテラル文字で、 "v"、"r"、および "m" は、それぞれバージョン、リリース、 および

		モディフィケーション・レベルを表す 1 つかそれ以上の数字です。"v" の値は、1 ～ 255 までででなければなりません。 "r" および "m" の値は、0 ～ 255 までででなければなりません。
minvrm=	*version-string*	この要素が存在する最低水準の iSeries のバージョンを指定します。 iSeries のバージョンが、 この属性で指定されたバージョンより低水準の場合、 この要素とこの要素の子要素 (存在していれば) は、プログラムの呼び出し中には処理されません。 この属性は、iSeries のリリース間で異なるプログラム・インターフェースを定義するのに役立ちます。 バージョン・ストリングの構文は、"VvRrMm" にする必要があります。 大文字の "V"、"R"、および "M" はリテラル文字で、 "v"、"r"、および "m" は、それぞれバージョン、リリース、 およびモディフィケーション・レベルを表す 1 つかそれ以上の数字です。"v" の値は、1 ～ 255 までででなければなりません。 "r" および "m" の値は、0 ～ 255 までででなければなりません。
name=	*name*	<data> 要素の名前を指定します。
offset=	*number* *number* は、固定されていて変更できないオフセットを定義します。 *data-name* *data-name* は、PCML 文書内の <data> 要素 (この要素に対する実行時のオフセットが入る) の名前を定義します。 *data-name* には、現行要素の完全修飾名または相対名を指定できます。どちらの場合でも、 その名前は type=*"int"* で定義された <data> の要素を参照する必要があります。 相対名の解決方法に関する詳細は、相対名の解決を参照してください。	出力パラメーターにある <data> 要素に対するオフセットを指定します。 プログラムの中には、固定された構造体と、 その後に 1 つかそれ以上の可変長フィールドまたは構造体が付いた情報を戻すものもあります。 この場合、可変長要素のロケーションは、通常パラメーター内のオフセットまたは変位として指定されます。 offset 属性は、offsetfrom 属性と一緒に使用します。 offsetfrom 属性を指定しないと、 offset 属性に指定したオフセットの基本ロケーションは、 この要素の親になります。offset および offsetfrom 属

		性の使用法に関する詳しい情報は、 オフセットの指定を参照してください。 offset および offsetfrom 属性は、プログラムからの出力データを処理するためにのみ使用します。こうした属性が、入力データのオフセットまたは変位を制御することはありません。 この属性を省略すると、この要素のデータのロケーション (存在していれば) は、 パラメーター内の先行する要素のすぐ後にきます。
offsetfrom=	*number* *number* は、固定されていて変更できない基本ロケーションを定義します。通常 *number* は、 このオフセットがパラメーターの先頭からの絶対オフセットであることを示す、number=*"0"* を指定するために使用します。 *data-name* *data-name* は、 オフセットの基本ロケーションとして使われる <data> 要素の名前を定義します。 指定する要素名は、この要素の親または親元でなければなりません。 offset 属性からの値は、 この属性で指定する要素のロケーションに対する相対位置になります。 *data-name* には、現行要素の完全修飾名または相対名を指定できます。 どちらの場合でも、その名前はこの要素の親元を参照する必要があります。 相対名の解決方法に関する詳細は、相対名の解決を参照してください。 *struct-name* *struct-name* は、 オフセットの基本ロケーションとして使われる <struct> 要素の名前を定義します。 指定する要素名は、この要素の親または親元でなければなりません。 offset 属性からの値は、 この属性で指定する要素のロケーションに対する相対位置になります。 *struct-name* には、現行要素の完全修飾名または相対名を指定できます。 どちらの場合でも、その名前はこの要素の親元を参照する必要があります。 相対名の解決方法に関する詳細は、相対名の解決を参照してください。	offset 属性の相対位置からの基本ロケーションを指定します。 offsetfrom 属性を指定しないと、offset 属性に指定したオフセットの基本ロケーションは、 この要素の親になります。offset および offsetfrom 属性の使用法に関する詳しい情報は、 オフセットの指定を参照してください。 offset および offsetfrom 属性は、プログラムからの出力データを処理するためにのみ使用します。こうした属性が、入力データのオフセットまたは変位を制御することはありません。

outputsize=	*number* *number* は、固定されていて変更できないバイト数を定義して予約します。 *data-name* *data-name* は、 PCML 文書内の <data> 要素 (実行時に出力データ用に予約するバイト数) の名前を定義します。 *data-name* には、現行要素の完全修飾名または相対名を指定できます。どちらの場合でも、 その名前は type=*"int"* で定義された <data> の要素を参照する必要があります。 相対名の解決方法に関する詳細は、相対名の解決を参照してください。	要素の出力データ用に予約するバイト数を指定します。 出力パラメーターが可変長である場合、 iSeries プログラムから戻されるデータ用に予約するバイト数を指定するため、 outputsize 属性が必要です。 outputsize 属性は、すべての可変長フィールドまたは可変サイズ配列で指定することができます。 あるいは、1 つかそれ以上の可変長フィールドを含むパラメーター全体に指定することができます。 outputsize は必要ありません。また、 サイズを固定したパラメーターには指定しないでください。 この属性に指定した値は、要素 (子要素をすべて含む) の合計サイズとして使われます。 それで、 outputsize 属性は、要素の子または子孫では無視されます。 outputsize を省略すると、出力データに予約するバイト数は、 <struct> 要素のすべての子要素に予約するバイト数を追加することにより、 実行時に決まります。
passby=	*reference* ここで、*reference* は、 パラメーターが参照によって受け渡されることを示します。プログラムが呼び出されると、 このプログラムにはパラメーター値へのポインターが渡されます。 *value* ここで、*value* は整数値を表します。この値は、 type= *"int"* および length=*"4"* が指定されている場合にのみ使用可能です。	パラメーターが参照によって受け渡されるか、 値によって受け渡されるかを指定します。 この要素がサービス・プログラム呼び出しを定義する <program> 要素の子である場合にのみ、 この属性は使用可能になります。
precision=	*number*	ある種の数値データ・タイプの精度のバイト数を指定します。 詳細については、 長さおよび精度の値を参照してください。
struct=	*name*	<data> 要素の <struct> 要素の名前を指定します。 struct 属性は、 type=*"struct"* が付いた <data> 要素のみに指定することができます。

»trim=	*right* ここで、*right* は、末尾空白文字の切り取りを意味する、デフォルトの動作です。 *left* ここで、*left* は、先頭空白文字を切り取ることを意味します。 *both* ここで、*both* は、先頭空白文字と末尾空白文字の両方を切り取ることを意味します。 *none* ここで、*none* は、空白文字を切り取らないことを意味します。	文字データから空白文字を切り取る方法を指定します。«
usage=	*inherit*	使用法は、親要素から継承されます。 構造体に親がない場合には、使用法は *inputoutput* であると見なされます。
	input	ホスト・プログラムへの入力値を定義します。 文字および数字タイプの場合、ふさわしく変換されます。
	output	ホスト・プログラムからの出力値を定義します。 文字および数字タイプの場合、ふさわしく変換されます。
	inputoutput	入力値と出力値の両方を定義します。

オフセットの指定

プログラムの中には、固定された構造体と、 その後に 1 つかそれ以上の可変長フィールドまたは構造体が付いた情報を戻すものもあります。 この場合、可変長要素のロケーションは、通常パラメーター内のオフセットまたは変位として指定されます。

オフセットとは、パラメーターの先頭から、フィールドまたは構造体の先頭までの距離 (バイト) です。 変位とは、ある構造体の先頭から別の構造体の先頭までの距離 (バイト) です。

オフセットの場合、この距離はパラメーターの先頭から始まるため、 offsetfrom="0" と指定する必要があります。 以下に示すのは、パラメーターの先頭から始まるオフセットの例です。

```
<pcml version="1.0">
  <program name="myprog" path="/QSYS.lib/MYLIB.lib/MYPROG.pgm">
    <!-- receiver variable contains a path -->
    <struct name="receiver" usage="output" outputsize="2048">
      <data name="pathType"         type="int"  length="4" />
      <data name="offsetToPathName" type="int"  length="4" />
      <data name="lengthOfPathName" type="int"  length="4" />
      <data name="pathName"         type="char" length="lengthOfPathName"
```

```
                offset="offsetToPathName"  offsetfrom="0"/>
    </struct>
  </program>
</pcml>
```

変位の場合、この距離は別の構造体の先頭から始まるため、 オフセットに対して相対位置にある構造体の名前を指定します。 以下に示すのは、名前付き構造体の先頭から始まる変位の例です。

```
<pcml version="1.0">
  <program name="myprog" path="/QSYS.lib/MYLIB.lib/MYPROG.pgm">
    <!-- receiver variable contains an object -->
    <struct name="receiver" usage="output" >
      <data name="objectName"       type="char"  length="10" />
      <data name="libraryName"      type="char"  length="10" />
      <data name="objectType"       type="char"  length="10" />
      <struct name="pathInfo" usage="output" outputsize="2048" >
        <data name="pathType"         type="int"  length="4" />
        <data name="offsetToPathName" type="int"  length="4" />
        <data name="lengthOfPathName" type="int"  length="4" />
        <data name="pathName"         type="char" length="lengthOfPathName"
                 offset="offsetToPathName"  offsetfrom="pathInfo"/>
      </struct>
    </struct>
  </program>
</pcml>
```

長さおよび精度の値

データ・タイプが異なれば、長さおよび精度の属性値も異なります。 以下の表には、長さおよび精度に指定できる値の説明と共に各データ・タイプがリストされています。

データ・タイプ	長さ	精度
type="char"	この要素のデータのバイト数を指定します。これは必ずしも文字数ではありません。 リテラルの *number* または *data-name* のどちらかを指定する必要があります。	該当しません
type="int"	この要素のデータのバイト数は、2、4、または 8 です。 リテラルの *number* を指定しなければなりません。	精度のビット数、または整数が符号付きまたは符号なしのどちらであるかを指定します。 ● length="2" の場合 ❍ 符号付き 2 バイト整数の場合、precision="15" を使用します。 これが、デフォルト値です。 ❍ 符号なし 2 バイト整数の場合、precision="16" を使用します。 ● length="4" の場合 ❍ 符号付き 4 バイト整数の場合、precision="31" を使用します。 ❍ 符号なし 4 バイト整数の場合、 precision="32" を使用します。 ● length="8" の場合、 符号付き 8 バイト整数には、precision="63" を使用します。
type="packed" または "zoned"	この要素のデータの数値の桁数です。 リテラルの *number* を指定する必要があります。	この要素の 10 進数の数値です。 この数値は、ゼロ以上でなければならず、 length 属性で指定される合計桁数以下でなければなりません。
type="float"	この要素のデータのバイト数 (4 または 8) です。 リテラルの *number* を指定する必要があります。	該当しません
type="byte"	この要素のデータのバイト数です。 リテラルの *number* または *data-name* を指定する必要があります。	該当しません
type="struct"	使用できません。	該当しません

相対名の解決

属性の中には、文書内にある別の要素またはタグの名前を、属性値として指定できるものもあります。 指定する名前は、現行のタグに対する相対名にすることができます。

この名前をタグ (現行のタグを含む) の子または子孫にすることができる場合には、 表示して名前を決定します。 このレベルで名前を決定できない場合には、タグを含むさらに上位のレベルで検索を続けます。 こうし

て決定していくと、その名前は相対名ではなく絶対名と見なされて、 結局は <pcml> タグ » または <rfml> タグのどちらかに含まれるタグと一致するようになります。 «

ここに、PCML の使用の例を示します。

```
<pcml version="1.0">
  <program name="polytest" path="/QSYS.lib/MYLIB.lib/POLYTEST.pgm">
    <!-- Parameter 1 contains a count of polygons along with an array of polygons -->
    <struct name="parm1" usage="inputoutput">
      <data name="nbrPolygons" type="int" length="4" init="5" />
      <!-- Each polygon contains a count of the number of points along with an array of points -->
      <struct name="polygon" count="nbrPolygons">
        <data name="nbrPoints" type="int" length="4" init="3" />
        <struct name="point" count="nbrPoints" >
          <data name="x" type="int" length="4" init="100" />
          <data name="y" type="int" length="4" init="200" />
        </struct>
      </struct>
    </struct>
  </program>
</pcml>
```

» ここに、RFML の使用の例を示します。

```
<rfml version="4.0">
  <struct name="polygon">
    <!-- Each polygon contains a count of the number of points along with an array of points. -->
    <data name="nbrPoints" type="int" length="4" init="3" />
    <data name="point" type="struct" struct="point" count="nbrPoints" />
  </struct>
  <struct name="point" >
    <data name="x" type="int" length="4" init="100" />
    <data name="y" type="int" length="4" init="200" />
  </struct>
  <recordformat name="polytest">
    <!-- This format contains a count of polygons along with an array of polygons -->
    <data name="nbrPolygons" type="int" length="4" init="5" />
    <data name="polygon" type="struct" struct="polygon" count="nbrPolygons" />
  </recordformat>
</rfml>«
```

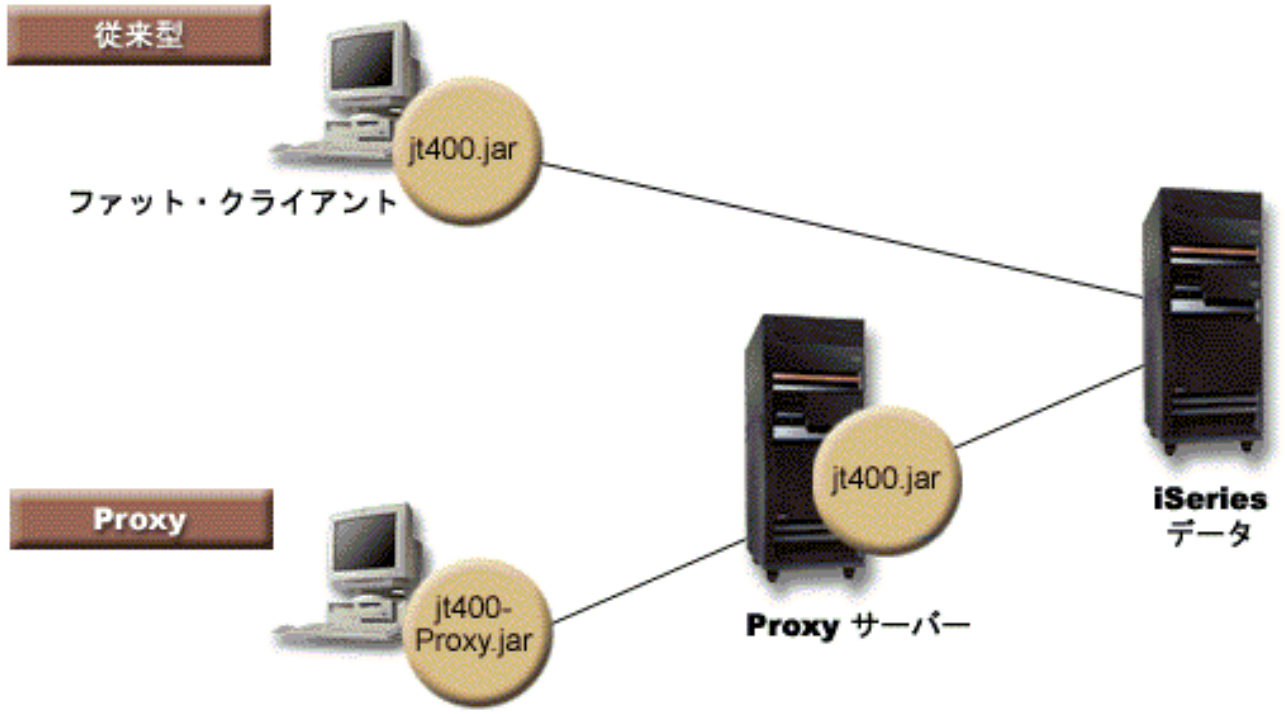
Proxy サポート

IBM Toolbox for Java には、複数のクラス用の Proxy サポートが含まれます。 Proxy サポートは、アプリケーションがある Java 仮想マシン (JVM) 上にある場合に、 IBM Toolbox for Java が別の JVM 上でタスクを実行するときに必要な処理です。 Proxy サポートには、 Secure Sockets Layer (SSL) プロトコルを使用してデータを暗号化することが含まれます。

Proxy クラスは jt400Proxy.jar にあります。 このファイルは、IBM Toolbox for Java の残りの部分に付属しています。 Proxy クラスは、IBM Toolbox for Java の他のクラスと同様、 Java 仮想マシンを持つ任意のコンピューターで実行できる、 プラットフォームから独立した Java クラスのセットで構成されます。 Proxy クラスは、 すべてのメソッド呼び出しをサーバー・アプリケーションまたは Proxy サーバーにディスパッチします。 Proxy サーバー上には、完全な IBM Toolbox for Java クラスがあります。 クライアントが Proxy クラスを使用すると、 実際の IBM Toolbox for Java オブジェクトを作成および管理する Proxy サーバーに要求が転送されます。

図 1 は、標準クライアントと Proxy クライアントがサーバーに接続される方法を示しています。 Proxy サーバーとして、データを含む iSeries を使用できます。

図 1: 標準クライアントと Proxy クライアントがサーバーに接続される方法



Proxy サポートを使用するアプリケーションは、標準の IBM Toolbox for Java クラスを使用する場合より、 パフォーマンスが遅くなります。 これは、小さくなった Proxy クラスをサポートするために余分な通信が必要となるためです。 メソッド呼び出しの数が少ないアプリケーションほど、 性能低下が少なくてすみます。

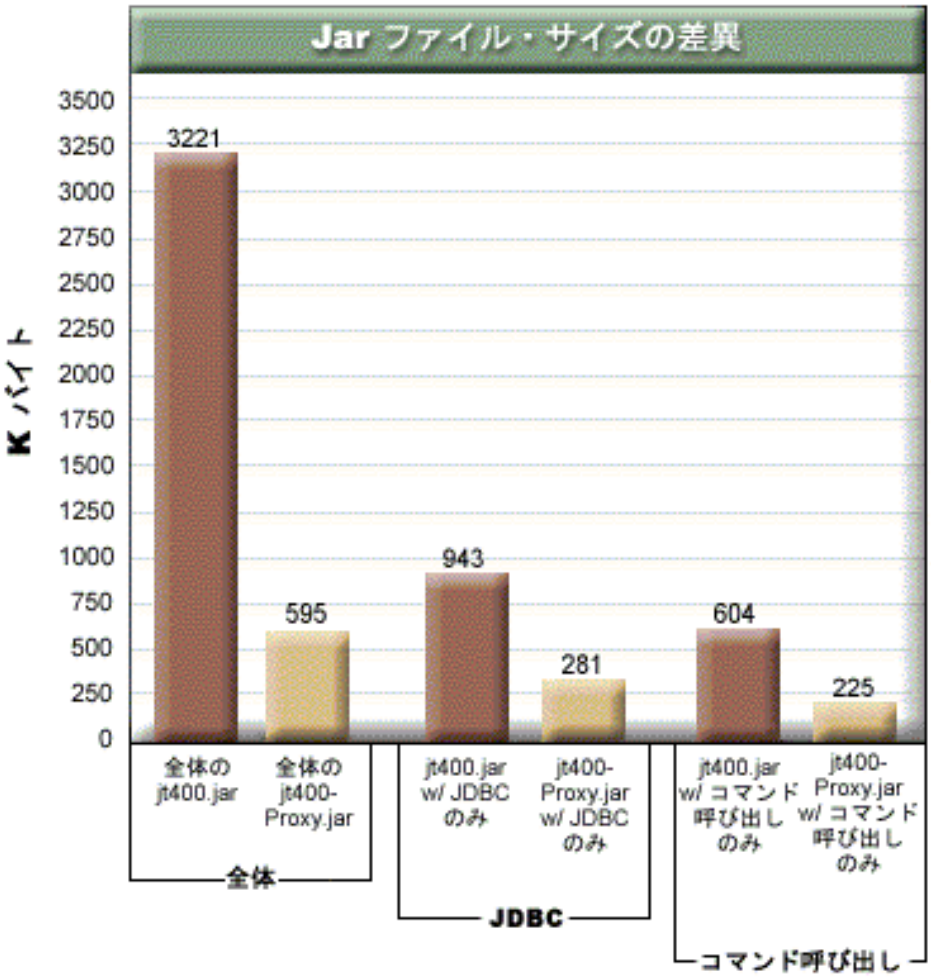
Proxy サポートが存在する前には、パブリック・インターフェースを含むクラス、要求を処理するすべてのクラス、およびアプリケーションそのものが同じ JVM で実行されていました。 Proxy サポートを使用すると、パブリック・インターフェースはアプリケーションと一緒でなければなりませんが、 要求を処理するためのクラスは別の JVM で実行できます。 Proxy サポートは、パブリック・インターフェースを変更しません。 同じプログラムを、IBM Toolbox for Java の Proxy バージョンと、 標準のバージョンで実行できます。

jt400Proxy.jar ファイルを使用する

複数層の Proxy の目標は、パブリック・インターフェースの JAR ファイルをできるだけ小さくして、 そのファイルをアプレットからダウンロードする時間を短くすることです。 Proxy クラスを使用するとき、 IBM Toolbox for Java の全体をクライアントにインストールする必要はありません。 その代わり、jt400Proxy.jar ファイルで AS400JarMaker を使用して、 必要なコンポーネントだけを含めてください。 そうすれば、JAR ファイルをできるだけ小さくすることができます。

図 2 は、Proxy JAR ファイルのサイズと標準の JAR ファイルのサイズとを比較しています。

図 2: Proxy JAR ファイルのサイズと標準の JAR ファイルのサイズの比較



さらに、 他の利点としては、Proxy サポートによりファイアウォールを介してオープンするポートがより少なくてすみます。 標準の IBM Toolbox for Java では、複数のポートをオープンしていなければなりません。 これは、 各 IBM Toolbox for Java サービスが、サーバーとの通信に別々のポートを使用するためです。 たとえば、コマンド呼び出しが JDBC と異なるポートを使用し、JDBC は印刷と異なるポートを使用し、 その他も異なるポートを使用したためです。 それぞれのポートに対して、ファイアウォールの通過を許可する必要があります。 しかし、Proxy サポートを使用すると、すべてのデータは同じポートを通過します。

標準 Proxy および HTTP トンネル

Proxy を介して実行するために、 標準 Proxy と HTTP トンネルという 2 つのオプションを使用できます。

- 標準 Proxy は、Proxy クライアントと Proxy サーバーがポート経由でソケットを使用して通信を行います。 デフォルトのポートは 3470 です。 Proxy サーバーの開始時に、ProxyServer クラスの setPort() メソッド、または -port オプションを使用してデフォルト・ポートを変更します。 たとえば、次のようにします。

```
java com.ibm.as400.access.ProxyServer -port 1234
```

- HTTP トンネルでは、Proxy クライアントと Proxy サーバーが HTTP サーバーを介して通信を行います。 IBM Toolbox for Java は、Proxy 要求を扱うサーブレットを提供します。 Proxy クライアントは、HTTP サーバーを介してサーブレットを呼び出します。 トンネルの利点は、通信が HTTP ポートを介しているので、 ファイアウォールを介して追加のポートをオープンする必要がないということがあります。 トンネルの短所は、標準 Proxy より速度が遅いことです。

IBM Toolbox for Java は Proxy サーバー名を使用して、 標準 Proxy と Tunneling Proxy のどちらが使用されているかを判別します。

- 標準 Proxy の場合、サーバー名だけを使用します。 たとえば、次のようにします。

```
com.ibm.as400.access.AS400.proxyServer=myServer
```

- Tunneling の場合、URL を使用して Proxy クライアントがトンネルを使用するように強制します。 たとえば、次のようにします。

```
com.ibm.as400.access.AS400.proxyServer=http://myServer
```

標準 Proxy の実行時に、接続はクライアントとサーバー間に存在します。 この接続が失敗すると、サーバーはそのクライアントに関連したリソースをクリーンアップします。

HTTP トンネルの使用時に、HTTP プロトコルを使用すると、Proxy の接続がなくなります。 つまり、各データ・フローごとに新しい接続が作成されます。 プロトコルには接続がないため、サーバーはクライアント・アプリケーションが活動状態ではなくなっているかどうかを確認できません。 その結果、サーバーはリソースをいつクリーンアップするかを認識しません。 トンネル・サーバーは、スレッドを使用して、 事前定義した時間間隔 (タイムアウト値に基づく) でリソースをクリーンアップすることにより、 この問題を解決します。

事前定義された時間間隔の終わりに、スレッドが実行し、 最近使用されていないリソースをクリーンアップします。 2 つのシステム・プロパティーは、以下のようにスレッドを制御します。

- com.ibm.as400.access.TunnelProxyServer。 clientCleanupInterval は、 クリーンアップ・スレッドを実行する頻度 (秒単位) を表します。 デフォルトは 2 時間ごとです。
- com.ibm.as400.access.TunnelProxyServer。 clientLifetime は、 リソースがクリーンアップされる前にアイドル状態になることができる時間を秒単位で表します。 デフォルトは 30 分です。

Proxy サーバーを使用する

IBM Toolbox for Java クラスで実装された Proxy サーバーを使用するには、 以下の手順を完了する必要があります。

1. AS400ToolboxJarMaker を jt400Proxy.jar で実行して、 必要のないクラスを廃棄します。 このステップはオプションですが、実行するように推奨されています。
2. jt400Proxy.jar をクライアントに渡す方法を決めます。
   - Java プログラムの場合、 AS400ToolboxInstaller クラスまたは他のメソッドを使用して、 そのプログラムをクライアントに渡します。
   - Java アプレットの場合、 JAR ファイルを HTML サーバーからダウンロードできます。
3. Proxy サーバーとして使用するサーバーを決めます。
   - Java アプリケーションでは、Proxy サーバーとして任意のコンピューターを使用できます。
   - Java アプレットでは、Proxy サーバーは HTTP サーバーと同じコンピューター上で実行する必要があります。
4. サーバー上の CLASSPATH に jt400.jar を指定したことを確認します。
5. Proxy サーバーを開始するか、Proxy サーブレットを使用します。
   - 標準 Proxy の場合、次のコマンドを使用して Proxy サーバーを開始します。

     ```
     java com.ibm.as400.access.ProxyServer
     ```
   - Tunneling Proxy の場合、 Proxy サーブレットを使用するように HTTP サーバーを構成します。 サーブレットのクラス名は com.ibm.as400.access.TunnelProxyServer で、jt400.jar に入っています。
6. クライアント上では、Proxy サーバーを識別するためにシステム・プロパティーを設定します。 IBM Toolbox for Java はこのシステム・プロパティーを使って、 標準 Proxy または Tunneling Proxy が使用されているかどうか判別します。
   - 標準 Proxy の場合、プロパティー値は Proxy サーバーを実行するマシンの名前です。 たとえば、次のようにします。

     ```
     com.ibm.as400.access.AS400.proxyServer=myServer
     ```
   - Tunneling Proxy の場合、URL を使用して Proxy クライアントがトンネルを使用するように強制します。 たとえば、次のようにします。

     ```
     com.ibm.as400.access.AS400.proxyServer=http://myServer
     ```
7. クライアント・プログラムを実行します。

Proxy クラスおよび jt400Proxy.jar 内に存在しないクラスの両方を処理したい場合、 jt400Proxy.jar の代わりに jt400.jar を参照できます。 jt400Proxy.jar は jt400.jar のサブセットなので、 Proxy クラスのすべては jt400.jar ファイルに含まれています。

SSL を使用する

Proxy を使用する際に、Proxy クライアントからターゲットの iSeries サーバーに流れるデータを暗号化するための、 3 つのオプションが使用可能です。 データの暗号化には SSL アルゴリズムが使用されます。

1. Proxy クライアントと Proxy サーバー間のデータ・フローは暗号化可能です。
2. Proxy サーバーとターゲットの iSeries 間のデータ・フローは暗号化可能です。
3. 上記の両方。Proxy クライアントと Proxy サーバー間のデータ・フロー、 Proxy サーバーとターゲットの iSeries 間のフローは暗号化可能です。

詳細については、Secure Sockets Layer を参照してください。

例: Proxy サーバーの使用

Proxy サーバーを上記のリストに従って使用するための、 3 つの例が提供されています。

- Proxy サポートを使用する Java アプリケーションの実行
- Proxy サポートを使用する Java アプレットの実行
- Tunneling Proxy サポートを使用する Java アプリケーションの実行

Proxy サーバーを処理できるクラス

いくつかの IBM Toolbox for Java クラスは、 Proxy サーバー・アプリケーションと共に使用できます。 これらには以下のものが含まれます。

- JDBC
- レコード・レベルでのアクセス
- 統合ファイル・システム
- 印刷
- データ待ち行列
- コマンド呼び出し
- プログラム呼び出し
- サービス・プログラム呼び出し
- ユーザー・スペース
- データ域
- AS400 クラス

- SecureAS400 クラス

その他のクラスは、現時点では jt400Proxy によってサポートされていません。 さらに、統合ファイル・システム許可は Proxy JAR ファイルを使用するだけでは機能しません。 ただし、JarMaker クラスを使用して、 これらのクラスを jt400.jar ファイルから組み込むことができます。

レコード・フォーマット・マークアップ言語

レコード・フォーマット・マークアップ言語 (RFML) は、レコード様式を指定するための XML 拡張機能です。 IBM Toolbox for Java の RFML コンポーネントを使用すると、 Java アプリケーションが RFML 文書を使用して特定の種類のレコード内部のフィールドを指定し、取り扱うことが可能になります。

RFML ソース・ファイルと呼ばれる RFML 文書は、 iSeries のシステム上の物理ファイルおよび論理ファイル用に定義された、 データ記述仕様 (DDS) データ・タイプの実用的なサブセットです。 以下の情報を管理するために RFML を使用することができます。

- ファイル・レコード
- データ待ち行列項目
- ユーザー・スペース
- 任意のデータ・バッファー

注: データ属性を記述するための DDS の使用の詳細については、 DDS 解説書 を参照してください。

RFML は、Toolbox for Java によりサポートされる別の XML 拡張機能である、プログラム呼び出しマークアップ言語 (PCML) に非常によく似ています。 RFML は PCML サブセットでも PCML スーパーセットでもなく、むしろ一種の兄弟言語であり、 いくつかの新しい要素および属性を追加し、別の要素および属性を省略しています。

PCML は、ProgramCall クラスおよび ProgramParameter クラスの使用に代わる、XML の流れをくむ言語です。 同様に、RFML は、Record クラス、RecordFormatクラス、および FieldDescription クラスに代わる、 使いやすく、管理が容易な言語です。

RFML についての詳細は、以下のトピックを参照してください。

要件
RFML を使用するための要件についてお読みください。

RFML の例
アプリケーション内で RFML を使用すると、 書くべきコードの量や、時には複雑さがどのように削減されるかがわかります。 例にはサンプル RFML ソース・ファイルが含まれています。

RecordFormatDocument クラス
データを読み書きするために、RecordFormatDocument クラスを他の Toolbox for Java クラスと共に使用する方法についてお読みくださ

い。

RFML 文書および RFML 構文
RFML ソース・ファイルと呼ばれる RFML 文書および RFML データ・タイプ定義で定義される RFML 構文について学べます。

RFML は、サーバーで XML を使用するための一つの方法にすぎません。iSeries サーバーでの XML の使用についての詳細は、 『Toolbox for Java の XML 拡張機能』および Extensible Markup Language (XML) を参照してください。 «

RFML を使用するための要件

RFML コンポーネントには、 IBM Toolbox for Java のその他のコンポーネントと同じワークステーションの Java 仮想マシン要件があります。

加えて、RFML を実行時に解析するためには、 アプリケーションの CLASSPATH に XML パーサーが組み込まれている必要があります。 XML パーサーは、 org.apache.xerces.parsers.SAXParser クラスを拡張する必要があります。 Toolbox for Java には、互換性のあるパーサーである、XML Parser for Java が組み込まれており、 それは Toolbox for Java x4j400.jar ファイルに組み込まれています。 詳細については、JAR ファイルを参照してください。

注: RFML には、PCML と同様のパーサー要件があります。 PCML に関しては、RFML ファイルをあらかじめシリアル化してある場合、 アプリケーションを実行するために XML パーサーをアプリケーションの CLASSPATH に組み込む必要はありません。

例: Toolbox for Java の Record クラスの使用と比較した RFML の使用

以下の例では、RFML を使用する時と Toolbox for Java の Record クラスを使用する時の相違点を示します。

従来の Record クラスの使用時には、 データ・フォーマット仕様をアプリケーションのビジネス・ロジックに織り込みます。 フィールドの追加、変更、あるいは削除をする時には、Java コードを編集し、再コンパイルしなければなりません。 しかしながら RFML を使用するなら、 ビジネス・ロジックとは完全に別個の RFML ソース・ファイルにデータ・フォーマット仕様が分離されます。 フィールドの変更を適用する時には、RFML ファイルを変更するだけでよく、 多くの場合、Java アプリケーションを変更したり再コンパイルする必要はありません。

この例は、アプリケーションがカスタマー・レコードを扱っており、カスタマー・レコードは、qcustcdt.rfml という名前が付けられた RFML ソース・ファイルで定義済みであるという前提で作成されています。 ソース・ファイルは、各カスタマー・レコードを構成するフィールドを表しています。

以下のリストでは、Toolbox for Java の Record クラス、RecordFormat クラス、 および FieldDescription クラスを使用して Java アプリケーションがカスタマー・レコードを解釈する方法を示しています。

```
// Buffer containing the binary representation of one record of information.
    byte[] bytes;

    // ... Read the record data into the buffer ...

    // Set up a RecordFormat object to represent one customer record.
    RecordFormat recFmt1 = new RecordFormat("cusrec");
    recFmt1.addFieldDescription(new ZonedDecimalFieldDescription(new AS400ZonedDecimal(6, 0),
"cusnum"));
    recFmt1.addFieldDescription(new CharacterFieldDescription(new AS400Text(8, 37), "lstnam"));
    recFmt1.addFieldDescription(new CharacterFieldDescription(new AS400Text(3, 37), "init"));
    recFmt1.addFieldDescription(new CharacterFieldDescription(new AS400Text(13, 37), "street"));
    recFmt1.addFieldDescription(new CharacterFieldDescription(new AS400Text(6, 37), "city"));
    recFmt1.addFieldDescription(new CharacterFieldDescription(new AS400Text(2, 37), "state"));
    recFmt1.addFieldDescription(new ZonedDecimalFieldDescription(new AS400ZonedDecimal(5, 0),
"zipcod"));
    recFmt1.addFieldDescription(new ZonedDecimalFieldDescription(new AS400ZonedDecimal(4, 0),
"cdtlmt"));
    recFmt1.addFieldDescription(new ZonedDecimalFieldDescription(new AS400ZonedDecimal(1, 0),
"chgcod"));
    recFmt1.addFieldDescription(new ZonedDecimalFieldDescription(new AS400ZonedDecimal(6, 2),
"baldue"));
    recFmt1.addFieldDescription(new ZonedDecimalFieldDescription(new AS400ZonedDecimal(6, 2),
"cdtdue"));

    // Read the byte buffer into the RecordFormatDocument object.
    Record rec1 = new Record(recFmt1, bytes);

    // Get the field values.
    System.out.println("cusnum: " + rec1.getField("cusnum"));
    System.out.println("lstnam: " + rec1.getField("lstnam"));
    System.out.println("init:   " + rec1.getField("init"));
    System.out.println("street: " + rec1.getField("street"));
    System.out.println("city:   " + rec1.getField("city"));
    System.out.println("state:  " + rec1.getField("state"));
    System.out.println("zipcod: " + rec1.getField("zipcod"));
    System.out.println("cdtlmt: " + rec1.getField("cdtlmt"));
    System.out.println("chgcod: " + rec1.getField("chgcod"));
```

```
    System.out.println("baldue: " + rec1.getField("baldue"));
    System.out.println("cdtdue: " + rec1.getField("cdtdue"));
```

比較のため、同じレコードを RFML を使用して解釈する別の方法を示します。

RFML を使用してカスタマー・データ・レコードの内容を解釈する Java コードの一例は、次のようになります。

```
    // Buffer containing the binary representation of one record of information.
    byte[] bytes;

    // ... Read the record data into the buffer ...

    // Parse the RFML file into a RecordFormatDocument object.
    // The RFML source file is called qcustcdt.rfml.
    RecordFormatDocument rfml1 = new RecordFormatDocument("qcustcdt");

    // Read the byte buffer into the RecordFormatDocument object.
    rfml1.setValues("cusrec", bytes);

    // Get the field values.
    System.out.println("cusnum: " + rfml1.getValue("cusrec.cusnum"));
    System.out.println("lstnam: " + rfml1.getValue("cusrec.lstnam"));
    System.out.println("init:   " + rfml1.getValue("cusrec.init"));
    System.out.println("street: " + rfml1.getValue("cusrec.street"));
    System.out.println("city:   " + rfml1.getValue("cusrec.city"));
    System.out.println("state:  " + rfml1.getValue("cusrec.state"));
    System.out.println("zipcod: " + rfml1.getValue("cusrec.zipcod"));
    System.out.println("cdtlmt: " + rfml1.getValue("cusrec.cdtlmt"));
    System.out.println("chgcod: " + rfml1.getValue("cusrec.chgcod"));
    System.out.println("baldue: " + rfml1.getValue("cusrec.baldue"));
    System.out.println("cdtdue: " + rfml1.getValue("cusrec.cdtdue"));≪
```

例: RFML ソース・ファイル

この RFML ソース・ファイル例では、 Toolbox for Java の Record クラスの使用と比較した RFML の使用の、 RFML の例で使用されたものと同じ様式のカスタマー・レコードを定義しています。 この RFML ソース・ファイルは、qcustcdt.rfml という名前のテキスト・ファイルになります。

注: 法律上の重要な情報に関しては、 コードの特記事項情報をお読みください。

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE rfml SYSTEM "rfml.dtd">

<rfml version="4.0" ccsid="819">

  <recordformat name="cusrec">

    <data name="cusnum" type="zoned" length="6"  precision="0" init="0"/>
    <data name="lstnam" type="char"  length="8"  ccsid="37" init="A"/>
    <data name="init"   type="char"  length="3"  ccsid="37" init="B"/>
    <data name="street" type="char"  length="13" ccsid="37" init="C"/>
    <data name="city"   type="char"  length="6"  ccsid="37" init="D"/>
    <data name="state"  type="char"  length="2"  ccsid="37" init="E"/>
    <data name="zipcod" type="zoned" length="5" init="1"/>
    <data name="cdtlmt" type="zoned" length="4" init="2"/>
    <data name="chgcod" type="zoned" length="1" init="3"/>
    <data name="baldue" type="zoned" length="6" precision="2" init="4"/>
    <data name="cdtdue" type="zoned" length="6" precision="2" init="5"/>

  </recordformat>

  <recordformat name="cusrec1">

    <data name="cusnum" type="zoned" length="6"  precision="0" init="0"/>
    <data name="lstnam" type="char"  length="8"  ccsid="37" init="A"/>
    <data name="init"   type="char"  length="3"  ccsid="37" init="B"/>
    <data name="street" type="char"  length="13" ccsid="37" init="C"/>
    <data name="city"   type="char"  length="6"  ccsid="37" init="D"/>
    <data name="state"  type="char"  length="2"  ccsid="37" init="E"/>
    <data name="zipcod" type="zoned" length="5" init="1"/>
    <data name="cdtlmt" type="zoned" length="4" init="2"/>
    <data name="chgcod" type="zoned" length="1" init="3"/>
    <data name="baldue" type="struct" struct="balance"/>
```

```
    <data name="cdtdue" type="struct" struct="balance"/>

  </recordformat>

  <recordformat name="cusrecAscii">

    <data name="cusnum" type="zoned" length="6"  precision="0" init="0"/>
    <data name="lstnam" type="char"  length="8" init="A"/>
    <data name="init"   type="char"  length="3" init="B"/>
    <data name="street" type="char"  length="13" init="C"/>
    <data name="city"   type="char"  length="6" init="D"/>
    <data name="state"  type="char"  length="2" init="E"/>
    <data name="zipcod" type="zoned" length="5" init="1"/>
    <data name="cdtlmt" type="zoned" length="4" init="2"/>
    <data name="chgcod" type="zoned" length="1" init="3"/>
    <data name="baldue" type="zoned" length="6" precision="2" init="4"/>
    <data name="cdtdue" type="zoned" length="6" precision="2" init="5"/>

  </recordformat>

  <struct name="balance">
    <data name="amount" type="zoned" length="6" precision="2" init="7"/>
  </struct>

</rfml>
```
«

RecordFormatDocument クラス

RecordFormatDocument クラスを使用することにより、 Java プログラムが、データの RFML 表記と、Record オブジェクトおよび RecordFormat オブジェクトとの間で変換を行い、 他の Toolbox for Java コンポーネントと共に使用することが可能になります。

RecordFormatDocument クラスは、RFML ソース・ファイルに相当するもので、Java プログラムが以下の処理を実行することを可能にするメソッドを備えています。

- Record オブジェクト、RecordFormat オブジェクト、およびバイト配列から RFML ソース・ファイルを構成する
- RecordFormatDocument オブジェクトに含まれている情報を表す Record オブジェクト、 RecordFormat オブジェクト、およびバイト配列を生成する
- 異なるオブジェクト・タイプおよびデータ・タイプの値を取得したり、設定する
- RecordFormatDocument オブジェクトに含まれているデータを表す XML (RFML) を生成する
- RecordFormatDocument オブジェクトが表す RFML ソース・ファイルをシリアル化する

使用可能なメソッドについての詳細は、 RecordFormatDocument の javadoc、 メソッドの概要を参照してください。

RecordFormatDocument クラスを他の Toolbox for Java クラスと共に使用する

RecordFormatDocument クラスを以下の Toolbox for Java クラスと共に使用します。

- Record オブジェクトの読み取り、取り扱い、書き込みを行う、 レコード・レベルのアクセス・ファイル・クラス (AS400File、SequentialFile、および KeyedFile) を含む、 レコード指向のクラス。 このカテゴリーには、さらに LineDataRecordWriter クラスも含まれています。
- バイト配列のデータを一度に読み書きする特定の DataQueue クラス、UserSpace クラス、 および IFSFile クラスを含む、バイト指向のクラス。

RecordFormatDocument クラスを以下の Toolbox for Java クラスと共に使用しないでください。 それらは RecordFormatDocument が扱わない形式でデータを読み書きします。

- DataArea クラス。読み取りメソッドおよび書き込みメソッドは、

String、boolean、および BigDecimal のデータ・タイプのみを扱うため。

- IFSTextFileInputStream および IFSTextFileOutputStream。 これらの読み取りメソッドおよび書き込みメソッドは、String のみを扱うため。
- JDBC クラス。 RFML は、iSeries データ記述仕様 (DDS) で記述されたデータのみに焦点を当てているため。 «

レコード様式文書と RFML 構文

RFML ソース・ファイルと呼ばれる RFML 文書は、 特定のデータ・フォーマットのための仕様を定義するタグを含んでいます。

RFML は PCML を基にしているため、その構文は PCML ユーザーにとってなじみのあるものとなっています。 RFML は XML 拡張機能であるため、RFML ソース・ファイルは読んで理解しやすく、作成も簡単です。 たとえば、RFML ソース・ファイルを、簡単なテキスト・エディターを使用して作成することができます。 さらに、RFML ソース・ファイルは、Java のようなプログラミング言語で行うよりも、 データ構造をより理解しやすい仕方で表現します。

Toolbox for Java の Record クラスの使用と比較した RFML の使用にある RFML の例には、 RFML ソース・ファイルの例が含まれています。

RFML の DTD

RFML 文書タイプ定義 (DTD) では、 有効な RFML 要素および RFML 構文を定義しています。 XML パーサーが RFML ソース・ファイルを実行時に確実に検証できるようにするために、 ソース・ファイル内で RFML DTD を宣言してください。

```
<!DOCTYPE rfml SYSTEM "rfml.dtd">
```

RFML DTD は、jt400.jar ファイル (com/ibm/as400/data/rfml.dtd) にあります。

RFML の構文

RFML DTD は、それぞれに独自の属性タグがあるタグを定義します。 RFML タグを用いて、RFML ソース・ファイル内で以下の要素を宣言し、定義します。

- rfml タグは、 データ・フォーマットを記述する RFML ソース・ファイルを開始、および終了します。
- struct タグは、 RFML ソース・ファイル内部で再使用することのできる名前付き構造体を定義します。 その構造体には、構造体にある各フィールドの data タグが含まれます。
- recordformat タグは、 データ要素あるいは構造体要素への参照を含むレコード様式を定義します。
- data タグは、 レコード様式あるいは構造体内部のフィールドを定義します。

以下の例では、RFML 構文は、1 つのレコード様式および 1 つの構造体を記述

しています。

```
<rfml>

    <recordformat>
        <data> </data>
    </recordformat>

    <struct>
        <data> </data>
    </struct>

</rfml>
```

≪

»

RFML 文書タイプ定義

これは RFML の DTD です。 このバージョンは 4.0 である点に注意してください。 RFML DTD は、jt400.jar ファイル (com/ibm/as400/data/rfml.dtd) にあります。

```
<!--
Record Format Markup Language (RFML) Document Type Definition.

RFML is an XML language.  Typical usage:
   <?xml version="1.0"?>
   <!DOCTYPE rfml SYSTEM "rfml.dtd">
   <rfml version="4.0">
   ...
   </rfml>


(C) Copyright IBM Corporation, 2001,2002
All rights reserved. Licensed Materials Property of IBM
US Government Users Restricted Rights
Use, duplication or disclosure restricted by
GSA ADP Schedule Contract with IBM Corp.
-->

<!-- Convenience entities -->
<!ENTITY % string             "CDATA">    <!-- a string of length 0 or greater -->
<!ENTITY % nonNegativeInteger "CDATA">    <!-- a non-negative integer -->
<!ENTITY % binary2            "CDATA">    <!-- an integer in range 0-65535 -->
<!ENTITY % boolean            "(true|false)">
<!ENTITY % datatype  "(char | int | packed | zoned | float | byte | struct)">
<!ENTITY % biditype  "(ST4 | ST5 | ST6 | ST7 | ST8 | ST9 | ST10 | ST11 | DEFAULT)">


<!-- The document root element -->
<!ELEMENT rfml (struct | recordformat)+>
<!ATTLIST rfml
          version       %string;    #FIXED "4.0"
          ccsid         %binary2;   #IMPLIED
>
<!-- Note: The ccsid is the default value that will be used for any contained <data type="char">
elements that do not specify a ccsid. -->


<!-- Note: RFML does not support nested struct declarations.  All struct elements are direct children
of the root node. -->
<!ELEMENT struct (data)+>
<!ATTLIST struct
          name          ID          #REQUIRED
>


<!-- <!ELEMENT recordformat (data | struct)*> -->
<!ELEMENT recordformat (data)*>
<!ATTLIST recordformat
          name          ID                    #REQUIRED
          description   %string;              #IMPLIED
>
```

```
<!-- Note: On the server, the Record "text description" field is limited to 50 bytes. -->

<!ELEMENT data EMPTY>
<!ATTLIST data
          name          %string;              #REQUIRED

          count         %nonNegativeInteger;  #IMPLIED

          type          %datatype;            #REQUIRED
          length        %nonNegativeInteger;  #IMPLIED
          precision     %nonNegativeInteger;  #IMPLIED
          ccsid         %binary2;             #IMPLIED
          init          CDATA                 #IMPLIED
          struct        IDREF                 #IMPLIED

          bidistringtype %biditype;           #IMPLIED
>
<!-- Note: The 'name' attribute must be unique within a given recordformat. -->
<!-- Note: On the server, the length of Record field names is limited to 10 bytes. -->
<!-- Note: The 'length' attribute is required, except when type="struct". -->
<!-- Note: If type="struct", then the 'struct' attribute is required. -->
<!-- Note: The 'ccsid' and 'bidistringtype' attributes are valid only when type="char". -->
<!-- Note: The 'precision' attribute is valid only for types "int", "packed", and "zoned". -->


<!-- The standard predefined character entities -->
<!ENTITY quot  "&#34;">     <!-- quotation mark -->
<!ENTITY amp   "&#38;#38;"> <!-- ampersand -->
<!ENTITY apos  "&#39;">     <!-- apostrophe -->
<!ENTITY lt    "&#38;#60;"> <!-- less than -->
<!ENTITY gt    "&#62;">     <!-- greater than -->
<!ENTITY nbsp  " ">    <!-- non-breaking space -->
<!ENTITY shy   "&#173;">    <!-- soft hyphen (discretionary hyphen) -->
<!ENTITY mdash "&#38;#x2014;">
<!ENTITY ldquo "&#38;#x201C;">
<!ENTITY rdquo "&#38;#x201D;">
```


RFML data タグ

data タグには、以下の属性を含めることができます。 ブラケット [] で囲まれた属性は、その属性が任意選択であることを示します。 任意選択属性を指定する場合には、ソースにブラケットを含めないでください。 選択項目のリストに示しているように、属性値の中には、 ブレース {} (垂直バー | で区切られた選択可能な項目付き) で囲まれているものもあります。 こうした属性の 1 つを指定する場合には、ソースにはブレースを含めず、 表示される選択項目の 1 つだけを指定してください。

```
<data type="{ char | int | packed | zoned | float | byte | struct }" ]
    [ bidistringtype="{ ST4 | ST5 | ST6 | ST7 | ST8 | ST9 | ST10 | ST11 | DEFAULT }"]
    [ ccsid="{ number | data-name }" ]
    [ count="{ number | data-name }" ]
    [ init="string" ]
    [ length="{ number | data-name }" ]
    [ name="name" ]
    [ precision="number" ]
    [ struct="struct-name" ]>
</data>
```

以下の表では、data タグ属性をリストします。 各エントリーには、属性名、可能な有効値、および属性の説明が含まれています。

属性	値	説明
type=	*char* 文字値。 *char* データ値は、 *java.lang.String* として戻されます。 詳細については、長さの *char* 値を参照してください。 *int* 整数値。 *int* データ値は、 *java.lang.Long* として戻されます。 詳細については、長さおよび精度の *int* 値を参照してください。 *packed* パック 10 進数値。 *packed* データ値は、 *java.math.BigDecimal* として戻されます。 詳細については、 長さおよび精度の *packed* 値を参照してください。 *zoned* ゾーン 10 進数値。 *zoned* データ値は、 *java.math.BigDecimal* として戻されます。 詳細については、 長さおよび精度の *zoned* 値を参照してください。 *float* 浮動小数点値。 length 属性は、バイト数 (4 または 8のいずれか) を指定します。 4 バイト整数は、 *java.lang.Float* として戻されます。 8 バイト整数は、 *java.lang.Double* として戻されま	使われているデータのタイプ (文字、整数、パック、ゾーン、バイト、または構造体) を示します。 データ・タイプが異なれば、長さおよび精度の属性値も異なります。 詳細については、 長さおよび精度の値を参照してください。

	す。 詳細については、 長さの *float* 値を参照してください。 *byte* バイト値。 データの変換が行われることはありません。 *byte* データ値は、*byte* 値の配列 (*byte[]*) として戻されます。 詳細については、長さの *byte* 値を参照してください。 *struct* <struct> 要素の名前。 *struct* を使用すると、1 度構造体を定義しておけば、 その構造体を文書内で何度でも再使用することができます。 type="struct" を使用すると、 指定した構造体が文書のこの位置に表示されたかのようになります。 *struct* では、長さの値を定めることはできません。また、精度の値を持ちません。	
bidistringtype=	*DEFAULT* DEFAULT は、非両方向データ (LTR) のデフォルトのストリング・タイプです。 *ST4* *ST4* は、 ストリング・タイプ 4 です。 *ST5* *ST5* は、ストリング・タイプ 5 です。 *ST6* *ST6* は、ストリング・タイプ 6 です。 *ST7* *ST7* は、ストリング・タイプ 7 です。 *ST8* *ST8* は、ストリング・タイプ 8 です。 *ST9* *ST9* は、ストリング・タイプ 9 です。 *ST10* *ST10* は、 ストリング・タイプ 10 です。 *ST11* *ST11* は、ストリング・タイプ 11 です。	両方向ストリング・タイプを type="char" が付いた <data> 要素に指定します。 この属性を省略すると、この要素のストリング・タイプは、CCSID によって暗黙指定されます (明示的に指定されるか、 ホスト環境のデフォルトの CCSID を取ります)。 ストリング・タイプは、BidiStringType クラスの javadoc で定義されます。

ccsid=	*number* *number* は、固定されていて変更できない CCSID を定義します。 *data-name* *data-name* は、文字データの CCSID (実行時) を含む名前を定義します。 *data-name* には、現行要素の完全修飾名または相対名を指定できます。どちらの場合でも、 その名前は type="int" で定義された <data> の要素を参照する必要があります。 相対名の解決方法に関する詳細は、相対名の解決を参照してください。	<data> 要素の文字データに応じたホストのコード化文字セット識別子 (CCSID) を指定します。 ccsid 属性は、 type="char" が付いた <data> 要素にのみ指定することができます。 この属性を省略すると、この要素の文字データは、ホスト環境のデフォルトの CCSID であると見なされます。
count=	*number* *number* は、固定されていて変更できない、 サイズ配列内の要素数を定義します。 *data-name* *data-name* は、 RFML 文書内の <data> 要素 (実行時の配列内の要素数が入る) の名前を定義します。 *data-name* には、現行要素の完全修飾名または相対名を指定できます。どちらの場合でも、 その名前は type="int" で定義された <data> の要素を参照する必要があります。 相対名の解決方法に関する詳細は、相対名の解決を参照してください。	要素が配列であると指定し、配列内にある項目数を識別します。 count 属性を省略すると、配列として定義されている別の要素に含まれていても、 その要素は配列として定義されません。
init=	*string*	<data> 要素の初期値を指定します。 指定された初期値は、スカラー値の初期設定に使われます。 要素が配列として定義されるか、または配列として定義された構造体に含まれる場合、 指定された初期値は配列内のすべての項目の初期値として使用されます。
length=	*number* *number* は、固定されていて変更できない長さを定義します。 *data-name* *data-name* は、RFML 文書内の <data> (実行時の長さが入る) の名前を定義します。 *data-name* は、 type="char" または type="byte" が付いた <data> 要素にのみ指定することができます。 *data-name* には、現行要素の完全修飾名または相対名を指定できます。 どちらの場合でも、 その名前は type="int" で定義された <data> の要素を参照する必要があります。 相対名の解決方法に関	データ要素の長さを指定します。 この属性値の使用法は、データ・タイプによって変わります。 詳細については、 長さおよび精度の値を参照してください。

	する詳細は、相対名の解決を参照してください。	
name=	*name*	<data> 要素の名前を指定します。
precision=	*number*	ある種の数値データ・タイプの精度のバイト数を指定します。詳細については、 長さおよび精度の値を参照してください。
struct=	*name*	<data> 要素の <struct> 要素の名前を指定します。 struct 属性は、 type="struct" が付いた <data> 要素のみに指定することができます。

«

RFML rfml タグ

rfml タグには、以下の属性を含めることができます。 ブラケット [] で囲まれた属性は、その属性が任意選択であることを示します。 任意選択属性を指定する場合には、ソースにブラケットを含めないでください。

```
<rfml version="version-string"
    [ ccsid="number" ]>
</rfml>
```

以下の表では、rfml タグ属性をリストします。 各エントリーには、属性名、可能な有効値、および属性の説明が含まれています。

属性	値	説明
version=	*version-string* RFML DTD の固定バージョン。 V5R2 では、4.0 が唯一の有効値です。	正確な値の検証のために使用できる、RFML DTD のバージョンを指定します。
ccsid=	*number* 固定した、変更されることがないコード化文字セット ID (CCSID)。	ホスト CCSID を指定します。これは、 CCSID を指定しない、 記号で囲まれた <data type="char"> の要素すべてに適用されます。 詳細については、 RFML <data> タグを参照してください。 この属性を省略すると、ホスト環境のデフォルトの CCSID が使用されます。

RFML recordformat タグ

recordformat タグには、以下の属性を含めることができます。 ブラケット [] で囲まれた属性は、その属性が任意選択であることを示します。 任意選択属性を指定する場合には、ソースにブラケットを含めないでください。

```
<recordformat name="name"
    [ description="description" ]>
</recordformat>
```

以下の表では、recordformat タグ属性をリストします。 各エントリーには、属性名、可能な有効値、および属性の説明が含まれています。

属性	値	記述
name=	*name*	recordformat の名前を指定します。
description=	*description*	recordformat の記述を指定します。

RFML struct タグ

struct タグには、以下の属性を含めることができます。 ブラケット [] で囲まれた属性は、その属性が任意選択であることを示します。 任意選択属性を指定する場合には、ソースにブラケットを含めないでください。

<struct name="*name*">
</struct>

以下の表では、struct タグ属性をリストします。 各エントリーには、属性名、可能な有効値、および属性の説明が含まれています。

属性	値	記述
name=	*name*	<struct> 要素の名前を指定します。

Secure Sockets Layer および Java Secure Socket Extension

IBM Toolbox for Java は、 Java Secure Sockets Layer (SSL) 接続用の Java Secure Socket Extension (JSSE) の使用をサポートしています。 JSSE は、Java 2 Platform、Standard Edition (J2SE)、 バージョン 1.2 および 1.3 のオプションのパッケージとして入手できます。 JSSE は、J2SE、バージョン 1.4 には組み込まれています。

JSSE についての詳細は、 Sun 社の JSSE Web サイト を参照してください。

JSSE は、サーバー認証を行う機能、セキュア通信を可能にする機能、およびデータを暗号化する機能を備えています。 JSSE を使用することにより、クライアントと、 TCP/IP によるアプリケーション・プロトコル (たとえば、HTTP および FTP) を実行するサーバーとの間のセキュア・データ交換を提供することができます。

JSSE のインストールおよび構成の終了後は、Toolbox for Java は JSSE をデフォルトで使用します。 sslight はもはや拡張されないため、JSSE への移行を検討してください。 SSL を使用可能にするための sslight の使用に関する情報は、後方互換性だけのために提供されています。

SSL を IBM Toolbox for Java と共に使うためには、 法的責任を理解しておかなければなりません。

»iSeries システム・デバッガー

iSeries システム・デバッガーは、 iSeries サーバーに新しいグラフィカル・ユーザー・デバッグ環境を提供します。 システム・デバッガーを使用して、 OS/400 PASE 環境を含む iSeries サーバー上で実行されるプログラムをデバッグおよびテストします。

iSeries システム・デバッガーについての詳細は、以下のトピックを参照してください。

コンポーネント
iSeries システム・デバッガーを構成するさまざまなコンポーネントについて、 さらに強力なデバッグ・ツールを提供するためにそれらがどのように連携して機能するかについてお読みください。

インストール
iSeries システム・デバッガーのインストール要件、 およびワークステーションにインストールする方法を見出します。

iSeries システム・デバッガーの実行
さまざまなデバッグ・コンポーネントを実行する方法について参照します。

«

iSeries システム・デバッガーのコンポーネント

iSeries システム・デバッガーには、以下のコンポーネントがあります。

- クライアント・ベースのデバッグ・マネージャー
- クライアント・ベースのシステム・デバッガー
- クライアント・ベースの OS/400 PASE デバッガー
- ホスト・ベースのデバッグ・ハブ
- ホスト・ベースのデバッグ・サーバー

以下の説明では、iSeries システム・デバッガーのコンポーネントについての一般情報のみを掲載しています。 コンポーネントについての詳細を調べるためには、 iSeries システム・デバッガーを実行し、 そのオンライン・ヘルプを参照してください。 iSeries システム・デバッガーのオンライン・ヘルプを表示するには、以下のアクションのいずれかを行ってください。

- いずれかのデバッガー・インターフェースの中の 「ヘルプ (Help)」メニューから、「ヘルプ (Help)」をクリックする
- F1 を押す

デバッグ・マネージャー

デバッグ・マネージャーは、クライアントをデバッグ・ハブで登録します。 デバッグ・ハブは、選択したシステムのグラフィカル・デバッグ・モードを使用可能にします。 登録が終わると、エミュレーション・セッションから 「デバッグの開始 (STRDBG)」CL コマンドを実行したクライアントが、 システム・デバッガーを開始します。

デバッグ・マネージャーを使用して、デバッグ操作および接続を管理します。

- システムを追加および除去する
- ユーザーを追加および除去する
- デバッグ操作を開始する
- システム・デバッガーを立ち上げる

システム・デバッガー

システム・デバッガーは、iSeries サーバー上で実行するプログラムをデバッグするために使用します。 システム上の既存のジョブ内で実行中のプログラムをデバッグすることもできますし、 システム・デバッガーを使用して、 システ

ム・バッチ・ジョブ内のプログラムを立ち上げてからデバッグすることもできます。

システム・デバッガーをセットアップして、自動的に開始するか、 ワークステーションのコマンド・プロンプトから手動で開始するか、 あるいはデバッグ・マネージャー・インターフェースを使用して開始するように設定できます。

システム・デバッガーを使用して、以下を含むデバッグ活動を実行します。

- ブレークポイントの設定
- プログラムのステップスルー
- 変数の検査
- コール・スタックの調査
- プログラム変数に関連するメモリーの調査
- スレッド活動の調査

OS/400 PASE デバッガー

OS/400 PASE デバッガーを使用して OS/400 PASE 環境で実行するプログラムをデバッグします。 システム上の既存のプロセスで実行中のプログラムをデバッグすることもできますし、 OS/400 PASE デバッガーを使用して、プログラムを立ち上げてからデバッグすることもできます。

コマンド行から直接に、あるいはデバッグ・マネージャー・インターフェースを使用して、 OS/400 PASE デバッガーを立ち上げることができます。

システム・デバッガーについてすでに列挙したデバッグ活動に加え、 OS/400 PASE デバッガーを使用して以下を含む OS/400 PASE 特有のデバッグ活動を実行することもできます。

- プログラム・ロード・マップによるデバッグ
- ソース・ファイルおよびメソッドのリストの表示
- 親子プロセスのトラッキング
- レジスターの検査

デバッグ・ハブ

デバッグ・ハブには、次の機能があります。

- システム・デバッガー » または OS/400 PASE デバッガー « を使用したいクライアントのためのレジストリーとして機能する
- デバッグ・サーバーを開始するための着信要求を処理する

デバッグ・ハブでクライアントを登録するには、 デバッグ・マネージャー・イ

ンターフェースを使用します。 クライアントを登録すると、レジストリーにクライアントのユーザー情報と TCP/IP アドレスを格納します。 エミュレーション・セッションから 「デバッグの開始 (STRDBG)」 CL コマンドを使用するとデバッグ・ハブと交信し、 そのコマンドを実行するユーザーがデバッグ・マネージャーで登録済みかどうかを調べます。 デバッグ・ハブはさらに、実行中のコマンドがデバッグ・マネージャーと同じ TCP/IP アドレスからのものかどうかを検査します。 これらの制限を満たしている時、従来のデバッグ環境に代わって、 グラフィカル iSeries システム・デバッグ・アプリケーションが開始されます。

デバッグ・ハブはさらに、 すべての iSeries システム・デバッグ・アプリケーションの単一の接点として機能します。 iSeries システム・デバッガー・ » コンポーネント « がデバッグ開始の操作を行う時、デバッグ・ハブは、 デバッグ・サーバー・ジョブをユーザーの代わりに実行依頼し、 関連した TCP/IP 接続をデバッグ・サーバー・ジョブに渡します。

デバッグ・サーバー

デバッグ・サーバーは、 » デバッガーの 1 つ « がデバッグ開始の要求を出す時、 デバッグ・ハブによって開始される TCP/IP サーバーです。 そのサーバー・ジョブはその後、デバッグされているジョブを保守し、 必要なデバッガー API およびコマンドを実行します。 «

iSeries システム・デバッガーのインストール

iSeries システム・デバッガーを実行するために、クライアント・ワークステーションは、 以下のハードウェア要件およびソフトウェア要件を満たす必要があります。

ハードウェア要件

以下のハードウェアがクライアントに実装されている必要があります。

- CPU: 400～500 MHz
- メモリー: 最低 128 MB (256 MB を推奨)

ソフトウェア要件

以下のソフトウェアがクライアントにインストール済みでなければなりません。

- 次のどちらかです。
  - Java 2 Platform、Standard Edition (J2SE) あるいは Enterprise Edition (J2EE) の、 バージョン 1.3 あるいはそれ以降
  - Java 2 Runtime Environment (JRE)、Standard Edition、バージョン 1.3.1 あるいはそれ以降
- jhall.jar (JavaHelp(TM) にある JAR ファイルの 1 つ)

注: 必ず jhall.jar をクライアントの CLASSPATH 環境変数に追加してください。

上記のソフトウェアのインストールについての情報は、 Sun 社の Java Web サイト を参照してください。

iSeries システム・デバッガーの JAR ファイルのインストール

iSeries システム・デバッガーは、IBM Toolbox for Java の一部としてパッケージされています。 Toolbox for Java の jt400.jar ファイルがまだクライアントにインストール済みでない場合、 iSeries システム・デバッガーをインストールする時にそれをインストールする必要があります。

iSeries システム・デバッガーをインストールする前に、 クライアント・システムが上に挙げられた要件を満たしているかを確認してください。 iSeries システム・デバッガーをインストールするための手順は以下のとおりです。

1. Toolbox for Java をインストールし、 確実に jt400.jar および tes.jar をクライアントにコピーします。

   注: Toolbox for Java をすでにサーバーにインストール済みである場合、jt400.jar ファイルと tes.jar ファイルは両方ともサーバー上の次の同じディレクトリーにあります。

   /QIBM/ProdData/HTTP/Public/jt400/lib/
2. クライアントに JAR ファイルをクライアントにコピーした後、 それらのファイルをクライアントの CLASSPATH 環境変数に追加します。

これで、iSeries システム・デバッガーを実行するためにクライアントを使用できるようになります。 «

iSeries システム・デバッガーの実行

クライアントのコマンド・プロンプトを使用して デバッグ・マネージャーを開始したり、 システム・デバッガーを開始したり、 あるいは OS/400 PASE デバッガーを開始したりすることができます。

iSeries システム・デバッガーについての詳細を調べるには、 iSeries システム・デバッガーを開始し、そのオンライン・ヘルプを参照してください。 iSeries システム・デバッガーのオンライン・ヘルプを表示するには、 以下のアクションのいずれかを行ってください。

- いずれかのデバッガー・インターフェースの中の 「ヘルプ (Help)」メニューから、「ヘルプ (Help)」をクリックする
- F1 を押す

デバッグ・マネージャーの開始

クライアント上のコマンド・プロンプトからデバッグ・マネージャーを開始するには、次のコマンドを実行します。

```
java utilities.DebugMgr
```

システム・デバッガーの開始

クライアント上のコマンド・プロンプトからシステム・デバッガーを開始するには、次のコマンドを実行します。

```
java utilities.Debug <args>
```

ここで、<args> は以下のコマンド引き数のいずれかを表します。

- -u = ユーザー
- -s = システム名
- -j = ジョブ記述を次の形式で: ジョブ番号/ジョブ・ユーザー/ジョブ名
- -p = 実行するプログラムを次の形式で: プログラム・ライブラリー/プログラム名

注: デバッグ・マネージャーを使用してクライアントの登録が終わると、 エミュレーション・セッションから「デバッグの開始 (STRDBG)」CL コマンドを実行して、 システム・デバッガーを開始できます。 さらに、システム・デバッグ・マネージャーから直接システム・デバッガーを立ち上げることもできます。

OS/400 PASE デバッガーの開始

クライアント上のコマンド・プロンプトから OS/400 デバッガーを開始するには、次のコマンドを実行します。

```
java utilities.DebugPASE <args>
```

ここで、<args> は以下のコマンド引き数のいずれかを表します。

- -u = ユーザー
- -s = システム名
- -p = 実行するプログラムの完全修飾パス
- -pid = プロセス ID

注：システム・デバッグ・マネージャーから直接 OS/400 PASE デバッガーを立ち上げることもできます。 システム・デバッガーとは異なり、 エミュレーター・セッションから OS/400 PASE デバッガーを立ち上げることはできません。

iSeries システム・デバッガーについての詳細を調べるためには、 iSeries システム・デバッガーを実行し、そのオンライン・ヘルプを参照してください。 iSeries システム・デバッガーのオンライン・ヘルプを表示する方法は次のとおりです。

- いずれかの iSeries システム・デバッガー・インターフェースの中の 「ヘルプ (Help)」メニューから、「ヘルプ (Help)」をクリックする
- いずれかの iSeries システム・デバッガー・インターフェースから、F1 を押す ≪

例: プロパティー・ファイル

```
#=========================================================#
# IBM Toolbox for Java                                    #
#---------------------------------------------------------#
# Sample properties file                                  #
#                                                         #
# Name this file jt400.properties and store it in a       #
# com/ibm/as400/access directory that is pointed to by    #
# the classpath.                                          #
#=========================================================#

#---------------------------------------------------------#
# Proxy server system properties                          #
#---------------------------------------------------------#

# This system property specifies the proxy server host name
# and port number, specified in the format: hostName:portNumber
# The port number is optional.
com.ibm.as400.access.AS400.proxyServer=hqoffice

# This system property specifies which portion of the proxy
# data flow is encrypted via SSL.  Valid values are:
#   1 - Proxy client to proxy server
#   2 - Proxy server to AS/400
#   3 - Proxy client to proxy, and proxy server to AS/400
com.ibm.as400.access.SecureAS400.proxyEncryptionMode=1

# This system property specifies how often, in seconds,
# the proxy server will look for idle connections. The
# proxy server starts a thread to look for clients that are
# no longer communicating. Use this property to set how
# often the thread looks for idle connections.
com.ibm.as400.access.TunnelProxyServer.clientCleanupInterval=7200

# This system property specifies how long, in seconds, a
# client can be idle before it is cleaned up. The proxy server
# starts a thread to look for clients that are no longer
# communicating. Use this property to set long a client can
# be idle before it is cleaned up.
com.ibm.as400.access.TunnelProxyServer.clientLifetime=2700
```

```
#-------------------------------------------------------#
# Trace system properties                               #
#-------------------------------------------------------#

# This system property specifies which trace categories to enable.
# This is a comma-delimited list containing any combination of trace
# categories.  The complete list of trace categories is defined in
# the Trace class.
com.ibm.as400.access.Trace.category=error,warning,information

# This system property specifies the file to which trace output
# is written.  The default is to write trace output to System.out.
com.ibm.as400.access.Trace.file=c://temp//trace.out


#-------------------------------------------------------#
# Command Call system properties                        #
#-------------------------------------------------------#

# This system property specifies whether CommandCalls should
# be assumed to be thread-safe. If true, all CommandCalls are
# assumed to be thread-safe. If false, all CommandCalls are
# assumed to be non-thread-safe. This property is ignored
# for a given CommandCall object if either
# CommandCall.setThreadSafe(true/false) or
# AS400.setMustUseSockets(true) has been performed on the object.
com.ibm.as400.access.CommandCall.threadSafe=true


#-------------------------------------------------------#
# Program Call system properties                        #
#-------------------------------------------------------#

# This system property specifies whether ProgramCalls should
# be assumed to be thread-safe. If true, all ProgramCalls are
# assumed to be thread-safe. If false, all ProgramCalls are
# assumed to be non-thread-safe. This property is ignored
# for a given ProgramCall object if either
# ProgramCall.setThreadSafe(true/false) or
# AS400.setMustUseSockets(true) has been performed on the object.
```

```
com.ibm.as400.access.ProgramCall.threadSafe=true

# End
```

例: システム・プロパティーのクラス・ソース・ファイル

```
//=============================================================
// IBM Toolbox for Java
//-------------------------------------------------------------
// Sample properties class source file
//
// Compile this source file and store the class file in
// the classpath.
//=============================================================
package com.ibm.as400.access;

public class Properties
extends java.util.Properties
{
   public Properties ()
   {
      /*-----------------------------------------------------------*/
      /* Proxy server system properties                            */
      /*-----------------------------------------------------------*/

      // This system property specifies the proxy server host name
      // and port number, specified in the format: hostName:portNumber
      // The port number is optional.
      put ("com.ibm.as400.access.AS400.proxyServer", "hqoffice");

      // This system property specifies which portion of the proxy
      // data flow is encrypted via SSL.  Valid values are:
      //   1 - Proxy client to proxy server
      //   2 - Proxy server to iSeries or AS/400e server
      //   3 - Proxy client to proxy, and proxy server to iSeries or AS/400e server
      put("com.ibm.as400.access.SecureAS400.proxyEncryptionMode", "1");

      // This system property specifies how often, in seconds,
      // the proxy server will look for idle connections. The
      // proxy server starts a thread to look for clients that are
      // no longer communicating. Use this property to set how
      // often the thread looks for idle connections.
      put("com.ibm.as400.access.TunnelProxyServer.clientCleanupInterval", "7200");

      // This system property specifies how long, in seconds, a
      // client can be idle before it is cleaned up. The proxy server
      // starts a thread to look for clients that are no longer
      // communicating. Use this property to set long a client can
      // be idle before it is cleaned up.
      put("com.ibm.as400.access.TunnelProxyServer.clientLifetime", "2700");

      /*-----------------------------------------------------------*/
```

```
        /* Trace system properties                                  */
        /*----------------------------------------------------------*/

        // This system property specifies which trace categories to enable.
        // This is a comma-delimited list containing any combination of trace
        // categories.  The complete list of trace categories is defined in
        // the Trace class.
        put ("com.ibm.as400.access.Trace.category", "error,warning,information");

        // This system property specifies the file to which trace output
        // is written.  The default is to write trace output to System.out.
        put ("com.ibm.as400.access.Trace.file", "c:/temp/trace.out");


        /*----------------------------------------------------------*/
        /* Command Call system properties                           */
        /*----------------------------------------------------------*/

        // This system property specifies whether CommandCalls should
        // be assumed to be thread-safe. If true, all CommandCalls are
        // assumed to be thread-safe. If false, all CommandCalls are
        // assumed to be non-thread-safe. This property is ignored
        // for a given CommandCall object if either
        // CommandCall.setThreadSafe(true/false) or
        // AS400.setMustUseSockets(true) has been performed on the object.
        put ("com.ibm.as400.access.CommandCall.threadSafe", "true");


        /*----------------------------------------------------------*/
        /* Program Call system properties                           */
        /*----------------------------------------------------------*/

        // This system property specifies whether ProgramCalls should
        // be assumed to be thread-safe. If true, all ProgramCalls are
        // assumed to be thread-safe. If false, all ProgramCalls are
        // assumed to be non-thread-safe. This property is ignored
        // for a given ProgramCall object if either
        // ProgramCall.setThreadSafe(true/false) or
        // AS400.setMustUseSockets(true) has been performed on the object.
        put ("com.ibm.as400.access.ProgramCall.threadSafe", "true");
    }
}
```

»IBM Toolbox for Java 2 Micro Edition

IBM Toolbox for Java 2 Micro Edition パッケージ (com.ibm.as400.micro) により、Java プログラムの作成が可能になり、 携帯情報端末 (PDA) や携帯電話のような種々の Tier0 ワイヤレス・デバイスを使用して、 iSeries のデータおよびリソースに直接アクセスすることができます。

ToolboxME for iSeries の詳細については、以下のトピックを参照してください。

要件
ToolboxME for iSeries によってアプリケーションを開発する際の、および Tier0 デバイス上でそれらのアプリケーションを実行する際の要件がわかります。

ToolboxME for iSeries をダウンロードしてセットアップする
サーバー、ワークステーション、および Tier0 デバイスに ToolboxME for iSeries をダウンロードしてセットアップする方法を学べます。

概念
Tier0 デバイスで実行するアプリケーションの開発に重要な概念について定義している、 簡単な概要をお読みください。

ToolboxME for iSeries クラス
ToolboxME for iSeries コンポーネント (com.ibm.as400.micro package) のクラスについて参照してください。 このクラスは、Toolbox for Java アクセス・クラス、 JDBC サポートなどで使用可能な、削減された機能の集合を提供します。

ToolboxME for iSeries プログラムを作成する
Tier0 デバイスで実行する ToolboxME for iSeries プログラムの作成手順について理解します。 指示に従って、初めての ToolboxME for iSeries プログラムを作成します。

例
ToolboxMe for iSeries の作業例を調べ、ダウンロードして、実行すると、 ワイヤレス・アプリケーションを作成して使用する方法を理解するのに役立ちます。 «

ToolboxME for iSeries をダウンロードしてセットアップする

JTOpen に含まれる、ToolboxME for iSeries (jt400Micro.jar) を別個にダウンロードする必要があります。 ToolboxME for iSeries は、そのセットアップについての追加情報も提供している IBM Toolbox for Java/JTOpen Web サイト  からダウンロードできます。

ToolboxME for iSeries のセットアップ方法は、Tier0 デバイス、開発ワークステーション、 およびサーバーでそれぞれ異なります。

- ワイヤレス・デバイス用のアプリケーションを構築し (jt400Micro.jar を使用して)、 デバイス・メーカーによって文書化されているようにアプリケーションをインストールします。
- iSeries ホスト・サーバーが、 ターゲットのデータを含むサーバー上で開始されていることを確認してください。
- MEServer を実行するシステムが jt400.jar へのアクセス権を持っていることを確認してください。 詳細については、Toolbox for Java をワークステーションにインストールするおよび Toolbox for Java を iSeries サーバーにインストールするを参照してください。

ToolboxME for iSeries を使用する上で重要な概念

ToolboxME for iSeries Java アプリケーションの開発を始める前に、 そうした開発を統制する以下の概念および規格を理解する必要があります。

Java 2 Platform, Micro Edition (J2ME)

J2ME(TM) は、携帯情報端末 (PDA) および携帯電話のような、 Tier0 ワイヤレス・デバイス用の Java runtime Environment を提供する Java 2 規格のインプリメンテーションです。 IBM Toolbox for Java 2 Micro Edition は、この規格に従います。

Tier0 デバイス

ワイヤレス・テクノロジーを使用してコンピューターおよびネットワークに接続する、 PDA および携帯電話のようなワイヤレス・デバイスは、Tier0 デバイスと呼ばれます。 この名前は、一般的な 3 層アプリケーション・モデルに基づきます。 3 層モデルは、それぞれが単一のコンピューターまたはネットワーク上に常駐する、 3 つの部分から編成される分散プログラムを説明するものです。

- 3 番目の層は、多くの場合 2 番目の層とは別のサーバーに常駐するデータベースおよび関連するプログラムです。 この層は、他の層が作業を実行するのに使用する情報、およびその情報へのアクセスを提供します。
- 2 番目の層はビジネス・ロジックで、ネットワークを共用する別のコンピューター上、通常はサーバー上にあります。
- 1 番目の層は普通アプリケーションの部分で、 ユーザー・インターフェースを含むワークステーションに常駐します。

Tier0 デバイスは多くの場合、PDA および携帯電話のような、 小型でポータブルな、リソースが制約されるデバイスです。 Tier0 デバイスは、1 番目の層のデバイスの機能に代わる、あるいはその機能を補います。

Connected Limited Device Configuration (CLDC)

より大規模なデバイスで必要とされる機能を提供するために、 最少の API および Java 仮想マシンの必要な能力を定義する構成。 CLDC は、Tier0 デバイスを含む、リソースが制約された広範なデバイスのセットをターゲットとします。

詳細については、 CLDC and the K Virtual Machine (KVM) を参照してくださ

い。

Mobile Information Device Profile (MIDP)

特定のタイプのデバイスまたはオペレーティング・システムをターゲットとする既存の構成に構築される、 API の集合を表すプロファイル。 CLDC に構築される MIDP は、Tier0 デバイスにアプリケーションおよびサービスを動的に配置できるようにする、 標準の実行時環境を提供します。

詳細については、 Mobile Information Device Profile (MIDP) を参照してください。

ワイヤレス・デバイス用の Java 仮想マシン

Java アプリケーションを実行するには、Tier0 デバイスは、 ワイヤレス・デバイスの限定リソース用に特別に設計された Java 仮想マシンを必要とします。 使用できる適切な JVM には以下のものが含まれます。

- IBM J9 仮想マシン (IBM WebSphere Micro Environment の一部)
- Sun K Virtual Machine (KVM)
- MIDP

関連情報

ワイヤレスの Java アプリケーションを構築する助けとして作成された、多くの開発ツールのいずれかを使用できます。 そのようなツールの簡単なリストに関しては、 IBM Toolbox for Java の関連情報を参照してください。

ワイヤレス・デバイスのシミュレーターおよびエミュレーターの詳細を理解するため、 およびそれらをダウンロードするためには、 アプリケーションを実行するデバイスまたはオペレーティング・システムの Web サイトを参照してください。 «

» ToolboxME for iSeries クラス

com.ibm.as400.micro パッケージは、 Tier 0 デバイスが iSeries サーバーのデータおよびリソースにアクセスできるようにするアプリケーションを作成するのに必要なクラスを提供します。

注: ToolboxMe for iSeries クラスを使うには、 ToolboxME for iSeries コンポーネントを別個にダウンロードしてセットアップする必要があります。

ToolboxME for iSeries には以下のクラスがあります。

- MEServer は、 Tier0 デバイスからホスト・サーバーへの要求を仲介します。
- いくつかのクラスは、Toolbox for Java アクセス・パッケージからの機能のサブセットを提供します。
  - AS400 - iSeries システムにサインオンする
  - CommandCall - iSeries コマンドを呼び出す
  - DataQueue - iSeries サーバーのデータ待ち行列から読み取り、データ待ち行列に書き込む
  - ProgramCall - iSeries サーバー・プログラムを呼び出し、 プログラムの実行後に戻されるデータにアクセスする
- 他のクラスは JDBC サポートを提供し、 それらには最小限のメソッドおよび java.sql パッケージからのデータも含まれます。 «

»MEServer クラス

MEServer クラスを使用して、 ToolboxME for iSeries の jar ファイルを使用する Tier0 クライアント・アプリケーションからの要求を実行します。 MEServer は Toolbox for Java オブジェクトを作成し、 それらのオブジェクトでクライアント・アプリケーション用にメソッドを呼び出します。

注: ToolboxMe for iSeries クラスを使うには、 ToolboxME for iSeries コンポーネントを別個にダウンロードしてセットアップする必要があります。 詳細は、 ToolboxME for iSeries をダウンロードしてセットアップするを参照してください。

以下のコマンドを使用して、MEServer を開始します。

```
java com.ibm.as400.micro.MEServer [options]
```

ここで [options] は、1 つまたは複数の次のオプションです。

-pcml pcml-doc1 [;pcml_doc2;...]

プリロードおよび構文解析する PCML 文書を指定します。 -pc を使用してこのオプションを短縮できます。

このオプションの使用に関する重要な情報については、 MEServer javadoc を参照してください。

-port port

ポートを指定してクライアントからの接続を受け入れるために使用します。 このオプションは、-po と短縮することができます。デフォルトのポートは 3470 です。 -po を使用してこのオプションを短縮できます。

-verbose [true|false]

System.out に状況および接続情報を印刷するかどうかを指定します。 -v を使用してこのオプションを短縮できます。

-help

System.out に使用法についての情報をプリントします。 -h または -? を使用してこのオプションを短縮できます。 デフォルトでは、この使用法についての情報をプリントしません。

指定されているポートで別のサーバーがすでにアクティブになっている場合、MEServer は開始されません。 «

»AS400 クラス

micro パッケージの AS400 クラス (com.ibm.as400.micro.AS400) は、 アクセス・パッケージの AS400 クラス (com.ibm.as400.access.AS400) で使用可能な変更済みの機能のサブセットを提供します。 ToolboxMe for iSeries AS400 クラスを使用して、Tier0 デバイスから iSeries システムにサインオンします。

注: ToolboxMe for iSeries クラスを使うには、 ToolboxME for iSeries コンポーネントを別個にダウンロードしてセットアップする必要があります。 詳細は、ToolboxME for iSeries をダウンロードしてセットアップするを参照してください。

AS400 クラスには、以下の機能があります。

- MEServer に接続する
- MEServer から切断する

MEServer への接続は、暗黙的に作成されます。 たとえば、AS400 オブジェクトを作成した後、 CommandCall の run() メソッドを使用して connect() を自動的に実行できます。 言い換えれば、接続が確立する際に制御することを望まないなら、connect() メソッドを明示的に呼び出す必要はありません。

以下の例は、AS400 クラスを使用して iSeries システムにサインオンする方法を示します。

```
AS400 system = new AS400("mySystem", "myUserid", "myPwd", "myMEServer");
try
{
    system.connect();
}
catch (Exception e)
{
    // Handle the exception
}
// Done with the system object.
system.disconnect();
```

«

CommandCall クラス

micro パッケージの CommandCall クラス (com.ibm.as400.micro.AS400) は、 access パッケージの CommandCall クラス (com.ibm.as400.access.AS400) で使用可能な変更済みの機能のサブセットを提供します。 CommandCall クラスを使用して、Tier0 デバイスから iSeries コマンドを呼び出します。

注: ToolboxMe for iSeries クラスを使うには、 ToolboxME for iSeries コンポーネントを別個にダウンロードしてセットアップする必要があります。

CommandCall run() メソッドは、ストリング (実行するコマンド) を必要とし、 コマンドを実行した結果生じるすべてのメッセージをストリングとして戻します。 コマンドが完了したもののメッセージを何も生成しない場合、 run() メソッドは空のストリング配列を戻します。

以下の例は、CommandCall の使用方法を示します。

```
// Work with commands.
AS400 system = new AS400("mySystem", "myUserid", "myPwd", "myMEServer");
try
{
    // Run the command "CRTLIB FRED."
    String[] messages = CommandCall.run(system, "CRTLIB FRED");
    if (messages != null)
    {
        // Note that there was an error.
        System.out.println("Command failed:");
        for (int i = 0; i < messages.length; ++i)
        {
            System.out.println(messages[i]);
        }
    }
    else
    {
        System.out.println("Command succeeded!");
    }
}
catch (Exception e)
{
    // Handle the exception
}
// Done with the system object.
system.disconnect();
```

DataQueue クラス

micro パッケージの DataQueue クラス (com.ibm.as400.micro.AS400) は、 access パッケージの DataQueue クラス (com.ibm.as400.access.AS400) で使用可能な変更済みの機能のサブセットを提供します。 DataQueue クラスを使用して、 Tier0 デバイスが iSeries サーバー上のデータ待ち行列を読み取る、 またはデータ待ち行列に書き込むようにします。

注: ToolboxMe for iSeries クラスを使うには、 ToolboxME for iSeries コンポーネントを別個にダウンロードしてセットアップする必要があります。

DataQueue クラスには、以下のメソッドが含まれます。

- エントリーをストリングとして読み取るまたは書き込む
- エントリーをバイトの配列として読み取るまたは書き込む

エントリーを読み取るまたは書き込むには、データ待ち行列が常駐する iSeries システムの名前、 およびデータ待ち行列の完全修飾された統合ファイル・システム・パス名を提供する必要があります。 使用可能なエントリーがない場合に、エントリーを読み取るとヌル値が戻されます。

以下の例は、DataQueue クラスを使用して iSeries システム上のデータ待ち行列からエントリーを読み取る方法と、 データ待ち行列にエントリーを書き込む方法を示します。

```
AS400 system = new AS400("mySystem", "myUserid", "myPwd", "myMEServer");
try
{
    // Write to the Data Queue.
    DataQueue.write(system, "/QSYS.LIB/FRED.LIB/MYDTAQ.DTAQ", "some text");

    // Read from the Data Queue.
    String txt = DataQueue.read(system, "/QSYS.LIB/FRED.LIB/MYDTAQ.DTAQ");
}
catch (Exception e)
{
    // Handle the exception
}
// Done with the system object.
system.disconnect();
```

»ProgramCall クラス

micro パッケージの ProgramCall クラス (com.ibm.as400.micro.ProgramCall) は、 access パッケージの ProgramCall クラス (com.ibm.as400.access.ProgramCall) で使用可能な変更済みの機能のサブセットを提供します。 ProgramCall クラスを使用し、 Tier0 デバイスを使用可能にして iSeries プログラムを呼び出し、 プログラムが実行された後に戻されるデータにアクセスします。

注: ToolboxMe for iSeries クラスを使うには、 ToolboxME for iSeries コンポーネントを別個にダウンロードしてセットアップする必要があります。 詳細については、ToolboxME for iSeries の要件およびインストールを参照してください。

ProgramCall.run() メソッドを使用するには、以下のパラメーターを提供する必要があります。

- プログラムを実行するシステム
- プログラム呼び出しマークアップ言語 (PCML) 文書の名前
- 実行するプログラムの名前
- 設定する 1 つ以上のプログラム・パラメーターの名前および関連する値を含むハッシュ・テーブル
- プログラムが実行された後に戻されるすべてのパラメーターの名前を含むストリング配列

ProgramCall は PCML を使用して、プログラムの入出力パラメーターを記述します。 PCML ファイルは MEServer と同じマシン上になければならず、 そのマシンの CLASSPATH の PCML ファイルを含むディレクトリーのエントリーを持っている必要があります。

各 PCML 文書を MEServer で登録する必要があります。 PCML 文書の登録は、実行する PCML 定義済みプログラムを MEServer に単に指示します。 実行時、または MEServer の開始時のどちらかに PCML 文書を登録します。

プログラム・パラメーターを含むハッシュ・テーブルまたは PCML 文書の登録方法についての詳細は、ToolboxME for iSeries ProgramCall javadoc を参照してください。 PCML の詳細については、 プログラム呼び出しマークアップ言語を参照してください。

以下の例は、Tier0 デバイスを用いてサーバー上のプログラムを実行するための、 ProgramCall クラスの使用方法を示しています。

```
// Call programs.
AS400 system = new AS400("mySystem", "myUserid", "myPwd", "myMEServer");

String pcmlName = "qsyrusri.pcml"; // The PCML document describing the program we want to use.
String apiName = "qsyrusri";

Hashtable parametersToSet = new Hashtable();
parametersToSet.put("qsyrusri.receiverLength", "2048");
parametersToSet.put("qsyrusri.profileName", "JOHNDOE" };

String[] parametersToGet = { "qsyrusri.receiver.userProfile",
            "qsyrusri.receiver.previousSignonDate",
            "qsyrusri.receiver.previousSignonTime",
            "qsyrusri.receiver.displaySignonInfo" };

String[] valuesToGet = null;

try
{
    valuesToGet = ProgramCall.run(system, pcmlName, apiName, parametersToSet, parametersToGet);

    // Get and display the user profile.
    System.out.println("User profile: " + valuesToGet[0]);
```

```
        // Get and display the date in a readable format.
        char[] c = valuesToGet[1].toCharArray();
        System.out.println("Last Signon Date: " + c[3]+c[4]+"/"+c[5]+c[6]+"/"+c[1]+c[2] );

        // Get and display the time in a readable format.
        char[] d = valuesToGet[2].toCharArray();
        System.out.println("Last Signon Time: " + d[0]+d[1]+":"+d[2]+d[3]);

        // Get and display the signon info.
        System.out.println("Signon Info: " + valuesToGet[3] );
    }
    catch (MEException te)
    {
        // Handle the exception.
    }
    catch (IOException ioe)
    {
        // Handle the exception
    }

    // Done with the system object.
    system.disconnect();
```

«

JdbcMe クラス

ToolboxME for iSeries クラスは、java.sql パッケージのサポートを含め、 JDBC サポートを提供します。 このクラスは、Tier 0 デバイスで実行されるプログラムで使用することになっています。

以下のセクションでは、それぞれの JdbcMe クラスに関する情報へのリンクを含め、 データにアクセスして使用することについて論じ、 JdbcMe の内容について説明します。

データにアクセスして使用する

Tier0 デバイスを使用してデータにアクセスして更新する際、 まるでご自分のオフィスのシステムに向かっているかのように、正確に動作することが望まれます。 しかし、多くの Tier0 デバイスの開発ではデータの同期に照準を合わせています。 データの同期を使用すると、 各 Tier0 デバイスはメイン・データベースからの特定のデータのコピーを持ちます。 定期的に、ユーザーはメイン・データベースと各デバイス上のデータを同期します。

データの同期は、動的なデータをうまく処理できません。 動的データを処理するには、最新のデータに即時にアクセスすることが必要です。 アクセスするのにデータが同期化されるまで待たなければならないというのは、 多くのビジネスマンにとって選択肢とはなりません。 加えて、メインの同期データへのサーバーおよびデバイス用のソフトウェアおよびハードウェア要求は、 かなり多くなる可能性があります。

データ同期モデルに固有のこの問題を解決するには、 ToolboxME for iSeries の JdbcMe クラスがメイン・データベースのライブ更新およびアクセスを可能にします。 それでも、オフラインのデータ・ストレージも可能です。 アプリケーションは、メイン・データベースの一部となってすぐにライブ更新する能力を失うことなく、 オフラインの大切なデータへのアクセス権を持つことができます。 この中間的方法は、同期データ・モデルおよびライブ・データ・モデルの双方にとって益となります。

JdbcMe の内容

定義によれば、Tier0 デバイスのすべての種類のドライバーは、 非常に小さくなければなりません。 しかし JDBC API はたいへん大きなものです。 JdbcMe クラスをかなり小さくし、なおかつ多くの JDBC インターフェースをサポートし、 Tier0 デバイスを使用して十分な作業を実行できるようにする必要があります。

JdbcMe クラスは以下の JDBC 機能を提供します。

- データを挿入および更新する能力
- トランザクション制御およびトランザクション分離レベルを変更する能力
- スクロール可能でかつ更新可能な ResultSet
- ストアード・プロシージャーおよびドライブ・トリガーを呼び出すための SQL サポート

加えて、JdbcMe クラスにはいくつかの独特な機能が含まれます。

- 構成の詳細の大部分をサーバー・サイドの単一ポイントに統合できるようにする汎用ドライバー
- データをオフライン・ストレージに保つための標準規格

JdbcMe には以下のクラスが含まれます。

- JdbcMeConnection
- JdbcMeDriver
- JdbcMeException
- JdbcMeLiveResultSet
- JdbcMeOfflineData
- JdbcMeOfflineResultSet
- JdbcMeResultSetMetaData
- JdbcMeStatement

SQL 準拠

ToolboxME for iSeries には、JDBC 仕様に従う Java.sql パッケージがありますが、 有用なクラスおよびメソッドの最小限の集合だけを含みます。 SQL 機能の最小限の集合を提供することにより、JdbcMe クラスのサイズを小さくし、なおかつ共通の JDBC タスクを実行するのに十分役立ちます。 «

» ToolboxME for iSeries を使用してホスト・サーバー上のデータベースに接続する

JdbcMeConnection クラスは、 Toolbox for Java の AS400JDBCConnection クラスで使用可能な機能のサブセットを提供します。 JdbcMeConnection を使用して、Tier0 デバイスがホスト・サーバー上の DB2 Universal Database (UDB) データベースにアクセスできるようにします。

注: ToolboxMe for iSeries クラスを使うには、 ToolboxME for iSeries コンポーネントを別個にダウンロードしてセットアップする必要があります。 詳細については、ToolboxME for iSeries の要件およびインストールを参照してください。

JdbcMeDriver.getConnection() を使用して、サーバー・データベースに接続します。 getConnection() メソッドは、引き数、ユーザー ID、およびパスワードに URL ストリングを取ります。 ホスト・サーバー上の JDBC ドライバー・マネージャーは、 URL で表されるデータベースに接続可能なドライバーがある場所を特定しようとします。 JdbcMeDriver は以下の構文を URL に使用します。

```
jdbc:as400://server-name/default-schema;meserver=<server>[:port];[other properties];
```

サーバー名を指定する必要があります。指定しないと JdbcMeDriver が例外を出します。 デフォルト・スキーマはオプション指定になります。 ポートを指定しない場合、JdbcMeDriver はポート 3470 を使用します。 さらに、URL にさまざまな JDBC プロパティーを設定できます。 プロパティーを設定するには、以下の構文を使用します。

```
name1=value1;name2=value2;...
```

JdbcMeDriver によってサポートされるプロパティーの完全なリストについては、 JDBC プロパティーを参照してください。

例: JdbcMeDriver ドライバーを使用したサーバー・データベースへの接続

例 1: デフォルト・スキーマ、ポート、 または JDBC プロパティーを指定せずにサーバー・データベースに接続します。 ユーザー ID およびパスワードはメソッドのパラメーターとして指定されます。

```
        // Connect to system 'mysystem'. No default schema, port or
        // properties are specified.
    Connection c  = JdbcMeDriver.getConnection
        ("jdbc:as400://mysystem.helloworld.com;
          meserver=myMeServer;"
         "auser",
         "apassword");
```

例 2: スキーマおよび JDBC プロパティーが指定される場合に、 サーバー・データベースに接続します。 ユーザー ID およびパスワードはメソッドのパラメーターとして指定されます。

```
        // Connect to system 'mysystem'. Specify a schema and
        // two JDBC properties. Do not specify a port.
    Connection c2 = JdbcMeDriver.getConnection
        ("jdbc:as400://mysystem.helloworld.com/mySchema;
```

```
        meserver=myMeServer;
        naming=system;
        errors=full;"
        "auser",
        "apassword");
```

例 3: サーバー・データベースに接続します。 URL を使ってプロパティー (ユーザー ID およびパスワードを含む) が指定されます。

```
        // Connect using properties. The properties are set on the URL
        // instead of through a properties object.
    Connection c = DriverManager.getConnection
        ("jdbc:as400://mySystem;
          meserver=myMeServer;
          naming=sql;
          errors=full;
          user=auser;
          password=apassword");
```

例 4: データベースから切断します。 Connecting オブジェクト上で close() メソッドを使用して、サーバーから切断します。

```
    c.close();
```

«

»JdbcMeDriver クラス

JdbcMeDriver クラスは、 Toolbox for Java の AS400JDBCDriver クラス で使用可能な機能のサブセットを提供します。 Tier0 クライアント・アプリケーションで JdbcMeDriver を使用し、 パラメーターを全く持たない簡単な SQL ステートメントを実行し、 そのステートメントが生成する ResultSets を取得します。

注: ToolboxMe for iSeries クラスを使うには、 ToolboxME for iSeries コンポーネントを別個にダウンロードしてセットアップする必要があります。 詳細は、 ToolboxME for iSeries をダウンロードしてセットアップするを参照してください。

明示的には JdbcMeDriver を登録しません。 その代わりに、 JdbcMeConnection.getConnection() メソッドの URL で指定する driver プロパティーが ドライバーを判別します。 たとえば、 IBM Developer Kit for Java の JDBC ドライバー (「ネイティブ」ドライバーと呼ばれる) をロードするには、 以下のコードを使用します。

```
Connection c = JdbcMeDriver.getConnection
     ("jdbc:as400://mysystem.helloworld.com;
       meserver=myMeServer;
       driver=native;
       user=auser;
       password=apassword");
```

IBM Toolbox for Java の JDBC ドライバーは、 サーバーからデータを取得する他の IBM Toolbox for Java クラスと同様に、 入力パラメーターとしての AS400 オブジェクトは必要としません。 しかし、AS400 オブジェクトは内部的に使用され、 ユーザー ID およびパスワードを明示的に提供する必要があります。 ユーザー ID およびパスワードを、URL に、 または getConnection() メソッドのパラメーターとして提供します。

getConnection() の使用例については、 JDBCMeConnection を参照してください。 «

»ResultSet

ToolboxME for iSeries ResultSet クラスは、以下のとおりです。

- JdbcMeLiveResultSet
- JdbcMeOfflineResultSet
- JdbcMeResultSetMetaData

JdbcMeLiveResultSet および JdbcMeOfflineResultSet には、以下を除き、同じ機能が含まれます。

- JdbcMeLiveResultSet は、サーバー上のデータベースに呼び出しを行い、データを検索します。
- JdbcMeOfflineResultSet は、ローカル・デバイス上のデータベースからデータを検索します。

注: ToolboxMe for iSeries クラスを使うには、 ToolboxME for iSeries コンポーネントを別個にダウンロードしてセットアップする必要があります。 詳細は、 ToolboxME for iSeries をダウンロードしてセットアップするを参照してください。

JdbcMeLiveResultSet

JdbcMeLiveResultSet クラスは、 Toolbox for Java の AS400JDBCResultSet クラスで使用可能な機能のサブセットを提供します。 Tier0 クライアント・アプリケーションで JdbcMeLiveResultSet を使用して、 照会の実行によって生成されたデータ・テーブルにアクセスします。

JdbcMeLiveResultSet はテーブル行を順次検索します。 行内では、どのような順序でも列値にアクセスできます。 JdbcMeLiveResultSet には、以下のアクションの実行を可能にするメソッドが含まれます。

- ResultSet に保管されるさまざまなタイプのデータを検索する
- 指定した行にカーソルを移動する (直前の行、現在行、次の行など)
- 行を挿入、 更新、 および削除する
- 列を更新する (String および int 値を使う)
- ResultSet の列を記述する ResultSetMetaData オブジェクトを検索する

ResultSet は、Java プログラムがアクセスしている ResultSet 内の行を指すのに、 カーソル (内部ポインター) を使用します。 JDBC 2.0 には、データベース内の特定の位置へアクセスするための追加メソッドがあります。

スクロール可能カーソル位置	
absolute	moveToInsertRow
first	previous
last	relative
moveToCurrentRow	

スクロール機能

ステートメントを実行して ResultSet が作成されたら、 テーブル内の行を逆方向 (最後から最初へ) または順方向 (最初から最後へ) で移動 (スクロール) することができます。

このような移動をサポートしている ResultSet を、 スクロール可能 ResultSet と呼びます。 スクロール可能 ResultSet は、 相対位置決めおよび絶対位置決めもサポートしています。 相対位置決め

では、 現在行を基準として相対的な位置を指定し、 ResultSet 内の行に移動することができます。 絶対位置決めでは、 ResultSet 内での位置を指定し、 目的の行へ直接移動することができます。

JDBC 2.0 を使用すれば、 ResultSet クラスで作業する際に 2 つの追加スクロール機能 (スクロール非認識 ResultSet とスクロール認識 ResultSet) を使用することができます。

スクロール非認識 ResultSet は、オープン中に行われた変更を通常認識しません。 それに対し、スクロール認識 ResultSet は変更を認識します。 » IBM Toolbox for Java JDBC ドライバーは、スクロール非認識 ResultSet をサポートしていません。 «

更新可能 ResultSet

アプリケーションで ResultSet を使用する際に、 同時読み取り専用か同時更新可能のいずれかを使用することができます。 同時読み取り専用ではデータを更新できないのに対し、 同時更新可能ではデータへの更新を行うことができ、 さらにデータベースへの書き込みをロックして、 別のトランザクションによる同じデータ項目へのアクセスを制御することもできます。 更新可能 ResultSet では、行の更新、挿入、および削除を行うことができます。

JdbcMeResultSet で使用できる更新メソッドの詳細なリストについては、メソッドの索引を参照してください。

例: 更新可能 ResultSet

以下の例は、 データを更新でき (同時更新)、オープン中に ResultSet へ変更を行うことのできる (スクロール認識) ResultSet を使用する方法を示しています。

```
     // Connect to the server.
Connection c = JdbcMeDriver.getConnection
     ("jdbc:as400://mySystem;
       meserver=myMeServer;
       user=auser;
       password=apassword");

     // Create a Statement object.  Set the result set
     // concurrency to updatable.
Statement s = c.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                ResultSet.CONCUR_UPDATABLE);

     // Run a query. The result is placed
     // in a ResultSet object.
ResultSet rs = s.executeQuery ("SELECT NAME,ID FROM MYLIBRARY.MYTABLE FOR UPDATE");

     // Iterate through the rows of the ResultSet. As we read
     // the row, we will update it with a new ID.
int newId = 0;
while (rs.next ())
{

         // Get the values from the ResultSet. The first value
         // is a string, and the second value is an integer.
    String name = rs.getString("NAME");
    int id = rs.getInt("ID");
```

```
        System.out.println("Name = " + name);
        System.out.println("Old id = " + id);

             // Update the id with a new integer.
        rs.updateInt("ID", ++newId);

             // Send the updates to the server.
        rs.updateRow ();

        System.out.println("New id = " + newId);
    }

         // Close the Statement and the Connection.
    s.close();
    c.close();
```

JdbcMeOfflineResultSet クラス

JdbcMeOfflineResultSet クラスは、 Toolbox for Java の AS400JDBCResultSet クラスで使用可能な機能のサブセットを提供します。 Tier0 クライアント・アプリケーションで JdbcMeOfflineResultSet を使用して、 照会の実行によって生成されたデータ・テーブルにアクセスします。

JdbcMeOfflineResultSet クラスを使用して、Tier0 デバイスにあるデータを処理します。 デバイスに常駐するデータは、すでにあるか、 JdbcMeStatement.executeToOfflineData() メソッドを呼び出しデバイスに置くことができます。 executeToOfflineData() メソッドは、 照会に応えるすべてのデータをダウンロードしてデバイスに保管します。 その後、JdbcMeOfflineResultSet クラスを使用して保管されたデータにアクセスできます。

JdbcMeOfflineResultSet には、以下のアクションの実行を可能にするメソッドが含まれます。

- ResultSet に保管されるさまざまなタイプのデータを検索する
- 指定した行にカーソルを移動する (直前の行、現在行、次の行など)
- 行を挿入、 更新、 および削除する
- 列を更新する (String および int 値を使う)
- ResultSet の列を記述する ResultSetMetaData オブジェクトを検索する

JdbcMe クラスの機能を使用して、 ローカル・デバイス・データベースと iSeries サーバー上のデータベースを同期する能力を提供できます。

JdbcMeResultSetMetaData クラス

JdbcMeResultSetMetaData クラスは、 Toolbox for Java の AS400JDBCResultSetMetaData クラスで使用可能な機能のサブセットを提供します。 Tier0 クライアント・アプリケーションで JdbcMeResultSetMetaData を使用して、 JDBCResultSet の列のタイプおよびプロパティーを判別します。

以下の例は、JdbcMeResultSetMetaData クラスを使用する方法を示します。

```
        // Connect to the server.
    Connection c = JdbcMeDriver.getConnection
        ("jdbc:as400://mySystem;
          meserver=myMeServer;
```

```
        user=auser;
        password=apassword");

     // Create a Statement object.
Statement s = c.createStatement();

     // Run a query. The result is placed in a ResultSet object.
JdbcMeLiveResultSet rs = s.executeQuery ("SELECT NAME,ID FROM MYLIBRARY.MYTABLE");

     // Iterate through the rows of the ResultSet.
while (rs.next ())
{

         // Get the values from the ResultSet. The first value is
         // a string, and the second value is an integer.
    String name = rs.getString("NAME");
    int id = rs.getInt("ID");

    System.out.println("Name = " + name);
    System.out.println("ID = " + id);
}

     // Close the Statement and the Connection.
s.close();
c.close();
```

»JdbcMeOfflineData クラス

JdbcMeOfflineData クラスは、 Tier0 デバイスで使用されることを意図したオフラインのデータ・リポジトリーです。 リポジトリーは、使用するプロファイルおよび Java 仮想マシンにかかわらず、汎用です。 詳細については、ToolboxME for iSeries の概念を参照してください。

注: ToolboxMe for iSeries クラスを使うには、 ToolboxME for iSeries コンポーネントを別個にダウンロードしてセットアップする必要があります。 詳細は、ToolboxME for iSeries をダウンロードしてセットアップするを参照してください。

JdbcMeOfflineData クラスは、以下の機能を実行できるようにするメソッドを提供します。

- オフラインのデータ・リポジトリーを作成する
- 既存のリポジトリーをオープンする
- リポジトリーの レコード数を取得する
- 個々のレコードを取得および削除する
- レコードを更新する (Robb: the set() method, right?)
- リポジトリーの最後にレコードを追加する
- リポジトリーをクローズする

JdbcMeOfflineData クラスの使用例については、 ToolboxMe for iSeries の例 ToolboxME for iSeries、MIDP、および IBM Toolbox for Java を使用するを参照してください。 «

»JdbcMeStatement クラス

JdbcMeStatement クラスは、 Toolbox for Java の AS400JDBCStatement クラスで使用可能な機能のサブセットを提供します。 Tier0 クライアント・アプリケーションで JdbcMeStatement を使用し、パラメーターを全く持たない簡単な SQL ステートメントを実行し、 そのステートメントが生成する ResultSets を取得します。

注: ToolboxMe for iSeries クラスを使うには、 ToolboxME for iSeries コンポーネントを別個にダウンロードしてセットアップする必要があります。 詳細は、ToolboxME for iSeries をダウンロードしてセットアップするを参照してください。

Statement クラス

JdbcMeConnection.createStatement() を使用して、新規の Statement オブジェクトを作成します。

以下の例は、JdbcMeStatement オブジェクトを使用する方法を示します。

```
     // Connect to the server.
JdbcMeConnection c = JdbcMeDriver.getConnection(
     "jdbc:as400://mysystem.helloworld.com/mylibrary;
      naming=system;
      errors=full;
      meserver=myMeServer;
      user=auser;
      password=apassword");

     // Create a Statement object.
JdbcMeStatement s = c.createStatement();

     // Run an SQL statement that creates a table in the database.
s.executeUpdate("CREATE TABLE MYLIBRARY.MYTABLE (NAME VARCHAR(20), ID INTEGER)");

     // Run an SQL statement that inserts a record into the table.
s.executeUpdate("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES ('DAVE', 123)");

     // Run an SQL statement that inserts a record into the table.
s.executeUpdate("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES ('CINDY', 456)");

     // Run an SQL query on the table.
JdbcMeLiveResultSet rs = s.executeQuery("SELECT * FROM MYLIBRARY.MYTABLE");

     // Close the Statement and the Connection.
s.close();
c.close();
```

«

» ToolboxME for iSeries プログラムを作成して実行する

この情報により、ToolboxME for iSeries プログラムの例の編集、コンパイル、および実行が可能になります。 さらに、ToolboxME for iSeries の作業例および独自の ToolboxME for iSeries アプリケーションを作成、テスト、実行するための 一般的な指針としてこの情報を使用することもできます。

このプログラム例は K Virtual Machine (KVM) を使用し、 ユーザーが任意の JDBC 照会を実行できるようにします。 それで、ユーザーは照会の結果に対して JDBC アクション (次へ、前へ、閉じる、コミット、 およびロールバック) を実行できます。

ToolboxME for iSeries の例の作成を開始する前に、 ご使用の環境が ToolboxME for iSeries の要件を満たしていることを確認してください。

ToolboxME for iSeries の例を作成する

Tier0 デバイス用に ToolboxME for iSeries のプログラム例を作成するには、 以下の手順に従ってください。

1. JdbcDemo.java と呼ばれる、 ToolboxME for iSeries の例の Java コードをコピーする。
2. 選択したテキスト・エディターまたは Java エディターで、 コードの部分をプログラム・コメントで指示されているように変更し、 JdbcDemo.java という名前でファイルを保管する。

   注: 残りの手順の実行がより容易になる、 ワイヤレス・アプリケーション開発ツールの使用を考慮してください。 ワイヤレス・アプリケーション開発ツールの中には単一ステップでコンパイルし、事前検査して、 プログラムを構築し、その後エミュレーターで自動的に実行するものもあります。
3. JdbcDemo.java をコンパイルし、KVM クラスを含む .jar ファイルを指定していることを確認する。
4. ワイヤレス・アプリケーション開発ツールまたは Java 事前検査コマンドのどちらかを使用して、 実行可能ファイルを事前検査する。
5. Tier0 デバイスのオペレーティング・システム用に適切なタイプの実行可能ファイルを構築する。 たとえば Palm OS の場合、JdbcDemo.prc という名前のファイルを構築します。
6. プログラムをテストする。 エミュレーターがインストール済みの場合、エミュレーターでプログラムを実行することにより、 プログラムをテス

トし、どのような感じなのかを把握できます。

注: ワイヤレス・デバイス上でプログラムをテストし、 ワイヤレス・アプリケーション開発ツールを使用しない場合、 選択した Java 仮想マシンまたはデバイスの MIDP をプリロードしたことを確認してください。

概念、ワイヤレス・アプリケーション開発ツール、およびエミュレーターに関する情報は、 ToolboxME for iSeries の概念を参照してください。

ToolboxME for iSeries の例を実行する

Tier0 デバイスで ToolboxME for iSeries のプログラム例を実行するには、 以下のタスクを実行してください。

- Tier0 デバイスのメーカーによって備えられている説明を使用して、 実行可能ファイルをデバイスにロードする。
- MEServer を開始する。
- 「JdbcDemo」アイコンをクリックして、Tier0 デバイスで JdbcDemo プログラムを実行する。 «

ToolboxME for iSeries の作業例

以下の ToolboxMe for iSeries の作業例は、 Mobile Information Device Profile (MIDP)によって ToolboxME for iSeries を使用する方法を例示します。 以下のリンクを使用し、選択されたソース・ファイル例を表示するか、 またはワイヤレス・アプリケーションの作業例を作成するのに必要なソース・ファイルすべてをダウンロードします。

例: ToolboxME for iSeries、MIDP、および JDBC

例: ToolboxME for iSeries、MIDP、および IBM Toolbox for Java

ToolboxME for iSeries の作業例をダウンロードする

ToolboxME for iSeries アプリケーションを構築する方法についての詳細は、ToolboxME for iSeries プログラムを作成して実行するを参照してください。

よく尋ねられる質問 (FAQ)

IBM Toolbox for Java FAQ (よく尋ねられる質問) では、 IBM Toolbox for Java のパフォーマンスの最適化、トラブルシューティングの実行、 JDBC の使用などに関する質問への回答を提供します。

- IBM Toolbox for Java FAQ: パフォーマンスの向上、 OS/400 の使用、トラブルシューティングの実行などを含む、各種の質問への回答が記載されています。
- IBM Toolbox for Java JDBC FAQ: Toolbox for Java による JDBC の使用に関する質問への回答が記載されています。

コード例

以下のリストは、IBM Toolbox for Java トピックで使用される多くの例のエントリー・ポイントへのリンクを提供します。

アクセス・クラス	Beans	グラフィカル・ツールボックス
HTML クラス	PCML	ReportWriter クラス
リソース・クラス	RFML	セキュリティー・クラス
サーブレット・クラス	簡単な例	プログラミングに関するヒント
ToolboxMe for iSeries	ユーティリティー・クラス	Vaccess クラス

以下の特記事項は、IBM Toolbox for Java のすべての例に適用されます。

コードの特記事項情報

IBM は、お客様に、すべてのプログラミング・コード・サンプルを使用することができる非独占的な使用権を許諾します。お客様は、このサンプル・コードから、お客様独自の特別のニーズに合わせた類似のプログラムを作成することができます。

すべてのサンプル・コードは、例として示す目的でのみ、IBM により提供されます。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。したがって 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。

ここに含まれるすべてのプログラムは、現存するままの状態で提供され、いかなる保証条件も適用されません。第三者の権利の不侵害、商品性、特定目的適合性に関する黙示の保証の適用も一切ありません。

例: アクセス・クラス

このセクションでは、アクセス・クラスの資料に記載されているコーディング例をリストします。

AS400JPing

- 例: Java プログラム内で AS400JPing を使用する

BidiTransform

- 例: AS400BidiTransform クラスを使用して両方向テキストを変換する

CommandCall

- 例: CommandCall を使用してサーバー上のコマンドを実行する
- 例: CommandCall を使用して、サーバーの名前および実行するコマンド名を求めるプロンプトを出し、結果を印刷する

ConnectionPool

- 例: AS400ConnectionPool を使用してサーバーへの接続を作成する

DataArea

- 例: 10 進数データ域を作成し、書き込みを行う

データ変換および記述

- 例: FieldDescription、RecordFormat、および Record クラスを使用する
- 例: 待ち行列にデータを書き込む
- 例: 待ち行列からデータを読み取る

DataQueue

- 例: DataQueue オブジェクトを作成し、データを読み取った後、切断する方法
- 例: 待ち行列にデータを書き込む
- 例: 待ち行列からデータを読み取る

ディジタル証明書

- 例: ユーザーが持っているディジタル証明書をリストする

EnvironmentVariable

- 例: 環境変数の作成、設定、および取得

例外

- 例: 出された例外を受け取り、戻りコードを取得して、例外テキストを表示する

FTP

- 例: FTP クラスを使用してサーバーのディレクトリーから一連のファイルをコピーする
- 例: AS400FTP クラスを使用してディレクトリーから一連のファイルをコピーする

統合ファイル・システム

- 例: IFSFile を使用する
- 例: IFSFile.listFiles() メソッドを使用してディレクトリーの内容をリストする
- 例: IFSFile クラスを使用してファイルをコピーする
- 例: IFSFile クラスを使用してディレクトリーの内容をリストする
- 例: java.io.File の代わりに IFSJavaFile を使用する方法
- 例: IFSFile クラスを使用してサーバー上のディレクトリーの内容をリストする

JavaApplicationCall

- 例: クライアントからサーバー上で "Hello World!" を出力するプログラムを実行する

JDBC

- 例: JDBC ドライバーを使用して、表の作成と記入を行う
- 例: JDBC ドライバーを使用して、表の照会およびその内容の出力を行う

ジョブ

- 例: キャッシュを使用してジョブ情報検索および変更を行う
- 例: 活動状態のジョブをすべてリストする
- 例: 指定したユーザー別にジョブ・ログ内のすべてのメッセージを印刷する
- 例: 指定したユーザー別にジョブ識別情報をリストする
- 例: サーバー上のジョブのリストを取得し、ジョブ状況およびジョブ識別子をリストする
- 例: 現行ユーザーに属するジョブの、ジョブ・ログ内のメッセージを表示する

メッセージ待ち行列

- 例: メッセージ待ち行列オブジェクトを使用する方法
- 例: メッセージ待ち行列の内容を印刷する
- 例: メッセージを検索して印刷する
- 例: メッセージ待ち行列の内容をリストする
- 例: CommandCall と共に AS400Message を使用する
- 例: ProgramCall と共に AS400Message を使用する

NetServer

- 例: NetServer オブジェクトを使用して NetServer の名前を変更する

印刷

- 例: 入力ストリームからスプール・ファイルを作成する
- 例: SCS3812Writer クラスを使用して SCS データ・ストリームを生成する
- 例: 既存のスプール・ファイルを読み取る
- 例: PrintObjectListListener インターフェースを使用してすべてのスプール・ファイルを非同期的にリストする

- 例: PrintObjectListListener インターフェースを使用せずにすべてのスプール・ファイルを非同期的にリストする
- 例: すべてのスプール・ファイルを同期的にリストする

許可

- 例: AS400 オブジェクトの権限を設定する

プログラム呼び出し

- 例: ProgramCall を使用する
- 例: ProgramCall を使用してシステム状況を取り出す
- 例: プログラム・パラメーター・オブジェクトを使ってパラメーター・データを渡す

QSYSObjectPathName

- 例: 統合ファイル・システム名を作成する
- 例: QSYSObjectPathName.toPath() を使用して AS400 オブジェクトを作成する
- 例: QSYSObjectPathName を使用して統合ファイル・システム・パス名を解析する

レコード・レベルでのアクセス

- 例: ファイルに順次にアクセスする
- 例: レコード・レベルでアクセス・クラスを使用してファイルを読み取る
- 例: レコード・レベルでアクセス・クラスを使用してキーによってレコードを読み取る
- 例: LineDataRecordWriter クラスを使用する

サービス・プログラム呼び出し

- 例: ServiceProgramCall を使用してプロシージャーを呼び出す

システム状況

- 例: SystemStatus クラスでキャッシュを使用する

システム・プール

- 例: SystemPool の最大障害サイズを設定する

SystemValue

- 例: SystemValue および SystemValueList を使用する

トレース

- 例: Trace.setTraceOn() メソッドを使用する
- 例: トレースの使用に関する望ましい方法
- 例: コンポーネント・トレースを使用する

ユーザー・グループ

- 例: ユーザーのリストを取得する
- 例: グループに含まれる全ユーザーをリストする

ユーザー・スペース

- 例: ユーザー・スペースを作成する方法

以下の特記事項は、IBM Toolbox for Java のすべての例に適用されます。

ここに含まれるすべてのプログラムは、現存するままの状態で提供され、いかなる保証条件も適用されません。第三者の権利の不侵害、商品性、特定目的適合性に関する黙示の保証の適用も一切ありません。

例: CommandCall を使用する

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// Command call example.  This program prompts the user
// for the name of the server and the command to run, then
// prints the result of the command.
//
// This source is an example of IBM Toolbox for Java "CommandCall"
//
///////////////////////////////////////////////////////////////////////////



import java.io.*;
import java.util.*;
import com.ibm.as400.access.*;

public class CommandCallExample extends Object
{
   public static void main(String[] parmeters)
   {

      // Created a reader to get input from the user
      BufferedReader inputStream = new BufferedReader(new InputStreamReader(System.in),1);



      // Declare variables to hold the system name and the command to run
      String systemString  = null;
      String commandString = null;

      System.out.println( " " );



      // Get the system name and the command to run from the user
      try
      {
         System.out.print("System name: ");
         systemString = inputStream.readLine();

         System.out.print("Command: ");
         commandString = inputStream.readLine();
      }
      catch (Exception e) {};

      System.out.println( " " );
```

```
        // Create an AS400 object.  This is the system we send the command to
        AS400 as400 = new AS400(systemString);



        // Create a command call object specifying the server that will
        // recieve the command.
        CommandCall command = new CommandCall( as400 );



        try
        {
           // Run the command.
           if (command.run(commandString))
               System.out.print( "Command successful" );
           else
               System.out.print( "Command failed" );



           // If messages were produced from the command, print them
           AS400Message[] messagelist = command.getMessageList();

           if (messagelist.length > 0)
           {
               System.out.println( ", messages from the command:" );
               System.out.println( " " );
           }

           for (int i=0; i < messagelist.length; i++)
           {
              System.out.print  ( messagelist[i].getID() );
              System.out.print  ( ": " );
              System.out.println( messagelist[i].getText() );
           }
        }
        catch (Exception e)
        {
           System.out.println( "Command " + command.getCommand() + " did not run" );
        }

        System.exit(0);
     }
}
```

例: AS400ConnectionPool を使用する

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// AS400ConnectionPooling example.  This program uses an AS400ConnectionPool to
// create connections to an iSeries system.
// Command syntax:
//    AS400ConnectionPooling system myUserId myPassword
//
// For example,
//    AS400ConnectionPooling MySystem MyUserId MyPassword
//
///////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;

public class AS400ConnectionPooling
{
   public static void main (String[] parameters)
   {
      // Check the input parameters.
      if (parameters.length != 3)
      {
         System.out.println("");
         System.out.println("Usage:");
         System.out.println("");
         System.out.println("   AS400ConnectionPooling system userId password");
         System.out.println("");
         System.out.println("");
         System.out.println("For example:");
         System.out.println("");
         System.out.println("");
         System.out.println("   AS400ConnectionPooling MySystem MyUserId MyPassword");
         System.out.println("");
         return;
      }

      String system   = parameters[0];
      String userId   = parameters[1];
      String password = parameters[2];

      try
      {
         // Create an AS400ConnectionPool.
         AS400ConnectionPool testPool = new AS400ConnectionPool();

         // Set a maximum of 128 connections to this pool.
         testPool.setMaxConnections(128);

         // Set a maximum lifetime for 30 minutes for connections.
         testPool.setMaxLifetime(1000*60*30);     // 30 min  Max lifetime since created.

         // Preconnect 5 connections to the AS400.COMMAND service.
         testPool.fill(system, userId, password, AS400.COMMAND, 1);
         System.out.println();
         System.out.println("Preconnected 1 connection to the AS400.COMMAND service");
```

```
        // Call getActiveConnectionCount and getAvailableConnectionCount to see how many
        // connections are in use and available for a particular system.
        System.out.println("Number of active connections: " +
testPool.getActiveConnectionCount(system, userId));
        System.out.println("Number of available connections for use:  " +
testPool.getAvailableConnectionCount(system, userId));

        // Create a connection to the AS400.COMMAND service. (Use the service number constants
        // defined in the AS400 class (FILE, PRINT, COMMAND, DATAQUEUE, etc.))
        // Since connections have already been filled, the usual time spent connecting to the command
        // service is avoided.
        AS400 newConn1 = testPool.getConnection(system, userId, password, AS400.COMMAND);

        System.out.println();
        System.out.println("getConnection gives out an existing connection to user");
        System.out.println("Number of active connections: " +
testPool.getActiveConnectionCount(system, userId));
        System.out.println("Number of available connections for use:  " +
testPool.getAvailableConnectionCount(system, userId));

        // Create a new command call object and run a command.
        CommandCall cmd1 = new CommandCall(newConn1);
        cmd1.run("CRTLIB FRED");

        // Return the connection to the pool.
        testPool.returnConnectionToPool(newConn1);

        System.out.println();
        System.out.println("Returned a connection to pool");
        System.out.println("Number of active connections: " +
testPool.getActiveConnectionCount(system, userId));
        System.out.println("Number of available connections for reuse: " +
testPool.getAvailableConnectionCount(system, userId));

        // Create a connection to the AS400.COMMAND service.  This will return the same
        // object as above for reuse.
        AS400 newConn2 = testPool.getConnection(system, userId, password, AS400.COMMAND);

        System.out.println();
        System.out.println("getConnection gives out an existing connection to user");
        System.out.println("Number of active connections: " +
testPool.getActiveConnectionCount(system, userId));
        System.out.println("Number of available connections for reuse: " +
testPool.getAvailableConnectionCount(system, userId));

        // Create a connection to the AS400.COMMAND service.  This will create a new
        // connection as there are not any connections in the pool to reuse.
        AS400 newConn3 = testPool.getConnection(system, userId, password, AS400.COMMAND);

        System.out.println();
        System.out.println("getConnection creates a new connection since there are no connections
available");
        System.out.println("Number of active connections: " +
testPool.getActiveConnectionCount(system, userId));
        System.out.println("Number of available connections for reuse: " +
testPool.getAvailableConnectionCount(system, userId));

        // Close the test pool.
        testPool.close();
    }
```

```
        catch (Exception e)
        {
           // If any of the above operations failed say the pool operations failed
           // and output the exception.

           System.out.println("Pool operations failed");
           System.out.println(e);
           e.printStackTrace();
        }
     }
}
```

例: DataQueue クラスを使用して待ち行列にデータを書き込む

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// Data Queue example.  This program uses the DataQueue class to put
// records on a data queue.
//
// This example uses the Record and Record format classes to put data
// on the queue.  String data is converted from Unicode to ebcdic
// and numbers are converted from Java to the server format.  Since data
// is converted the data queue entries can be read by a server program
// or a iSeries Access for Windows program as well as another Java program.
//
// This is the producer side of the producer/consumer example.  It puts work
// items on the queue for the consumer to process.
//
// Command syntax:
//    DQProducerExample system
//
///////////////////////////////////////////////////////////////////////////


import java.io.*;
import java.util.*;
import java.net.*;
import com.ibm.as400.access.*;

public class DQProducerExample extends Object
{
   // Create a reader to get input from the user.
   static BufferedReader inputStream = new BufferedReader(new InputStreamReader(System.in),1);

   public static void main(String[] parameters)
   {
      System.out.println( " " );

      // if the system name was not specified, display help text and exit.
      if (parameters.length >= 1)
      {
         try
         {
             // The first parameter is the system that contains the data queue.
             String system = parameters[0];

             // Create an AS400 object for the server that has the data queue.
             AS400 as400 = new AS400(system);

             // Build a record format for the format of the data queue entry.
             // This format matches the format in the DQConsumer class.  A
             // record consists of:
             //    - a four byte number    -- the customer number
             //    - a four byte number    -- the part number
             //    - a 20 character string -- the part description
             //    - a four byte number    -- the number of parts in this order
```

```
// First create the base data types.
BinaryFieldDescription customerNumber =
   new BinaryFieldDescription(new AS400Bin4(), "CUSTOMER_NUMBER");

BinaryFieldDescription partNumber =
   new BinaryFieldDescription(new AS400Bin4(), "PART_NUMBER");

CharacterFieldDescription partName =
   new CharacterFieldDescription(new AS400Text(20, as400), "PART_NAME");

BinaryFieldDescription quantity =
   new BinaryFieldDescription(new AS400Bin4(), "QUANTITY");

// Build a record format and fill it with the base data types.
RecordFormat dataFormat = new RecordFormat();
dataFormat.addFieldDescription(customerNumber);
dataFormat.addFieldDescription(partNumber);
dataFormat.addFieldDescription(partName);
dataFormat.addFieldDescription(quantity);

// Create the library that contains the data queue using CommandCall.
CommandCall crtlib = new CommandCall(as400);
crtlib.run("CRTLIB JAVADEMO");

// Create the data queue object.
DataQueue dq = new DataQueue(as400, "/QSYS.LIB/JAVADEMO.LIB/PRODCONS.DTAQ");

// Create the data queue just in case this is the first time this
// program has run.  The queue already exists exception is caught
// and ignored.
try
{
   dq.create(96);
}
catch (Exception e) {};

// Get the first field of data from the user.
System.out.print("Enter customer number (or 0 to quit): ");
int customer = getInt();

// While there is data to put on the queue.
while (customer > 0)
{
   // Get the rest of the data for this order from the user.
   System.out.print("Enter part number: ");
   int part = getInt();

   System.out.print("Enter quantity: ");
   int quantityToOrder = getInt();

   String description = "part " + part;

   // Create a record based on the record format.  The record
   // is empty now but will eventually contain the data.
   Record data = new Record(dataFormat);
```

```
            // Set the values we received from the user into the record.
            data.setField("CUSTOMER_NUMBER",  new Integer(customer));
            data.setField("PART_NUMBER",      new Integer(part));
            data.setField("QUANTITY",         new Integer(quantityToOrder));
            data.setField("PART_NAME", description);

            // Convert the record into a byte array.  The byte array is what
            // is actually put to the data queue.
            byte [] byteData = data.getContents();

            System.out.println("");
            System.out.println("Writing record to the server ...");
            System.out.println("");

            // Write the record to the data queue.
            dq.write(byteData);

            // Get the next value from the user.
            System.out.print("Enter customer number (or 0 to quit): ");
            customer = getInt();
         }
      }
      catch (Exception e)
      {
         // If any of the above operations failed say the data queue
         // operation failed and output the exception.

         System.out.println("Data Queue operation failed");
         System.out.println(e);
      }
   }

   // Display help text when parameters are incorrect.
   else
   {
      System.out.println("");
      System.out.println("");
      System.out.println("");
      System.out.println("Parameters are not correct. Command syntax is:");
      System.out.println("");
      System.out.println(" DQProducter system");
      System.out.println("");
      System.out.println("Where");
      System.out.println("");
      System.out.println(" system = Server that has the data queue");
      System.out.println("");
      System.out.println("For example:");
      System.out.println("");
      System.out.println(" DQProducerExample mySystem");
      System.out.println("");
      System.out.println("");
   }

   System.exit(0);
```

```
    }

    // This is the subroutine that gets a character string from the user
    // and converts it into an int.
    static int getInt()
    {
       int i = 0;
       boolean Continue = true;

       while (Continue)
       {
          try
          {
             String s = inputStream.readLine();

             i = (new Integer(s)).intValue();
             Continue = false;
          }
          catch (Exception e)
          {
             System.out.println(e);
             System.out.print("Please enter a number ==>");
          }
       }

       return i;
    }
}
```

例: DataQueue クラスを使用してデータ待ち行列から項目を読み取る

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// Data Queue example.  This program uses the Data Queue classes to read
// entries off a data queue on the server.  The entries were put on the
// queue with the DQProducer example program.
//
// This is the consumer side of the producer/consumer example.  It reads
// entries off the queue and process them.
//
// Command syntax:
//    DQConsumerExample system
//
///////////////////////////////////////////////////////////////////////////


import java.io.*;
import java.util.*;
import java.net.*;
import com.ibm.as400.access.*;

public class DQConsumerExample extends Object
{
   public static void main(String[] parameters)
   {
      System.out.println( " " );

      // if a system name was not specified, display help text and exit.
      if (parameters.length >= 1)
      {
         try
         {

             // The first parameter is the system that contains the data queue.
             String system = parameters[0];

             // Create an AS400 object for the server that has the data queue.
             AS400 as400 = new AS400(system);


             // Build a record format for the format of the data queue entry.
             // This format matches the format in the DQProducer class.  A
             // record consists of:
             //    - a four byte number -- the customer number
             //    - a four byte number -- the part number
             //    - a 20 character string -- the part description
             //    - a four byte number -- the number of parts in this order

             // First create the base data types.
             BinaryFieldDescription customerNumber =
```

```
        new BinaryFieldDescription(new AS400Bin4(), "CUSTOMER_NUMBER");

    BinaryFieldDescription partNumber =
        new BinaryFieldDescription(new AS400Bin4(), "PART_NUMBER");

    CharacterFieldDescription partName =
        new CharacterFieldDescription(new AS400Text(20, as400), "PART_NAME"

    BinaryFieldDescription quantity =
        new BinaryFieldDescription(new AS400Bin4(), "QUANTITY"

    // Build a record format and fill it with the base data types.
    RecordFormat dataFormat = new RecordFormat();

    dataFormat.addFieldDescription(customerNumber);
    dataFormat.addFieldDescription(partNumber);
    dataFormat.addFieldDescription(partName);
    dataFormat.addFieldDescription(quantity);

    // Create the data queue object that represents the data queue on
    // the server.
    DataQueue dq = new DataQueue(as400, "/QSYS.LIB/JAVADEMO.LIB/PRODCONS.DTAQ");

    boolean Continue = true;

    // Read the first entry off the queue.  The timeout value is
    // set to -1 so this program will wait forever for an entry.
    System.out.println("*** Waiting for an entry for process ***");

    DataQueueEntry DQData = dq.read(-1);

    while (Continue)
    {

        // We just read an entry off the queue.  Put the data into
        // a record object so the program can access the fields of
        // the data by name.  The Record object will also convert
        // the data from server format to Java format.
        Record data = dataFormat.getNewRecord(DQData.getData());

        // Get two values out of the record and display them.
        Integer amountOrdered = (Integer) data.getField("QUANTITY");
        String  partOrdered   = (String)  data.getField("PART_NAME");

        System.out.println("Need " + amountOrdered + " of " + partOrdered);
        System.out.println(" ");
        System.out.println("*** Waiting for an entry for process ***");

        // Wait for the next entry.
        DQData = dq.read(-1);
    }
}
```

```
            catch (Exception e)
            {
                // If any of the above operations failed say the data queue operation
                // failed and output the exception.
                System.out.println("Data Queue operation failed");
                System.out.println(e);
            }
        }

        // Display help text when parameters are incorrect.
        else
        {
            System.out.println("");
            System.out.println("");
            System.out.println("");
            System.out.println("Parameters are not correct. Command syntax is:");
            System.out.println("");
            System.out.println(" DQConsumerExample system");
            System.out.println("");
            System.out.println("Where");
            System.out.println("");
            System.out.println(" system = Server that has the data queue");
            System.out.println("");
            System.out.println("For example:");
            System.out.println("");
            System.out.println(" DQConsumerExample mySystem");
            System.out.println("");
            System.out.println("");
        }

        System.exit(0);
    }
}
```

例: IFSFile を使用する

以下の例では、IFSFile クラスを使用する方法を示します。

- 例: ディレクトリーを作成する
- 例: IFSFile 例外を使用してエラーを追跡する
- 例: .txt の拡張子が付けられたファイルをリストする
- 例: IFSFile listFiles() メソッドを使用して、 ディレクトリーの内容をリストする

例: ディレクトリーを作成する

```
                    // Create an AS400 object.  This new
                    // directory will be created on this
                    // iSeries.
AS400 sys = new AS400("mySystem.myCompany.com");

                    // Create a file object that
                    // represents the directory.
IFSFile aDirectory = new IFSFile(sys, "/mydir1/mydir2/newdir");

                    // Create the directory.
if (aDirectory.mkdir())
   System.out.println("Create directory was successful");

                    // Else the create directory failed.
else
{
                    // If the object already exists,
                    // find out if it is a directory or
                    // file, then display a message.
   if (aDirectory.exists())
   {
      if (aDirectory.isDirectory())
         System.out.println("Directory already exists");
      else
         System.out.println("File with this name already exists");
   }
   else
      System.out.println("Create directory failed");
```

```
}

                    // Disconnect since I am done
                    // accessing files.
sys.disconnectService(AS400.FILE);
```

例: IFSFile 例外を使用してエラーを追跡する

エラーが生じると、 IFSFile クラスは ExtendedIOException 例外を出します。 この例外には、障害の原因を説明した戻りコードが示されています。 IFSFile と重複する java.io クラスが例外を送出しない場合でも、 この IFSFile クラスは例外を送出します。 たとえば、 java.io.File からの削除メソッドは、 成功または失敗を示すブール値を戻します。 IFSFile での削除メソッドはブール値を戻しますが、 削除が失敗した場合は ExtendedIOException が送出されます。 この ExtendedIOException では、 削除が失敗した理由の詳細を Java プログラムに知らせます。

```
                    // Create an AS400 object.
AS400 sys = new AS400("mySystem.myCompany.com");

                    // Create a file object that
                    // represents the file.
IFSFile aFile = new IFSFile(sys, "/mydir1/mydir2/myfile");

                    // Delete the file.
try
{
   aFile.delete();

                    // The delete was successful.
   System.out.println("Delete successful ");
}

                    // The delete failed. Get the return
                    // code out of the exception and
                    // display why the delete failed.
catch (ExtendedIOException e)
{
   int rc = e.getReturnCode();

   switch (rc)
   {
```

```
        case ExtendedIOException.FILE_IN_USE:
           System.out.println("Delete failed, file is in use ");
           break;

        case ExtendedIOException.PATH_NOT_FOUND:
           System.out.println("Delete failed, path not found ");
           break;

                     // ... for every specific error
                     // you want to track.

        default:
           System.out.println("Delete failed, rc = ");
           System.out.println(rc);
     }
  }
```

例: .txt の拡張子が付けられたファイルをリストする

例 3: Java プログラムは、 ディレクトリー内のファイルをリストするときに、 任意に突き合わせの基準を指定することができます。 突き合わせの基準を指定すると、 サーバー によって IFSFile オブジェクトへ戻されるファイルの数が少なくなるので、 パフォーマンスが向上します。 以下の例は .txt の拡張子が付けられたファイルをリストする方法を示しています。

```
                    // Create the AS400 object.
AS400 system = new AS400("mySystem.myCompany.com");

                    // Create the file object.
IFSFile directory = new IFSFile(system, "/");

                    // Generate a list of all files with
                    // extension .txt
String[] names = directory.list("*.txt");

                    // Display the names.
if (names != null)
  for (int i = 0; i < names.length; i++)
    System.out.println(names[i]);
else
  System.out.println("No .txt files");
```

例: IFSFile listFiles() メソッドを使用して、ディレクトリーの内容をリストする

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
////////////////////////////////////////////////////////////////////////
//
// IFSListFiles example.  This program uses the integrated file system
// classes to list the contents of a directory on the server.
//
// Command syntax:
//    IFSListFiles system directory
//
// For example,
//    IFSListFiles MySystem /path1
//
////////////////////////////////////////////////////////////////////////

import java.io.*;
import java.util.*;
import com.ibm.as400.access.*;

public class IFSListFiles extends Object
{
    public static void main(String[] parameters)
    {
        System.out.println( " " );

        String directoryName = "";
        String system        = "";


        // if both parameters were not specified, display help text and exit.

        if (parameters.length >= 2)
        {

            // Assume the first parameter is the system name and
            // the second parameter is the directory name

            system = parameters[0];
            directoryName = parameters[1];

            try
            {
                // Create an AS400 object for the server that holds the files.

                AS400 as400 = new AS400(system);



                // Create the IFSFile object for the directory.

                IFSFile directory = new IFSFile(as400, directoryName);
```

```
// Generate a list of IFSFiles.  Pass the listFiles method
// the directory filter object and the search match
// criteria.  This method caches attribute information.  For
// instance, when isDirectory() is called on an IFSFile
// object in the file array returned in the following code,
// no call to the server is required.
//
// However, with the user of the listFiles method, attribute
// information will not be refreshed automatically from the
// server. This means attribute information can become
// inconsistent with information on the server.

IFSFile[] directoryFiles = directory.listFiles(new MyDirectoryFilter(),"*");

// Tell the user if the directory doesn't exist or is empty

if (directoryFiles == null)
{
    System.out.println("The directory does not exist");
    return;
}

else if (directoryFiles.length == 0)
{
    System.out.println("The directory is empty");
    return;
}

for (int i=0; i< directoryFiles.length; i++)
    {
        // Print out information on list.
        // Print the name of the current file

        System.out.print(directoryFiles[i].getName());


        // Pad the output so the columns line up

        for (int j = directoryFiles[i].getName().length(); j <18; j++)
            System.out.print(" ");


        // Print the date the file was last changed.

        long changeDate = directoryFiles[i].lastModified();
        Date d = new Date(changeDate);
        System.out.print(d);
        System.out.print("  ");


        // Print if the entry is a file or directory
```

```
                    System.out.print("   ");

                    if (directoryFiles[i].isDirectory())
                        System.out.println("");
                    else
                        System.out.println(directoryFiles[i].length());

                }
            }

            catch (Exception e)
            {
                // If any of the above operations failed say the list failed
                // and output the exception.

                System.out.println("List failed");
                System.out.println(e);
            }
        }

        // Display help text when parameters are incorrect.

        else
        {
            System.out.println("");
            System.out.println("");
            System.out.println("");
            System.out.println("Parameters are not correct.  Command syntax is:");
            System.out.println("");
            System.out.println("   IFSListFiles as400 directory");
            System.out.println("");
            System.out.println("Where");
            System.out.println("");
            System.out.println("   as400 = system that contains the files");
            System.out.println("   directory = directory to be listed");
            System.out.println("");
            System.out.println("For example:");
            System.out.println("");
            System.out.println("   IFSListFiles mySystem /dir1/dir2");
            System.out.println("");
            System.out.println("");
        }

        System.exit(0);
    }
}


////////////////////////////////////////////////////////////////////////
//
// The directory filter class prints information from the file object.
//
// Another way to use the filter is to simply return true or false
// based on information in the file object.  This lets the mainline
```

```
// function decide what to do with the list of files that meet the
// search criteria.
//
//////////////////////////////////////////////////////////////////////////



class MyDirectoryFilter implements IFSFileFilter
{
    public boolean accept(IFSFile file)
    {
        try
        {
            // Keep this entry.  Returning true tells the IFSList object
            // to return this file in the list of entries returned to the
            // .list() method.

            return true;
        }

        catch (Exception e)
        {
            return false;
        }
    }
}
```

例: IFS クラスを使用してディレクトリーからディレクトリーにファイルをコピーする

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
////////////////////////////////////////////////////////////////////////////
//
// IFSCopyFile example.  This program uses the installable file system classes
// to copy a file from one directory to another on the server.
//
// Command syntax:
//    IFSCopyFile system sourceName TargetName
//
// For example,
//    IFSCopyFile MySystem  /path1/path2/file.ext  /path3/path4/path5/file.ext
//
////////////////////////////////////////////////////////////////////////////



import java.io.*;
import java.util.*;
import com.ibm.as400.access.*;

public class IFSCopyFile extends Object
{
   public static void main(String[] parameters)
   {
      System.out.println( " " );

      String sourceName = "";
      String targetName = "";
      String system     = "";
      byte[] buffer     = new byte[1024 * 64];

      IFSFileInputStream  source = null;
      IFSFileOutputStream target = null;

      // if all three parameters were not specified, display help text and exit.

      if (parameters.length > 2)
      {

         // Assume the first parameter is the system name,
         // the second parameter is the source name and
         // the third parameter is the target name.

         system     = parameters[0];
         sourceName = parameters[1];
         targetName = parameters[2];
```

```
try
{
   // Create an AS400 object for the server that holds the files.

   AS400 as400 = new AS400(system);



   // Open the source file for exclusive access.

   source = new IFSFileInputStream(as400,
                                   sourceName,
                                   IFSFileInputStream.SHARE_NONE);

   System.out.println("Source file successfully opened");



   // Open the target file for exclusive access.

   target = new IFSFileOutputStream(as400,
                                    targetName,
                                    IFSFileOutputStream.SHARE_NONE,
                                    false);

   System.out.println("Target file successfully opened");



   // Read the first 64K bytes from the source file.

   int bytesRead = source.read(buffer);



   // While there is data in the source file copy the data from
   // the source file to the target file.

   while (bytesRead > 0)
   {
      target.write(buffer, 0, bytesRead);
      bytesRead = source.read(buffer);
   }

   System.out.println("Data successfully copied");



   // Clean up by closing the source and target files.
```

```
      source.close();
      target.close();



      // Get the last changed date/time from the source file and
      // set it on the target file.

      IFSFile src = new IFSFile(as400, sourceName);
      long dateTime = src.lastModified();

      IFSFile tgt = new IFSFile(as400, targetName);
      tgt.setLastModified(dateTime);

      System.out.println("Date/Time successfully set on target file");
      System.out.println("Copy Successful");

   }
   catch (Exception e)
   {
      // If any of the above operations failed say the copy failed
      // and output the exception.

      System.out.println("Copy failed");
      System.out.println(e);
   }
}

// Display help text when parameters are incorrect.

else
{
   System.out.println("");
   System.out.println("");
   System.out.println("");
   System.out.println("Parameters are not correct. Command syntax is:");
   System.out.println("");
   System.out.println("   IFSCopyFile as400 source target");
   System.out.println("");
   System.out.println("Where");
   System.out.println("");
   System.out.println("   as400  = system that contains the files");
   System.out.println("   source = source file in /path/path/name format");
   System.out.println("   target = target file in /path/path/name format");
   System.out.println("");
   System.out.println("For example:");
   System.out.println("");
   System.out.println("   IFSCopyFile myAS400  /dir1/dir2/a.txt  /dir3/b.txt");
   System.out.println("");
```

```
            System.out.println("");
        }

        System.exit(0);
    }
}
```

例: IFS クラスを使用して、ディレクトリーの内容をリストする

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
////////////////////////////////////////////////////////////////////////////
//
// IFSListFile example.  This program uses the integrated file system classes
// to list the contents of a directory on the server.
//
// Command syntax:
//    IFSList system directory
//
// For example,
//    IFSList MySystem /path1
//
////////////////////////////////////////////////////////////////////////////

import java.io.*;
import java.util.*;
import com.ibm.as400.access.*;

public class IFSList extends Object
{
   public static void main(String[] parameters)
   {
      System.out.println( " " );

      String directoryName = "";
      String system        = "";


      // if both parameters were not specified, display help text and exit.

      if (parameters.length >= 2)
      {

         // Assume the first parameter is the system name and
         // the second parameter is the directory name

         system = parameters[0];
         directoryName = parameters[1];

         try
         {
            // Create an AS400 object for the server that holds the files.

            AS400 as400 = new AS400(system);
```

```
      // Create the IFSFile object for the directory.

      IFSFile directory = new IFSFile(as400, directoryName);



      // Generate the list of name.  Pass the list method the
      // directory filter object and the search match criteria.
      //
      // Note - this example does the processing in the filter
      // object.  An alternative is to process the list after
      // it is returned from the list method call.

      String[] directoryNames = directory.list(new MyDirectoryFilter(),"*");



      // Tell the user if the directory doesn't exist or is empty

      if (directoryNames == null)
         System.out.println("The directory does not exist");

      else if (directoryNames.length == 0)
         System.out.println("The directory is empty");
   }

   catch (Exception e)
   {
      // If any of the above operations failed say the list failed
      // and output the exception.

      System.out.println("List failed");
      System.out.println(e);
   }
}

// Display help text when parameters are incorrect.

else
{
   System.out.println("");
   System.out.println("");
   System.out.println("");
   System.out.println("Parameters are not correct.  Command syntax is:");
   System.out.println("");
   System.out.println("   IFSList as400 directory");
```

```
            System.out.println("");
            System.out.println("Where");
            System.out.println("");
            System.out.println("   as400 = system that contains the files");
            System.out.println("   directory = directory to be listed");
            System.out.println("");
            System.out.println("For example:");
            System.out.println("");
            System.out.println("   IFSCopyFile mySystem /dir1/dir2");
            System.out.println("");
            System.out.println("");
         }

         System.exit(0);
      }
   }


   ////////////////////////////////////////////////////////////////////////
   //
   // The directory filter class prints information from the file object.
   //
   // Another way to use the filter is to simply return true or false
   // based on information in the file object.  This lets the mainline
   // function decide what to do with the list of files that meet the
   // search criteria.
   //
   ////////////////////////////////////////////////////////////////////////



   class MyDirectoryFilter implements IFSFileFilter
   {
      public boolean accept(IFSFile file)
      {
         try
         {
            // Print the name of the current file

            System.out.print(file.getName());



            // Pad the output so the columns line up

            for (int i = file.getName().length(); i < 18; i++)
               System.out.print(" ");
```

```
            // Print the date the file was last changed.

            long changeDate = file.lastModified();
            Date d = new Date(changeDate);
            System.out.print(d);
            System.out.print("  ");



            // Print if the entry is a file or directory

            System.out.print("   ");

            if (file.isDirectory())
               System.out.println("<DIR>");
            else
               System.out.println(file.length());



            // Keep this entry.  Returning true tells the IFSList object
            // to return this file in the list of entries returned to the
            // .list() method.

            return true;
         }

         catch (Exception e)
         {
            return false;
         }
      }
   }
```

例: JDBCPopulate を使用してテーブルを作成し、そこにデータを挿入する

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// JDBCPopulate example.  This program uses the IBM Toolbox for Java JDBC driver to
// create and populate a table.
//
// Command syntax:
//    JDBCPopulate system collectionName tableName
//
// For example,
//    JDBCPopulate MySystem MyLibrary MyTable
//
///////////////////////////////////////////////////////////////////////////

import java.sql.*;

public class JDBCPopulate
{



    // Strings to be added in the WORD column of the table.
    private static final String words[]
        = { "One",      "Two",      "Three",    "Four",     "Five",
            "Six",      "Seven",    "Eight",    "Nine",     "Ten",
            "Eleven",   "Twelve",   "Thirteen", "Fourteen", "Fifteen",
            "Sixteen",  "Seventeen","Eighteen", "Nineteen", "Twenty" };



    public static void main (String[] parameters)
    {
        // Check the input parameters.
        if (parameters.length != 3) {
            System.out.println("");
            System.out.println("Usage:");
            System.out.println("");
            System.out.println("   JDBCPopulate system collectionName tableName");
            System.out.println("");
            System.out.println("");
            System.out.println("For example:");
            System.out.println("");
            System.out.println("");
            System.out.println("   JDBCPopulate MySystem MyLibrary MyTable");
            System.out.println("");
            return;
```

```
}

String system          = parameters[0];
String collectionName  = parameters[1];
String tableName       = parameters[2];

Connection connection  = null;

try {

    // Load the IBM Toolbox for Java JDBC driver.
    DriverManager.registerDriver(new com.ibm.as400.access.AS400JDBCDriver());

    // Get a connection to the database.  Since we do not
    // provide a user id or password, a prompt will appear.
    //
    // Note that we provide a default schema here so
    // that we do not need to qualify the table name in
    // SQL statements.
    //
    connection = DriverManager.getConnection ("jdbc:as400://"
        + system + "/" + collectionName);

    // Drop the table if it already exists.
    try {
        Statement dropTable = connection.createStatement ();
        dropTable.executeUpdate ("DROP TABLE " + tableName);
    }
    catch (SQLException e) {
        // Ignore.
    }

    // Create the table.
    Statement createTable = connection.createStatement ();
    createTable.executeUpdate ("CREATE TABLE " + tableName
        + " (I INTEGER, WORD VARCHAR(20), SQUARE INTEGER, "
        + " SQUAREROOT DOUBLE)");

    // Prepare a statement for inserting rows.  Since we
    // execute this multiple times, it is best to use a
    // PreparedStatement and parameter markers.
    PreparedStatement insert = connection.prepareStatement ("INSERT INTO "
        + tableName + " (I, WORD, SQUARE, SQUAREROOT) "
        + " VALUES (?, ?, ?, ?)");

    // Populate the table.
    for (int i = 1; i <= words.length; ++i) {
        insert.setInt (1, i);
        insert.setString (2, words[i-1]);
        insert.setInt (3, i*i);
```

```
                insert.setDouble (4, Math.sqrt(i));
                insert.executeUpdate ();
            }

            // Output a completion message.
            System.out.println ("Table " + collectionName + "." + tableName
                + " has been populated.");
        }

        catch (Exception e) {
            System.out.println ();
            System.out.println ("ERROR: " + e.getMessage());
        }

        finally {

            // Clean up.
            try {
                if (connection != null)
                    connection.close ();
            }
            catch (SQLException e) {
                // Ignore.
            }
        }

        System.exit (0);
    }


}
```

例: JDBCQuery を使用してテーブルを照会する

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// JDBCQuery example.  This program uses the IBM Toolbox for Java JDBC driver to
// query a table and output its contents.
//
// Command syntax:
//    JDBCQuery system collectionName tableName
//
// For example,
//    JDBCQuery MySystem qiws qcustcdt
//
///////////////////////////////////////////////////////////////////////////

import java.sql.*;

public class JDBCQuery
{



    // Format a string so that it has the specified width.
    private static String format (String s, int width)
    {
        String formattedString;

        // The string is shorter than specified width,
        // so we need to pad with blanks.
        if (s.length() < width) {
            StringBuffer buffer = new StringBuffer (s);
            for (int i = s.length(); i < width; ++i)
                buffer.append (" ");
            formattedString = buffer.toString();
        }

        // Otherwise, we need to truncate the string.
        else
            formattedString = s.substring (0, width);

        return formattedString;
    }



   public static void main (String[] parameters)
    {
        // Check the input parameters.
        if (parameters.length != 3) {
```

```
    System.out.println("");
    System.out.println("Usage:");
    System.out.println("");
    System.out.println("   JDBCQuery system collectionName tableName");
    System.out.println("");
    System.out.println("");
    System.out.println("For example:");
    System.out.println("");
    System.out.println("");
    System.out.println("   JDBCQuery mySystem qiws qcustcdt");
    System.out.println("");
    return;
}

String system          = parameters[0];
String collectionName  = parameters[1];
String tableName       = parameters[2];

Connection connection  = null;

try {

    // Load the IBM Toolbox for Java JDBC driver.
    DriverManager.registerDriver(new com.ibm.as400.access.AS400JDBCDriver());

    // Get a connection to the database.  Since we do not
    // provide a user id or password, a prompt will appear.
    connection = DriverManager.getConnection ("jdbc:as400://" + system);
    DatabaseMetaData dmd = connection.getMetaData ();

    // Execute the query.
    Statement select = connection.createStatement ();
    ResultSet rs = select.executeQuery ("SELECT * FROM "
        + collectionName + dmd.getCatalogSeparator() + tableName);

    // Get information about the result set.  Set the column
    // width to whichever is longer: the length of the label
    // or the length of the data.
    ResultSetMetaData rsmd = rs.getMetaData ();
    int columnCount = rsmd.getColumnCount ();
    String[] columnLabels = new String[columnCount];
    int[] columnWidths = new int[columnCount];
    for (int i = 1; i <= columnCount; ++i) {
        columnLabels[i-1] = rsmd.getColumnLabel (i);
        columnWidths[i-1] = Math.max (columnLabels[i-1].length(),
            rsmd.getColumnDisplaySize (i));
    }

    // Output the column headings.
    for (int i = 1; i <= columnCount; ++i) {
```

```
            System.out.print (format (rsmd.getColumnLabel(i), columnWidths[i-1]));
            System.out.print (" ");
        }
        System.out.println ();

        // Output a dashed line.
        StringBuffer dashedLine;
        for (int i = 1; i <= columnCount; ++i) {
            for (int j = 1; j <= columnWidths[i-1]; ++j)
                System.out.print ("-");
            System.out.print (" ");
        }
        System.out.println ();

        // Iterate throught the rows in the result set and output
        // the columns for each row.
        while (rs.next ()) {
            for (int i = 1; i <= columnCount; ++i) {
                String value = rs.getString (i);
                if (rs.wasNull ())
                    value = "<null>";
                System.out.print (format (value, columnWidths[i-1]));
                System.out.print (" ");
            }
            System.out.println ();
        }

    }

    catch (Exception e) {
        System.out.println ();
        System.out.println ("ERROR: " + e.getMessage());
    }

    finally {

        // Clean up.
        try {
            if (connection != null)
                connection.close ();
        }
        catch (SQLException e) {
            // Ignore.
        }
    }

    System.exit (0);
}
```

```
}
```

例: JobList を使用してジョブ ID 情報のリストを表示する

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// This program is an example of the Job support in the IBM Toolbox
// for Java.  It lists job identification information for a specific
// user on the system.
//
// Command syntax:
//    listJobs2 system userID password
//
///////////////////////////////////////////////////////////////////////////

import java.io.*;
import java.lang.*;
import java.util.*;
import com.ibm.as400.access.*;

public class listJobs2 extends Object
{

                    // Create an object in case we want to call
                    // any non-staic methods.
   public static void main(String[] parameters)
   {
      listJobs2 me = new listJobs2();
      me.Main(parameters);

      System.exit(0);
   }



   void Main(String[] parameters)
   {

                    // If a system was not specified, display help text and exit.
      if (parameters.length == 0)
      {
         showHelp();
         return;
      }

                    // Assign the parameters to variables.  The
                    // first parameter is assumed to be the system
                    // name the second is a userID and the third
```

```
            // is a password.
String systemName = parameters[0];
String userID     = null;
String password   = null;

if (parameters.length > 1)
   userID = parameters[1].toUpperCase();

if (parameters.length >= 2)
   password = parameters[2].toUpperCase();

System.out.println(" ");



try
{

            // Create an AS400 object using the system name
            // specified by the user.   Set the userid and
            // password if specified by the user.
   AS400 as400 = new AS400(parameters[0]);

   if (userID != null)
      as400.setUserId(userID);

   if (password != null)
      as400.setPassword(password);



   System.out.println("retrieving list ...     ");


            // Create a jobList object.  This object is used
            // to retrieve the list of active jobs on the server.
   JobList jobList = new JobList(as400);


            // Get the list of active jobs.
   Enumeration list = jobList.getJobs();


            // For each job in the list ...
   while (list.hasMoreElements())
   {

            // Get a job off the list.  If a userID was
```

```
                // specified then print identification information
                // only if the job's user matches the userID.  If
                // no userID was specified then print information
                // for every job on the system.
          Job j = (Job) list.nextElement();

          if (userID != null)
          {
             if (j.getUser().trim().equalsIgnoreCase(userID))
             {
                System.out.println(j.getName().trim() + "." +
                                   j.getUser().trim() + "." +
                                   j.getNumber());
             }
          }
          else
             System.out.println(j.getName().trim() + "." +
                                j.getUser().trim() + "." +
                                j.getNumber());
       }

    }
    catch (Exception e)
    {
       System.out.println("Unexpected error");
       e.printStackTrace();
    }
}



    // Display help text when parameters are incorrect.
void showHelp()
{
      System.out.println("");
      System.out.println("");
      System.out.println("");
      System.out.println("Parameters are not correct.  Command syntax is:");
      System.out.println("");
      System.out.println("   listJobs2 System UserID Password");
      System.out.println("");
      System.out.println("Where");
      System.out.println("");
      System.out.println("   System   = server to connect to");
      System.out.println("   UserID   = valid userID on that system ");
      System.out.println("   Password = password for the UserID (optional)");
      System.out.println("");
      System.out.println("For example:");
```

```
            System.out.println("");
            System.out.println("   listJobs2 MYAS400 JavaUser pwd1");
            System.out.println("");
            System.out.println("");
    }
}
```

例: JobList を使用してジョブのリストを取得する

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// This program is an example of the "job" classes in the IBM
// Toolbox for Java.  It gets a list of jobs on the server and
// outputs the job's status followed by job identifier.
//
//
// Command syntax:
//    listJobs system userID password
//
// (UserID and password are optional)
//
///////////////////////////////////////////////////////////////////////////

import java.io.*;
import java.util.*;
import com.ibm.as400.access.*;

public class listJobs extends Object
{
   public static void main(String[] parameters)
   {
      listJobs me = new listJobs();
      me.Main(parameters);

      System.exit(0);
   }



   void Main(String[] parameters)
   {

      // If a system was not specified, display help text and exit.
      if (parameters.length == 0)
      {
         showHelp();
         return;
      }

      // Set up AS400 object parms.  The first is the system name and must
      // be specified by the user.  The second and third are optional.  They
      // are the userid and password.  Convert the userid and password
      // to uppercase before setting them on the AS400 object.
      String userID   = null;
      String password = null;
```

```
   if (parameters.length > 1)
      userID = parameters[1].toUpperCase();

   if (parameters.length >= 2)
      password = parameters[2].toUpperCase();

   System.out.println(" ");

   try
   {

      // Create an AS400 object using the system name specified by the user.
      AS400 as400 = new AS400(parameters[0]);

      // If a userid and/or password was specified, set them on the
      // AS400 object.
      if (userID != null)
         as400.setUserId(userID);

      if (password != null)
         as400.setPassword(password);

      // Create a job list object.  Input parm is the AS400 we want job
      // information from.
      JobList jobList = new JobList(as400);

      // Get a list of jobs running on the server.
      Enumeration listOfJobs = jobList.getJobs();

      // For each job in the list print information about the job.
      while (listOfJobs.hasMoreElements())
      {
         printJobInfo((Job) listOfJobs.nextElement(), as400);
      }

   }
   catch (Exception e)
   {
      System.out.println("Unexpected error");
      System.out.println(e);
   }
}



void printJobInfo(Job job, AS400 as400)
{
```

```
// Create the various converters we need
AS400Bin4 bin4Converter   = new AS400Bin4( );
AS400Text text26Converter = new AS400Text(26, as400);
AS400Text text16Converter = new AS400Text(16, as400);
AS400Text text10Converter = new AS400Text(10, as400);
AS400Text text8Converter  = new AS400Text(8,  as400);
AS400Text text6Converter  = new AS400Text(6,  as400);
AS400Text text4Converter  = new AS400Text(4,  as400);


// We have the job name/number/etc. from the list request.  Now
// make a server API call to get the status of the job.
try
{
   // Create a program call object
   ProgramCall pgm = new ProgramCall(as400);

   // The server program we call has five parameters
   ProgramParameter[] parmlist = new ProgramParameter[5];

   // The first parm is a byte array that holds the output
   // data.  We will allocate a 1k buffer for output data.
   parmlist[0] = new ProgramParameter( 1024 );

   // The second parm is the size of our output data buffer (1K).
   Integer iStatusLength = new Integer( 1024 );
   byte[]  statusLength = bin4Converter.toBytes( iStatusLength );
   parmlist[1] = new ProgramParameter( statusLength );

   // The third parm is the name of the format of the data.
   // We will use format JOBI0200 because it has job status.
   byte[] statusFormat = text8Converter.toBytes("JOBI0200");
   parmlist[2] = new ProgramParameter( statusFormat );

   // The fourth parm is the job name is format "name user number".
   // Name must be 10 characters, user must be 10 characters and
   // number must be 6 characters.  We will use a text converter
   // to do the conversion and padding.
   byte[] jobName = text26Converter.toBytes(job.getName());

   int    i       = text10Converter.toBytes(job.getUser(),
                                            jobName,
                                            10);

          i       = text6Converter.toBytes(job.getNumber(),
                                           jobName,
                                           20);

   parmlist[3] = new ProgramParameter( jobName );
```

```
      // The last paramter is job identifier.  We will leave this blank.
      byte[] jobID = text16Converter.toBytes("                ");
      parmlist[4] = new ProgramParameter( jobID );


      // Run the program.
      if (pgm.run( "/QSYS.LIB/QUSRJOBI.PGM", parmlist )==false)
      {
         // if the program failed display the error message.
         AS400Message[] msgList = pgm.getMessageList();
         System.out.println(msgList[0].getText());
      }
      else
      {
         // else the program worked.  Output the status followed by
         // the jobName.user.jobID
         byte[] as400Data = parmlist[0].getOutputData();
         System.out.print("  " + text4Converter.toObject(as400Data, 107) + "  ");

         System.out.println(job.getName().trim() + "." +
                            job.getUser().trim() + "." +
                            job.getNumber() + "   ");
      }

   }
   catch (Exception e)
   {
      System.out.println(e);
   }

}



   // Display help text when parameters are incorrect.
void showHelp()
{
      System.out.println("");
      System.out.println("");
      System.out.println("");
      System.out.println("Parameters are not correct.  Command syntax is:");
      System.out.println("");
      System.out.println("   listJobs System UserID Password");
      System.out.println("");
      System.out.println("Where");
      System.out.println("");
      System.out.println("   System   = server to connect to");
      System.out.println("   UserID   = valid userID on that system (optional)");
```

```
        System.out.println("   Password = password for the UserID (optional)");
        System.out.println("");
        System.out.println("For example:");
        System.out.println("");
        System.out.println("   listJobs MYAS400 JavaUser pwd1");
        System.out.println("");
        System.out.println("");
    }
}
```

例: JobLog を使用してジョブ・ログ内のメッセージを表示する

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////
//
// This program is an example of the job log fuction of the IBM
// Toolbox for Java.  It will display the messages in the job log
// for a job that belongs to the current user.
//
// Command syntax:
//    jobLogExample system userID password
//
// (Password is optional)
//
///////////////////////////////////////////////////////////////////////

import java.lang.*;
import java.util.*;
import com.ibm.as400.access.*;

public class jobLogExample
{


    public static void main (String[] args)
    {
        // If a system and user were not specified, display help text and exit.
        if (args.length < 2)
        {
            System.out.println("Usage:  jobLogExample system userid <password>");
            return;
        }

        String userID = null;

        try
        {
            // Create an AS400 object.  The system name was passed
            // as the first command line argument.  If a userid
            // and password were passed on the command line,
            // set those as well.
            AS400 system = new AS400 (args[0]);

            if (args.length > 1)
            {
               userID = args[1];
               system.setUserId(userID);
            }

            if (args.length > 2)
```

```
            system.setPassword(args[2]);


            // Create a job list object.  This object will be used to get
            // the list of active jobs on the system.  Once the list of
            // jobs is retrieved, the program will find a job for the
            // current user.
            JobList jobList = new JobList(system);

            // Get the list of active jobs on the AS/400
            Enumeration list = jobList.getJobs();

            boolean Continue = true;

            // Look through the list to find a job for the current user.
            while (list.hasMoreElements() && Continue)
            {
               Job j = (Job) list.nextElement();

               if (j.getUser().trim().equalsIgnoreCase(userID))
               {
                  // A job matching the current user was found.  Create
                  // a job log object for this job.
                  JobLog jlog = new JobLog(system,
                                           j.getName(),
                                           j.getUser(),
                                           j.getNumber());

                  // Enumerate the messages in the job log then print them.
                  Enumeration messageList = jlog.getMessages();

                  while (messageList.hasMoreElements())
                  {
                     AS400Message message = (AS400Message) messageList.nextElement();
                     System.out.println(message.getText());
                  }

                  // We found one job matching the current user so exit.
                  Continue = false;
               }
            }
         }
         catch (Exception e)
         {
            System.out.println ("Error: " + e.getMessage ());
         }

         System.exit(0);
      }
```

}

例: スプール・ファイルを作成する

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////
//
// Example that shows creating a spooled file on a server from an input stream.
//
///////////////////////////////////////////////////////////////////////

import java.io.*;
import java.util.*;

import com.ibm.as400.access.*;

class NPExampleCreateSplf
{

// method to create the spooled file on the specified system, in the specified
// output queue from the given input stream.
public SpooledFile createSpooledFile(AS400 system,
                                     OutputQueue outputQueue,
                                     InputStream in)
{
    SpooledFile spooledFile = null;
    try
    {
        byte[] buf = new byte[2048];
        int bytesRead;
        SpooledFileOutputStream out;
        PrintParameterList parms = new PrintParameterList();

        // create a PrintParameterList with the values that we want
        // to override from the default printer file...we will override
        // the output queue and the copies value.
        parms.setParameter(PrintObject.ATTR_COPIES, 4);
        if (outputQueue != null)
        {
           parms.setParameter(PrintObject.ATTR_OUTPUT_QUEUE, outputQueue.getPath());
        }
        out = new SpooledFileOutputStream(system,
                                          parms,
                                          null,
                                          null);

        // read from the inputstream in until end of stream, passing all data
        // to the spooled file output stream.
        do
        {
            bytesRead = in.read(buf);
            if (bytesRead != -1)
            {
```

```
                out.write(buf);
            }
        } while (bytesRead != -1);

        out.close();  // close the spooled file

        spooledFile = out.getSpooledFile();   // get a reference to the new spooled file

    }
    catch (Exception e)
    {
        //...handle exception...
    }
    return spooledFile;
}

}
```

例: SCS スプール・ファイルを作成する

以下の例では、SCS3812Writer クラスを使用して SCS データ・ストリームを生成し、 サーバー上のスプール・ファイルにそれを書き込みます。

このアプリケーションは以下の引き数をとることができますが、 定義されているデフォルト値を使用することもできます。

- スプール・ファイルを受信するサーバーの名前
- スプール・ファイルを受信するサーバー上の出力キューの名前

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////
//
// This source is an example of IBM Toolbox for Java "SCS3812Writer".
//
///////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;

class NPExampleCreateSCSSplf
{
    private static final String DEFAULT_SYSTEM = new String("RCHAS1");
    private static final String DEFAULT_OUTQ = new String("/QSYS.LIB/QUSRSYS.LIB/PRT01.OUTQ");

    public static void main(String [] args)
    {
        try
        {
            AS400 system;
            SpooledFileOutputStream out;
            PrintParameterList parms = new PrintParameterList();
            SCS3812Writer scsWtr;

            // Process the arguments.
            if (args.length >= 1)
            {
                system = new AS400(args[0]);    // Create an AS400 object
            } else {
                system = new AS400(DEFAULT_SYSTEM);
            }

            if (args.length >= 2)               // Set the outq
            {
                parms.setParameter(PrintObject.ATTR_OUTPUT_QUEUE, args[1]);
            } else {
                parms.setParameter(PrintObject.ATTR_OUTPUT_QUEUE, DEFAULT_OUTQ);
            }

            out = new SpooledFileOutputStream(system, parms, null, null);

            scsWtr = new SCS3812Writer(out, 37);

            // Write the contents of the spool file.
            scsWtr.setLeftMargin(1.0);
            scsWtr.absoluteVerticalPosition(6);
            scsWtr.setFont(scsWtr.FONT_COURIER_BOLD_5);
            scsWtr.write("          Java Printing");
            scsWtr.newLine();
```

```
            scsWtr.newLine();
            scsWtr.setCPI(10);
            scsWtr.write("This document was created using the IBM Toolbox for Java.");
            scsWtr.newLine();
            scsWtr.write("The rest of this document shows some of the things that");
            scsWtr.newLine();
            scsWtr.write("can be done with the SCS3812Writer class.");
            scsWtr.newLine();
            scsWtr.newLine();
            scsWtr.setUnderline(true); scsWtr.write("Setting fonts:"); scsWtr.setUnderline(false);
            scsWtr.newLine();
            scsWtr.setFont(scsWtr.FONT_COURIER_10); scsWtr.write("Courier font ");
            scsWtr.setFont(scsWtr.FONT_COURIER_BOLD_10); scsWtr.write(" Courier bold font ");
            scsWtr.setFont(scsWtr.FONT_COURIER_ITALIC_10); scsWtr.write(" Courier italic font ");
            scsWtr.newLine();
            scsWtr.setBold(true); scsWtr.write("Courier bold italic font ");
            scsWtr.setBold(false);
            scsWtr.setCPI(10);
            scsWtr.newLine();
            scsWtr.newLine();
            scsWtr.setUnderline(true); scsWtr.write("Lines per inch:"); scsWtr.setUnderline(false);
            scsWtr.newLine();
            scsWtr.write("The following lines should print at 8 lines per inch.");
            scsWtr.newLine();
            scsWtr.newLine();
            scsWtr.setLPI(8);
            scsWtr.write("Line one"); scsWtr.newLine();
            scsWtr.write("Line two"); scsWtr.newLine();
            scsWtr.write("Line three"); scsWtr.newLine();
            scsWtr.write("Line four"); scsWtr.newLine();
            scsWtr.write("Line five"); scsWtr.newLine();
            scsWtr.write("Line six"); scsWtr.newLine();
            scsWtr.write("Line seven"); scsWtr.newLine();
            scsWtr.write("Line eight"); scsWtr.newLine();
            scsWtr.endPage();
            scsWtr.setLPI(6);
            scsWtr.setSourceDrawer(1);
            scsWtr.setTextOrientation(0);
            scsWtr.absoluteVerticalPosition(6);
            scsWtr.write("This page should print in portrait orientation from drawer 1.");
            scsWtr.endPage();
            scsWtr.setSourceDrawer(2);
            scsWtr.setTextOrientation(90);
            scsWtr.absoluteVerticalPosition(6);
            scsWtr.write("This page should print in landscape orientation from drawer 2.");
            scsWtr.endPage();
            scsWtr.close();
            System.out.println("Sample spool file created.");
            System.exit(0);
        }
        catch (Exception e)
        {
            // Handle error.
            System.out.println("Exception occured while creating spooled file. " + e);
            System.exit(0);
        }
    }
}
```

例: スプール・ファイルを読み取る

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////
//
// Example that reads an existing server spooled file.
//
// This source is an example of IBM Toolbox for Java "PrintObjectInputStream".
//
///////////////////////////////////////////////////////////////////////
        try{
        byte[] buf = new byte[2048];
        int bytesRead;
        AS400 sys = new AS400();
        SpooledFile splf = new SpooledFile( sys,          // AS400
                                            "MICR",       // splf name
                                            17,           // splf number
                                            "QPRTJOB",    // job name
                                            "QUSER",      // job user
                                            "020791" );   // job number

        // open the spooled file for reading and get the input stream to
        // read from it.
        InputStream in = splf.getInputStream(null);

        do
        {
            // read up to buf.length bytes of raw spool data into
            // our buffer.  The actual bytes read will be returned.
            // The data will be a binary printer data stream that is the
            // contents of the spooled file.
            bytesRead = in.read( buf );
            if( bytesRead != -1 )
            {
                // process the spooled file data.
               System.out.println( "Read " + bytesRead + " bytes" );
            }
        } while( bytesRead != -1 );

        in.close();
    }
    catch( Exception e )
    {
        // exception
    }
```

例: スプール・ファイルを非同期でリストする (リスナーを使用)

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////
//
// Example that shows listing all spooled files on a system Asynchronously using
// the PrintObjectListListener interface to get feedback as the list is being built.
// Listing Asynchronously allows the caller to start processing the list objects
// before the entire list is built for a faster perceived response time
// for the user.
//
///////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.AS400;
import com.ibm.as400.access.SpooledFileList;
import com.ibm.as400.access.SpooledFile;
import com.ibm.as400.access.ExtendedIllegalStateException;
import com.ibm.as400.access.PrintObjectListListener;
import com.ibm.as400.access.PrintObjectListEvent;

public class NPExampleListSplfAsynch extends Object
                                     implements PrintObjectListListener
{
    private AS400 system_;
    private boolean fListError;
    private boolean fListClosed;
    private boolean fListCompleted;
    private Exception listException;
    private int listObjectCount;

    public NPExampleListSplfAsynch(AS400 system)
    {
        system_ = system;
    }

    // list all spooled files on the system asynchronously using a listener
    public void listSpooledFiles()
    {
        fListError = false;
        fListClosed = false;
        fListCompleted = false;
        listException = null;
        listObjectCount = 0;

        try
        {
            String strSpooledFileName;
            boolean fCompleted = false;
            int listed = 0, size;

            if( system_ == null )
            {
                system_ = new AS400();
            }

            System.out.println(" Now receiving all spooled files Asynchronously using a listener");
```

```
    SpooledFileList splfList = new SpooledFileList(system_);

    // set filters, all users, on all queues
    splfList.setUserFilter("*ALL");
    splfList.setQueueFilter("/QSYS.LIB/%ALL%.LIB/%ALL%.OUTQ");

    // add the listener.
    splfList.addPrintObjectListListener(this);

    // open the list, openAsynchronously returns immediately
    splfList.openAsynchronously();

    do
    {
        // wait for the list to have at least 25 objects or to be done
        waitForWakeUp();

        fCompleted = splfList.isCompleted();
        size = splfList.size();

        // output the names of all objects added to the list
        // since we last woke up
        while (listed < size)
        {
            if (fListError)
            {
                System.out.println(" Exception on list - " + listException);
                break;
            }

            if (fListClosed)
            {
            System.out.println(" The list was closed before it completed!");
                break;
            }

            SpooledFile splf = (SpooledFile)splfList.getObject(listed++);
            if (splf != null)
            {
                // output this spooled file name
                strSpooledFileName = splf.getStringAttribute(SpooledFile.ATTR_SPOOLFILE);
                System.out.println(" spooled file = " + strSpooledFileName);
            }
        }

    } while (!fCompleted);

    // clean up after we are done with the list
    splfList.close();
    splfList.removePrintObjectListListener(this);
}

catch( ExtendedIllegalStateException e )
{
    System.out.println(" The list was closed before it completed!");
}

catch( Exception e )
```

```
    {
        // ...handle any other exceptions...
        e.printStackTrace();
    }

}

// This is where the foreground thread waits to be awaken by the
// the background thread when the list is updated or it ends.
private synchronized void waitForWakeUp()
   throws InterruptedException
{
    // don''t go back to sleep if the listener says the list is done
    if (!fListCompleted)
    {
        wait();
    }
}

// The following methods implement the PrintObjectListListener interface

// This method is invoked when the list is closed.
public void listClosed(PrintObjectListEvent event)
{
    System.out.println("*****The list was closed*****");
    fListClosed = true;
    synchronized(this)
    {
        // Set flag to indicate that the list has
        // completed and wake up foreground thread.
        fListCompleted = true;
        notifyAll();
    }
}

// This method is invoked when the list is completed.
public void listCompleted(PrintObjectListEvent event)
{
    System.out.println("*****The list has completed*****");
    synchronized (this)
    {
        // Set flag to indicate that the list has
        // completed and wake up foreground thread.
        fListCompleted = true;
        notifyAll();
    }
}

// This method is invoked if an error occurs while retrieving
// the list.
public void listErrorOccurred(PrintObjectListEvent event)
{
    System.out.println("*****The list had an error*****");
    fListError = true;
    listException = event.getException();
    synchronized(this)
    {
        // Set flag to indicate that the list has
```

```
            // completed and wake up foreground thread.
            fListCompleted = true;
            notifyAll();
        }
    }

    // This method is invoked when the list is opened.
    public void listOpened(PrintObjectListEvent event)
    {
        System.out.println("*****The list was opened*****");
        listObjectCount = 0;
    }

    // This method is invoked when an object is added to the list.
    public void listObjectAdded(PrintObjectListEvent event)
    {
        // every 25 objects we'll wake up the foreground
        // thread to get the latest objects...
        if( (++listObjectCount % 25) == 0 )
        {
            System.out.println("*****25 more objects added to the list*****");
            synchronized (this)
            {
                // wake up foreground thread
                notifyAll();
            }
        }
    }

    public static void main( String args[] )
    {
        NPExampleListSplfAsynch list = new NPExampleListSplfAsynch(new AS400());
        try{
            list.listSpooledFiles();
        }
        catch( Exception e )
        {
            e.printStackTrace();
        }
        System.exit(0);
    }

}
```

例: スプール・ファイルを非同期でリストする (リスナーを使用しない)

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////
//
// Example that shows listing all spooled files on a system Asynchronously without
// using the PrintObjectListListener interface. After opening the list the caller
// can do some additional work before waiting for the list to complete.
//
///////////////////////////////////////////////////////////////////////
//
// This source is an example of IBM Toolbox for Java "PrintObjectList".
//
///////////////////////////////////////////////////////////////////////

import java.util.Enumeration;

import com.ibm.as400.access.AS400;
import com.ibm.as400.access.SpooledFileList;
import com.ibm.as400.access.SpooledFile;

public class NPExampleListSplfAsynch2 extends Object
{
    private AS400 system_;

    public NPExampleListSplfAsynch2(AS400 system)
    {
        system_ = system;
    }

    // list all spooled files on the system asynchronously
    public void listSpooledFiles()
    {
        try
        {
            String strSpooledFileName;
            int listed, size;

            if( system_ == null )
            {
                system_ = new AS400();
            }

            System.out.println(" Now receiving all spooled files Asynchronously without using a
listener");

            SpooledFileList splfList = new SpooledFileList(system_);

            // set filters, all users, on all queues
            splfList.setUserFilter("*ALL");
            splfList.setQueueFilter("/QSYS.LIB/%ALL%.LIB/%ALL%.OUTQ");

            // open list, openAsynchronously() returns immediately
            // we have not added any listeners...
            splfList.openAsynchronously();
```

```
            System.out.println(" Do some processing before waiting...");

            // ... do some processing here while the list is being built....

            System.out.println(" Now wait for list to complete.");

            // wait for the list to complete
            splfList.waitForListToComplete();

            Enumeration enum = splfList.getObjects();

            // output the name of all objects on the list
            while( enum.hasMoreElements() )
            {
                SpooledFile splf = (SpooledFile)enum.nextElement();
                if (splf != null)
                {
                    // output this spooled file's name
                    strSpooledFileName = splf.getStringAttribute(SpooledFile.ATTR_SPOOLFILE);
                    System.out.println(" spooled file = " + strSpooledFileName);
                }
            }
            // clean up after we are done with the list
            splfList.close();
        }

        catch( Exception e )
        {
            // ...handle any exceptions...
            e.printStackTrace();
        }
    }

    public static void main( String args[] )
    {
        NPExampleListSplfAsynch2 list = new NPExampleListSplfAsynch2(new AS400());
        try{
            list.listSpooledFiles();
        }
        catch( Exception e )
        {
            e.printStackTrace();
        }
        System.exit(0);
    }
}
```

例: スプール・ファイルを同期でリストする

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////
//
// Example that shows listing all spooled files on a system Synchronously.
// Listing Synchronously does not return to the caller until the complete list
// is built. The user perceives a slower response time then listing Asynchronously.
//
///////////////////////////////////////////////////////////////////////
//
// This source is an example of IBM Toolbox for Java "PrintObjectList".
//
///////////////////////////////////////////////////////////////////////

import java.util.Enumeration;

import com.ibm.as400.access.AS400;
import com.ibm.as400.access.SpooledFileList;
import com.ibm.as400.access.SpooledFile;

public class NPExampleListSplfSynch
{
    private AS400 system_ = new AS400();

    public NPExampleListSplfSynch(AS400 system)
    {
        system_ = system;
    }

    public void listSpooledFiles()
    {
        try{
            String strSpooledFileName;

            if( system_ == null )
            {
                system_ = new AS400();
            }

            System.out.println(" Now receiving all spooled files Synchronously");

            SpooledFileList splfList = new SpooledFileList( system_ );

            // set filters, all users, on all queues
            splfList.setUserFilter("*ALL");
            splfList.setQueueFilter("/QSYS.LIB/%ALL%.LIB/%ALL%.OUTQ");

            // open list, openSynchronously() returns when the list is completed.
            splfList.openSynchronously();
            Enumeration enum = splfList.getObjects();

            while( enum.hasMoreElements() )
            {
```

```
                SpooledFile splf = (SpooledFile)enum.nextElement();
                if ( splf != null )
                {
                    // output this spooled file's name
                    strSpooledFileName = splf.getStringAttribute(SpooledFile.ATTR_SPOOLFILE);
                    System.out.println(" spooled file = " + strSpooledFileName);
                }
            }
            // clean up after we are done with the list
            splfList.close();
        }
        catch( Exception e )
        {
            // ...handle any exceptions...
            e.printStackTrace();
        }
    }

    public static void main( String args[] )
    {
        NPExampleListSplfSynch list = new NPExampleListSplfSynch(new AS400());
        try{
            list.listSpooledFiles();
        }
        catch( Exception e )
        {
            e.printStackTrace();
        }
        System.exit(0);
    }
}
```

例: ProgramCall を使用する

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////
//
// Program call example.  This program calls the QWCRSSTS server program
// to retrieve the status of the system.
//
// Command syntax:
//    PCSystemStatusExample system
//
// This source is an example of IBM Toolbox for Java "ProgramCall".
//
///////////////////////////////////////////////////////////////////////



import java.io.*;
import java.util.*;
import java.math.*;
import java.lang.Thread.*;
import com.ibm.as400.access.*;

public class PCSystemStatusExample extends Object
{
   public static void main(String[] parameters)
   {
      System.out.println( " " );



      // if a system was not specified, display help text and exit.

      if (parameters.length >= 1)
      {

         try
         {
            // Create an AS400 object for the server that contains the
            // program.  Assume the first parameter is the system name.

            AS400 as400 = new AS400(parameters[0]);



            // Create the path to the program.
```

```
QSYSObjectPathName programName = new QSYSObjectPathName("QSYS",
                                                         "QWCRSSTS",
                                                         "PGM");



// Create the program call object.  Assocate the object with the
// AS400 object that represents the server we get status from.

ProgramCall getSystemStatus = new ProgramCall(as400);



// Create the program parameter list.  This program has five
// parameters that will be added to this list.

ProgramParameter[] parmlist = new ProgramParameter[5];



// The server program returns data in parameter 1.  It is an output
// parameter.  Allocate 64 bytes for this parameter.

parmlist[0] = new ProgramParameter( 64 );



// Parameter 2 is the buffer size of parm 1.  It is a numeric input
// parameter.  Sets its value to 64, convert it to the server format,
// then add the parm to the parm list.

AS400Bin4 bin4 = new AS400Bin4( );
Integer iStatusLength = new Integer( 64 );
byte[]  statusLength = bin4.toBytes( iStatusLength );
parmlist[1] = new ProgramParameter( statusLength );



// Parameter 3 is the status-format parameter.  It is a string input
// parameter.  Set the string value, convert it to the server format,
// then add the parameter to the parm list.

AS400Text text1 = new AS400Text(8, as400);
byte[] statusFormat = text1.toBytes("SSTS0200");
parmlist[2] = new ProgramParameter( statusFormat );



// Parameter 4 is the reset-statistics parameter.  It is a string input
```

```
// parameter.  Set the string value, convert it to the server format,
// then add the parameter to the parm list.

AS400Text text3 = new AS400Text(10, as400);
byte[] resetStats = text3.toBytes("*NO       ");
parmlist[3] = new ProgramParameter( resetStats );



// Parameter 5 is the error info parameter.  It is an input/output
// parameter.  Add it to the parm list.

byte[] errorInfo = new byte[32];
parmlist[4] = new ProgramParameter( errorInfo, 0 );



// Set the program to call and the parameter list to the program
// call object.

getSystemStatus.setProgram(programName.getPath(), parmlist );



// Run the program then sleep.  We run the program twice because
// the first set of results are inflated.  If we discard the first
// set of results and run the command again five seconds later the
// number will be more accurate.

getSystemStatus.run();
Thread.sleep(5000);



// Run the program

if (getSystemStatus.run()!=true)
{

   // If the program did not run get the list of error messages
   // from the program object and display the messages.  The error
   // would be something like program-not-found or not-authorized
   // to the program.

   AS400Message[] msgList = getSystemStatus.getMessageList();

   System.out.println("The program did not run.  Server messages:");

   for (int i=0; i<msgList.length; i++)
```

```
    {
        System.out.println(msgList[i].getText());
    }
}


// Else the program did run.

else
{

    // Create a server to Java numeric converter.  This converter
    // will be used in the following section to convert the numeric
    // output from the server format to Java format.

    AS400Bin4 as400Int = new AS400Bin4( );



    // Get the results of the program.  Output data is in
    // a byte array in the first parameter.

    byte[] as400Data = parmlist[0].getOutputData();



    // CPU utilization is a numeric field starting at byte
    // 32 of the output buffer.  Convert this number from the
    // server format to Java format and output the number.

    Integer cpuUtil = (Integer)as400Int.toObject( as400Data, 32 );
    cpuUtil = new Integer(cpuUtil.intValue()/10);
    System.out.print("CPU Utilization:  ");
    System.out.print(cpuUtil);
    System.out.println("%");



    // DASD utilization is a numeric field starting at byte
    // 52 of the output buffer.  Convert this number from the
    // server format to Java format and output the number.

    Integer dasdUtil = (Integer)as400Int.toObject( as400Data, 52 );
    dasdUtil = new Integer(dasdUtil.intValue()/10000);
    System.out.print("Dasd Utilization: ");
    System.out.print(dasdUtil);
    System.out.println("%");
```

```
            // Number of jobs is a numeric field starting at byte
            // 36 of the output buffer.  Convert this number from the
            // server format to Java format and output the number.

            Integer nj = (Integer)as400Int.toObject( as400Data, 36 );
            System.out.print("Active jobs:      ");
            System.out.println(nj);

         }

            // This program is done running program so disconnect from
            // the command server on the server.  Program call and command
            // call use the same server on the server.

         as400.disconnectService(AS400.COMMAND);
      }
      catch (Exception e)
      {
         // If any of the above operations failed say the program failed
         // and output the exception.

         System.out.println("Program call failed");
         System.out.println(e);
      }
   }

   // Display help text when parameters are incorrect.

   else
   {
      System.out.println("");
      System.out.println("");
      System.out.println("");
      System.out.println("Parameters are not correct.  Command syntax is:");
      System.out.println("");
      System.out.println("   PCSystemStatusExample myServer");
      System.out.println("");
      System.out.println("Where");
      System.out.println("");
      System.out.println("   myServer = get status of this server ");
      System.out.println("");
      System.out.println("For example:");
      System.out.println("");
      System.out.println("   PCSystemStatusExample mySystem");
      System.out.println("");
      System.out.println("");
   }
```

```
        System.exit(0);
    }
}
```

例: レコード・レベルのアクセス・クラスを使用する

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// Record level access example.  This program will prompt the user
// for the name of the server and the file to display.  The file must exist
// and should contain records.  Each record in the file will be displayed
// to System.out.
//
// Calling syntax: java RLSequentialAccessExample
//
// This source is an example of IBM Toolbox for Java "RecordLevelAccess"
//
///////////////////////////////////////////////////////////////////////////



import java.io.*;
import java.util.*;
import com.ibm.as400.access.*;

public class RLSequentialAccessExample
{
   public static void main(String[] parmeters)
   {
      // Created a reader to get input from the user
      BufferedReader inputStream = new BufferedReader(new InputStreamReader(System.in),1);

      // Declare variables to hold the system name, library, file and member names
      String systemName = "";
      String library = "";
      String file = "";
      String member = "";

      // Get the system name and and file to display from the user
      System.out.println();
      try
      {
        System.out.print("System name: ");
        systemName = inputStream.readLine();

        System.out.print("Library in which the file exists: ");
        library = inputStream.readLine();

        System.out.print("File name: ");
        file = inputStream.readLine();

        System.out.print("Member name (press enter for first member): ");
        member = inputStream.readLine();
        if (member.equals(""))
        {
          member = "*FIRST";
        }

        System.out.println();
      }
      catch (Exception e)
      {
```

```
        System.out.println("Error obtaining user input.");
        e.printStackTrace();
        System.exit(0);
      }

      // Create AS400 object and connect for the record level access service.
      AS400 system = new AS400(systemName);
      try
      {
        system.connectService(AS400.RECORDACCESS);
      }
      catch(Exception e)
      {
        System.out.println("Unable to connect for record level access.");
        System.out.println("Check the readme file for special instructions regarding record level
access");
        e.printStackTrace();
        System.exit(0);
      }

      // Create a QSYSObjectPathName object to obtain the integrated file system path name form
      // of the file to be displayed.
      QSYSObjectPathName filePathName = new QSYSObjectPathName(library, file, member, "MBR");

      // Create a SequentialFile object representing the file to be displayed
      SequentialFile theFile = new SequentialFile(system, filePathName.getPath());

      // Retrieve the record format for the file
      AS400FileRecordDescription recordDescription = new AS400FileRecordDescription(system,
filePathName.getPath());
      try
      {
        RecordFormat[] format = recordDescription.retrieveRecordFormat();

        // Set the record format for the file
        theFile.setRecordFormat(format[0]);

        // Open the file for reading.  Read 100 records at a time if possible.
        theFile.open(AS400File.READ_ONLY, 100, AS400File.COMMIT_LOCK_LEVEL_NONE);

        // Display each record in the file
        System.out.println("Displaying file " + library.toUpperCase() + "/" + file.toUpperCase() + "("
+ theFile.getMemberName().trim() + "):");

        Record record = theFile.readNext();
        while (record != null)
        {
          System.out.println(record);
          record = theFile.readNext();
        }
        System.out.println();

        // Close the file
        theFile.close();

        // Disconnect from the record level access service
        system.disconnectService(AS400.RECORDACCESS);
      }
      catch (Exception e)
      {
```

```
      System.out.println("Error occurred attempting to display the file.");
      e.printStackTrace();

      try
      {
        // Close the file
        theFile.close();
      }
      catch(Exception x)
      {
      }

      // Disconnect from the record level access service
      system.disconnectService(AS400.RECORDACCESS);
      System.exit(0);
    }

    // Make sure that the application ends; see readme for details
    System.exit(0);
  }
}
```

例: レコード・レベルのアクセス・クラスを使用してファイルからレコードを読み取る

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// Record-Level Access example.  This program uses the record-level
// access classes to read records from a file on the server.
//
// Command syntax:
//    java RLReadFile system
//
// This program reads the records from CA/400's sample database file
// (QCUSTCDT in library QIWS).  If you change this example to update
// records you should make a copy of QCUSTCDT and update the copy.
//
// This source is an example of IBM Toolbox for Java "Record-level access".
//
///////////////////////////////////////////////////////////////////////////


import java.io.*;
import java.util.*;
import java.math.*;
import com.ibm.as400.access.*;

public class RLReadFile extends Object
{
   public static void main(String[] parameters)
   {

      String system = "";

      // Continue only if a system name was specified.

      if (parameters.length >= 1)
      {

         try
         {

            // Assume the first parameter is the system name.

            system = parameters[0];

            // Create an AS400 object for the server that has the file.

            AS400 as400 = new AS400(system);


            // Create a record description for the file.  The file is QCUSTCDT
            // in library QIWS.
```

```
ZonedDecimalFieldDescription customerNumber =
             new ZonedDecimalFieldDescription(new AS400ZonedDecimal(6,0),
                                              "CUSNUM");
CharacterFieldDescription lastName =
             new CharacterFieldDescription(new AS400Text(8, as400), "LSTNAM");

CharacterFieldDescription initials =
             new CharacterFieldDescription(new AS400Text(3, as400), "INIT");

CharacterFieldDescription street =
             new CharacterFieldDescription(new AS400Text(13, as400), "STREET");

CharacterFieldDescription city =
             new CharacterFieldDescription(new AS400Text(6, as400), "CITY");

CharacterFieldDescription state =
             new CharacterFieldDescription(new AS400Text(2, as400), "STATE");

ZonedDecimalFieldDescription zipCode =
             new ZonedDecimalFieldDescription(new AS400ZonedDecimal(5,0),
                                              "ZIPCOD");
ZonedDecimalFieldDescription creditLimit =
             new ZonedDecimalFieldDescription(new AS400ZonedDecimal(4,0),
                                              "CDTLMT");
ZonedDecimalFieldDescription chargeCode =
             new ZonedDecimalFieldDescription(new AS400ZonedDecimal(1,0),
                                              "CHGCOD");
ZonedDecimalFieldDescription balanceDue =
             new ZonedDecimalFieldDescription(new AS400ZonedDecimal(6,2),
                                              "BALDUE");
ZonedDecimalFieldDescription creditDue =
             new ZonedDecimalFieldDescription(new AS400ZonedDecimal(6,2),
                                              "CDTDUE");


// The record format name should be specified for a DDM file.
// In the case of the QCUSTCDT file, its record format is called CUSREC.

RecordFormat qcustcdt = new RecordFormat("CUSREC");

qcustcdt.addFieldDescription(customerNumber);
qcustcdt.addFieldDescription(lastName);
qcustcdt.addFieldDescription(initials);
qcustcdt.addFieldDescription(street);
qcustcdt.addFieldDescription(city);
qcustcdt.addFieldDescription(state);
qcustcdt.addFieldDescription(zipCode);
qcustcdt.addFieldDescription(creditLimit);
qcustcdt.addFieldDescription(chargeCode);
qcustcdt.addFieldDescription(balanceDue);
qcustcdt.addFieldDescription(creditDue);


// Create the sequential file object that represents the
```

```
// file on the server.  We use a QSYSObjectPathName object
// to get the name of the file into the correct format.

QSYSObjectPathName fileName = new QSYSObjectPathName("QIWS",
                                                     "QCUSTCDT",
                                                     "FILE");

SequentialFile file = new SequentialFile(as400, fileName.getPath());


// Let the file object know the format of the records.

file.setRecordFormat(qcustcdt);


// Open the file for read-only access.  Specify a blocking
// factor of 10 (the file object will get 10 records when
// it accesses the server for data).  Do not use commitment
// control.

file.open(SequentialFile.READ_ONLY,
          10,
          SequentialFile.COMMIT_LOCK_LEVEL_NONE);


// Read the first record of the file.

Record data = file.readNext();


// Loop while there are records in the file (while we have not
// reached end-of-file).

while (data != null)
{

   // Display the record only if balance due is greater than
   // zero.  In that case display the customer name and
   // the balance due.  The following code pulls fields out
   // of the record by field name.  As the field is retrieved
   // from the record it is converted from server format to
   // Java format.

   if (((BigDecimal)data.getField("BALDUE")).floatValue() > 0.0)
   {
      System.out.print((String) data.getField("INIT") + "  ");
      System.out.print((String) data.getField("LSTNAM") + "  ");
      System.out.println((BigDecimal) data.getField("BALDUE"));
   }

   // Read the next record in the file.

   data = file.readNext();
}
```

```
            // When there are no more records to read, disconnect from the server.

            as400.disconnectAllServices();
         }

         catch (Exception e)
         {

            // If any of the above operations failed, print an error message
            // and output the exception.

            System.out.println("Could not read the file");
            System.out.println(e);
         }
      }


      // Display help text when parameters are incorrect.

      else
      {
         System.out.println("");
         System.out.println("");
         System.out.println("");
         System.out.println("Parameters are not correct.  Command syntax is:");
         System.out.println("");
         System.out.println("   RLReadFile as400");
         System.out.println("");
         System.out.println("Where");
         System.out.println("");
         System.out.println("   as400 = system that contains the file");
         System.out.println("");
         System.out.println("For example:");
         System.out.println("");
         System.out.println("   RLReadFile mySystem");
         System.out.println("");
         System.out.println("");
         System.out.println("Note, this program reads data base file QIWS/QCUSTCDT.  ");
         System.out.println("");
         System.out.println("");
      }

      System.exit(0);

   }
}
```

例: レコード・レベルのアクセス・クラスを使用してキー別にレコードを読み取る

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// Record-Level Access example.  This program uses the record-level
// access classes to read records by key from a file on the server.
// The user will be prompted for the server name to which to run and
// the library in which to create file QCUSTCDTKY.
//
// Command syntax:
//    java RLKeyedFileExample
//
// This program will copy the records from the iSeries Access for Windows sample
// database file (QCUSTCDT in library QIWS) to file QCUSTCDTKY which has
// the same format as QIWS/QCUSTCDT but has set the CUSNUM field as the key
// for the file.
//
// This source is an example of IBM Toolbox for Java "Record-level access".
//
///////////////////////////////////////////////////////////////////////////


import java.io.*;
import java.util.*;
import java.math.*;
import com.ibm.as400.access.*;

public class RLKeyedFileExample
{
  public static void main(String[] parameters)
  {

    // Created a reader to get input from the user
    BufferedReader inputStream = new BufferedReader(new InputStreamReader(System.in),1);

    // Declare variables to hold the system name, library, file and member names
    String systemName = "";
    String library = "";

    // Get the system name from the user
    System.out.println();
    try
    {
      System.out.print("System name: ");
      systemName = inputStream.readLine();

      System.out.print("Library in which to create file QCUSTCDTKY: ");
      library = inputStream.readLine();
    }
    catch(Exception e)
    {
      System.out.println("Error obtaining user input.");
      e.printStackTrace();
      System.exit(0);
    }
```

```
    // Create AS400 object and connect for the record level access service.
    AS400 system = new AS400(systemName);
    try
    {
      system.connectService(AS400.RECORDACCESS);
    }
    catch(Exception e)
    {
      System.out.println("Unable to connect for record level access.");
      System.out.println("Check the readme file for special instructions regarding record level
access");
      e.printStackTrace();
      System.exit(0);
    }

    RecordFormat qcustcdtFormat = null;
    try
    {
      // Create the RecordFormat object for creating the file.  The record format for the new
      // file will be the same as the record format for file QIWS/QCUSTCDT.  However we will
      // make the CUSNUM field a key field.
      AS400FileRecordDescription recordDescription = new AS400FileRecordDescription(system,
"/QSYS.LIB/QIWS.LIB/QCUSTCDT.FILE");

      // There is only one record format for the file, so take the first (and only) element
      // of the RecordFormat array returned as the RecordFormat for the file.
      System.out.println("Retrieving record format of QIWS/QCUSTCDT...");
      qcustcdtFormat = recordDescription.retrieveRecordFormat()[0];
      // Indicate that CUSNUM is a key field
      qcustcdtFormat.addKeyFieldDescription("CUSNUM");
    }
    catch(Exception e)
    {
      System.out.println("Unable to retrieve record format from QIWS/QCUSTCDT");
      e.printStackTrace();
      System.exit(0);
    }

    // Create the keyed file object that represents the
    // file we will create on the server.  We use a QSYSObectPathName object
    // to get the name of the file into the correct format.
    QSYSObjectPathName fileName = new QSYSObjectPathName(library,
                                                         "QCUSTCDTKY",
                                                         "*FILE",
                                                         "MBR");
    KeyedFile file = new KeyedFile(system, fileName.getPath());

    try
    {
      System.out.println("Creating file " + library + "/QCUSTCDTKY...");
      // Create the file using the qcustcdtFormat object
      file.create(qcustcdtFormat, "Keyed QCUSTCDT file");

      // Populate the file with the records contained in QIWS/QCUSTCDT
      copyRecords(system, library);
```

```
      // Open the file for read-only access.  Because we will be randomly
      // accessing the file, specify a blocking factor of 1.  The
      // commit lock level parameter will be ignored since commitment
      // control has not been started.
      file.open(AS400File.READ_ONLY,
                1,
                AS400File.COMMIT_LOCK_LEVEL_NONE);

      // Assume that we want to display the information for customers
      // 192837, 392859 and 938472
      // The CUSNUM field is a zoned decimal field of length 6 with
      // no decimal positions.  Therefore, the key field value is
      // represented with a BigDecimal.
      BigDecimal[] keyValues = {new BigDecimal(192837), new BigDecimal(392859), new
BigDecimal(938472)};

      // Create the key for reading the records.  The key for a KeyedFile
      // is specified with an Object[]
      Object[] key = new Object[1];

      Record data = null;
      for (int i = 0; i < keyValues.length; i++)
      {
        // Setup the key for reading
        key[0] = keyValues[i];

        // Read the record for customer number keyValues[i]
        data = file.read(key);
        if (data != null)
        {
          // Display the record only if balance due is greater than
          // zero.  In that case display the customer name and
          // the balance due.  The following code pulls fields out
          // of the record by field name.  As the field is retrieved
          // from the record it is converted from the server format to
          // Java format.
          if (((BigDecimal)data.getField("BALDUE")).floatValue() > 0.0)
          {
            System.out.print((String) data.getField("INIT") + "  ");
            System.out.print((String) data.getField("LSTNAM") + "  ");
            System.out.println((BigDecimal) data.getField("BALDUE"));
          }
        }
      }

      // All done with the file
      file.close();

      // Get rid of the file from the user's system
      file.delete();
    }
    catch(Exception e)
    {
      System.out.println("Unable to create/read from QTEMP/QCUSTCDT");
      e.printStackTrace();
```

```
      try
      {
        file.close();
        // Get rid of the file from the user's system
        file.delete();
      }
      catch(Exception x)
      {
      }
    }

    // All done with record level access; disconnect from the
    // record-level access server.
    system.disconnectService(AS400.RECORDACCESS);
    System.exit(0);
  }

  public static void copyRecords(AS400 system, String library)
  {
    // Use the CommandCall class to run the CPYF command to copy the records
    // in QIWS/QCUSTCDT to QTEMP/QCUSTCDT
    CommandCall c = new CommandCall(system, "CPYF FROMFILE(QIWS/QCUSTCDT) TOFILE(" + library +
"/QCUSTCDTKY) MBROPT(*REPLACE)");
    try
    {
      System.out.println("Copying records from QIWS/QCUSTCDT to " + library + "/QCUSTCDTKY...");
      c.run();
      AS400Message[] msgs = c.getMessageList();
      if (!msgs[0].getID().equals("CPC2955"))
      {
        System.out.println("Unable to populate " + library + "/QCUSTCDTKY");
        for (int i = 0; i < msgs.length; i++)
        {
          System.out.println(msgs[i]);
        }
        System.exit(0);
      }
    }
    catch(Exception e)
    {
      System.out.println("Unable to populate " + library + "/QCUSTCDTKY");
      System.exit(0);
    }
  }
}
```

例: UserList を使用して特定のグループ内のすべてのユーザーをリストする

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////
//
// User list example.  This program lists all of the users in a given
// group.
//
// Command syntax:
//    UserListExample system group
//
// This source is an example of IBM Toolbox for Java "UserList".
//
///////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import java.util.Enumeration;

public class UserListExample
{


    public static void main (String[] args)
    {
        // If a system and group were not specified, then display
        // help text and exit.
        if (args.length != 2)
        {
            System.out.println("Usage:  UserListExample system group");
            return;
        }

        try
        {
            // Create an AS400 object.  The system name was passed
            // as the first command line argument.
            AS400 system = new AS400 (args[0]);

            // The group name was passed as the second command line
            // argument.
            String groupName = args[1];
```

```
            // Create the user list object.
            UserList userList = new UserList (system);

            // Get a list of the users in the given group.
            userList.setUserInfo (UserList.MEMBER);
            userList.setGroupInfo (groupName);
            Enumeration enum = userList.getUsers ();

            // Iterate through the list and print out the
            // users' names and descriptions.
            while (enum.hasMoreElements ())
            {
                User u = (User) enum.nextElement ();
                System.out.println ("User name:   " + u.getName ());
                System.out.println ("Description: " + u.getDescription ());
                System.out.println ("");
            }

        }
        catch (Exception e)
        {
           System.out.println ("Error: " + e.getMessage ());
        }

        System.exit (0);
    }


}
```

例: JavaBeans

このセクションでは、 bean のトピックを扱った資料に記載されているコーディング例をリストします。

- 例: リスナーを使用して、システムへの接続あるいは切断時、およびコマンドの実行時に、コメントを印刷する
- 例: アプレットおよび IBM VisualAge for Java を使用して、コマンドを実行するボタンを作成する

以下の特記事項は、IBM Toolbox for Java のすべての例に適用されます。

> コードの特記事項情報
>
> IBM は、お客様に、すべてのプログラミング・コード・サンプルを使用することができる非独占的な使用権を許諾します。お客様は、このサンプル・コードから、お客様独自の特別のニーズに合わせた類似のプログラムを作成することができます。
>
> すべてのサンプル・コードは、例として示す目的でのみ、IBM により提供されます。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。したがって 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。
>
> ここに含まれるすべてのプログラムは、現存するままの状態で提供され、いかなる保証条件も適用されません。第三者の権利の不侵害、商品性、特定目的適合性に関する黙示の保証の適用も一切ありません。

例: IBM Toolbox for Java bean コード

以下の例では、AS400 オブジェクトと CommandCall オブジェクトを作成してから、 それらのオブジェクトにリスナーを登録します。 それらのオブジェクトのリスナーは、サーバーへの接続あるいは切断時、 および CommandCall オブジェクトがコマンドの実行を完了したときに、コメントを印刷します。

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////
//
// Beans example.  This program uses the JavaBeans support in the
// IBM Toolbox for Java classes.
//
// Command syntax:
//    BeanExample
//
///////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.AS400;
import com.ibm.as400.access.CommandCall;
import com.ibm.as400.access.ConnectionListener;
import com.ibm.as400.access.ConnectionEvent;
import com.ibm.as400.access.ActionCompletedListener;
import com.ibm.as400.access.ActionCompletedEvent;

class BeanExample
{
    AS400       as400_  = new AS400();
    CommandCall cmd_    = new CommandCall( as400_ );

    BeanExample()
    {
        // Whenever the system is connected or disconnected print a
        // comment. Do this by adding a listener to the AS400 object.
        // When a system is connected or disconnected, the AS400 object
        // will call this code.

        as400_.addConnectionListener
        (new ConnectionListener()
          {
            public void connected(ConnectionEvent event)
            {
                System.out.println( "System connected." );
            }
            public void disconnected(ConnectionEvent event)
            {
                System.out.println( "System disconnected." );
```

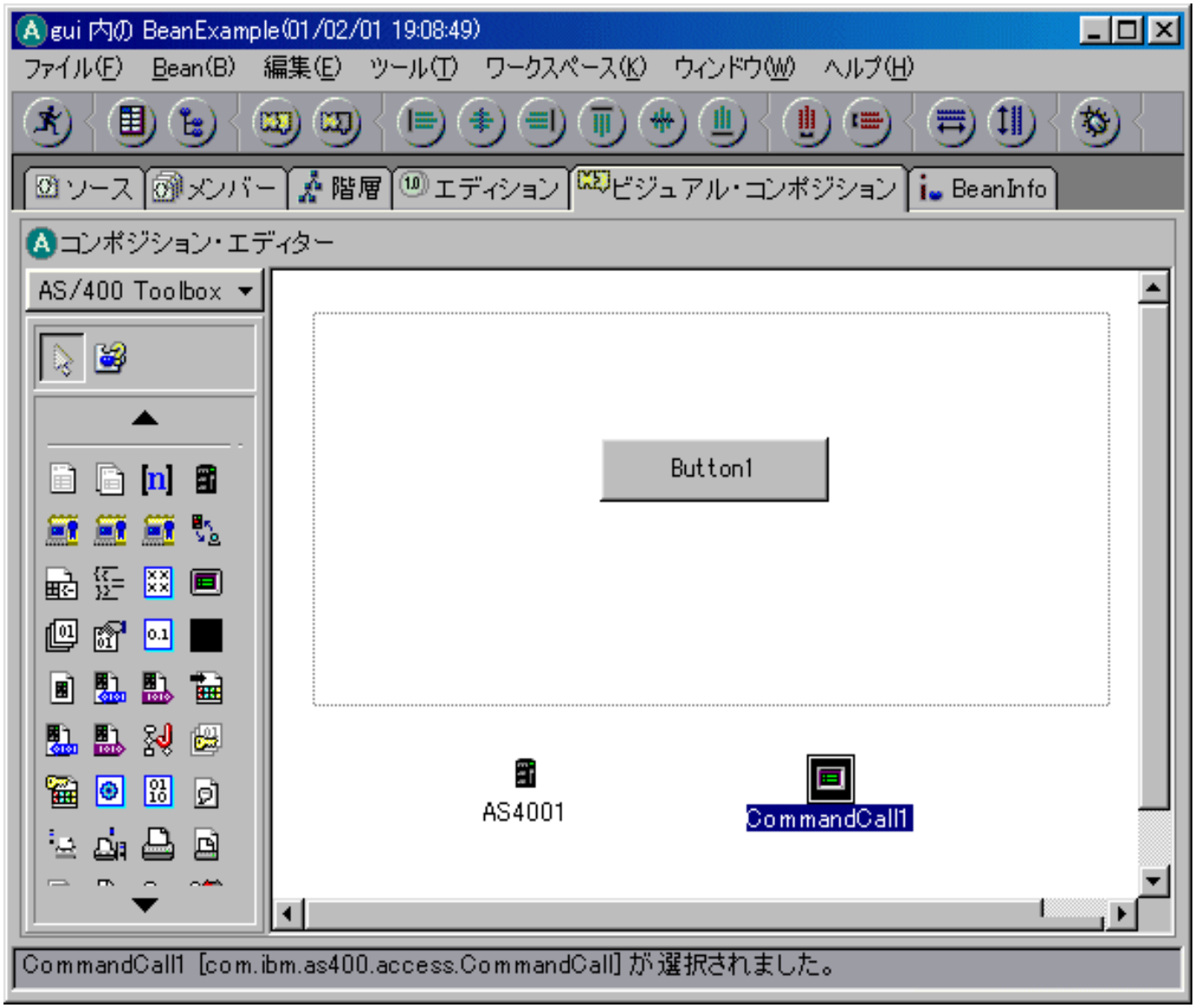
```
            }
        }
    );

    // Whenever a command runs to completion print a comment. Do this
    // by adding a listener to the commandCall object. The commandCall
    // object will call this code when it runs a command.

    cmd_.addActionCompletedListener(
        new ActionCompletedListener()
        {
            public void actionCompleted(ActionCompletedEvent event)
            {
                System.out.println( "Command completed." );
            }
        }
    );
}

void runCommand()
{
    try
    {
        // Run a command. The listeners will print comments when the
        // system is connected and when the command has run to
        // completion.
        cmd_.run( "TESTCMD PARMS" );
    }
    catch (Exception ex)
    {
        System.out.println( ex );
    }
}

public static void main(String[] parameters)
{
   BeanExample be = new BeanExample();

   be.runCommand();

   System.exit(0);
}
}
```

visual bean ビルダーのコード例

この例では IBM VisualAge for Java エンタープライズ版 V3.5 コンポジション・エディターを使用していますが、 他の visual bean ビルダーもこれと類似しています。 この例では、 押されると iSeries または AS/400e サーバー上でコマンドを実行するボタンのアプレットを作成します。

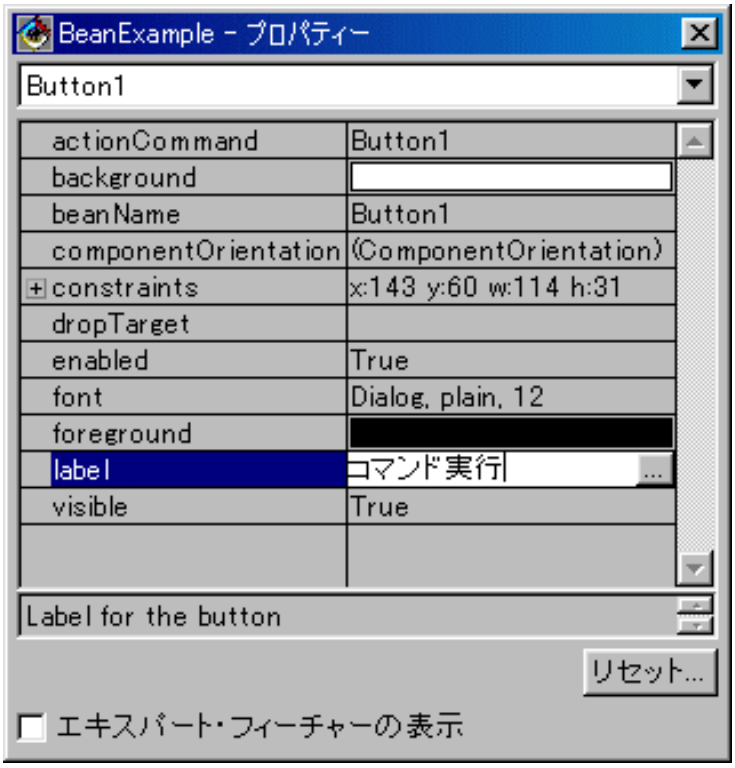
- ボタンをそのアプレットにドラッグ・アンド・ドロップします。(このボタンは、図 1 に示されている bean ビルダーの「ビジュアル・コンポジション」タブの左にあります。)
- アプレットの外に CommandCall bean と AS400 bean をドロップします。(この Bean は、図 1 に示されている bean ビルダーの「ビジュアル・コンポジション」タブの左にあります。)

図 1: 「VisualAge ビジュアル・コンポジション・エディター」ウィンドウ - gui.BeanExample



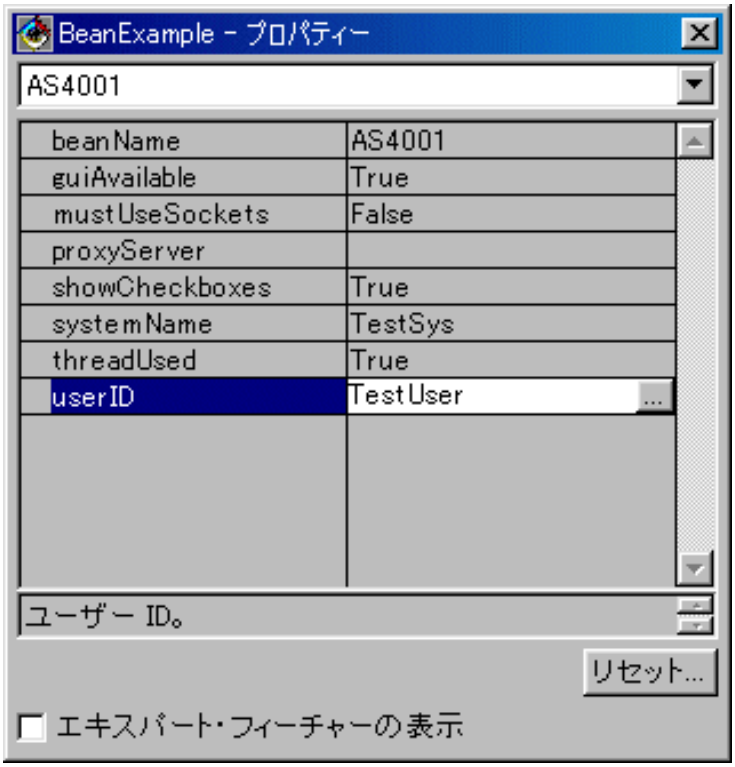
- bean のプロパティーを編集する。(編集する際には、bean を選択してから右クリックし、「プロパティー」オプションのあるポップアップ・ウィンドウを表示します。)
  - ボタンのラベルを図 2 に示されているように「コマンド実行」に変更します。

    図 2: ボタン・ラベルの「コマンド実行」への変更



- AS400 bean のシステム名を「TestSys」に変更します。
- AS400 bean のユーザー ID を図 3 に示されているように「TestUser」に変更します。

図 3: ユーザー ID の名前の「TestUser」への変更



- CommandCall bean のコマンドを図 4 に示されているように、 SNDMSG MSG('Testing') TOUSR('TESTUSER') に変更します。

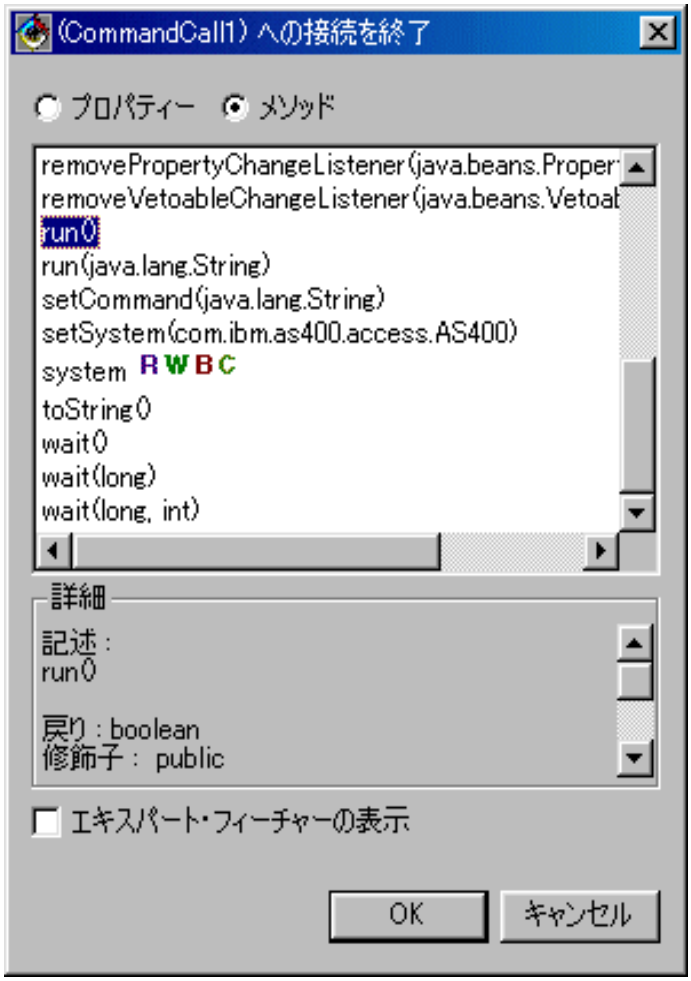
図 4: CommandCall bean のコマンドの変更

- AS400 bean を CommandCall bean に組み合わせます。 このときに使用するメソッドは、各 bean ビルダー間で異なります。 この例では、以下を行います。
  - CommandCall bean を選択してから右マウス・ボタンをクリックする。
  - 「接続」を選択する。
  - 「接続可能なフィーチャー」を選択する。
  - 図 5 に示されているように、機能のリストから「system」を選択する。
  - AS400 bean を選択する。
  - その AS400 bean 上に表示されるポップアップ・メニューから「this」を選択する。

図 5: CommandCall bean への AS400 bean の接続

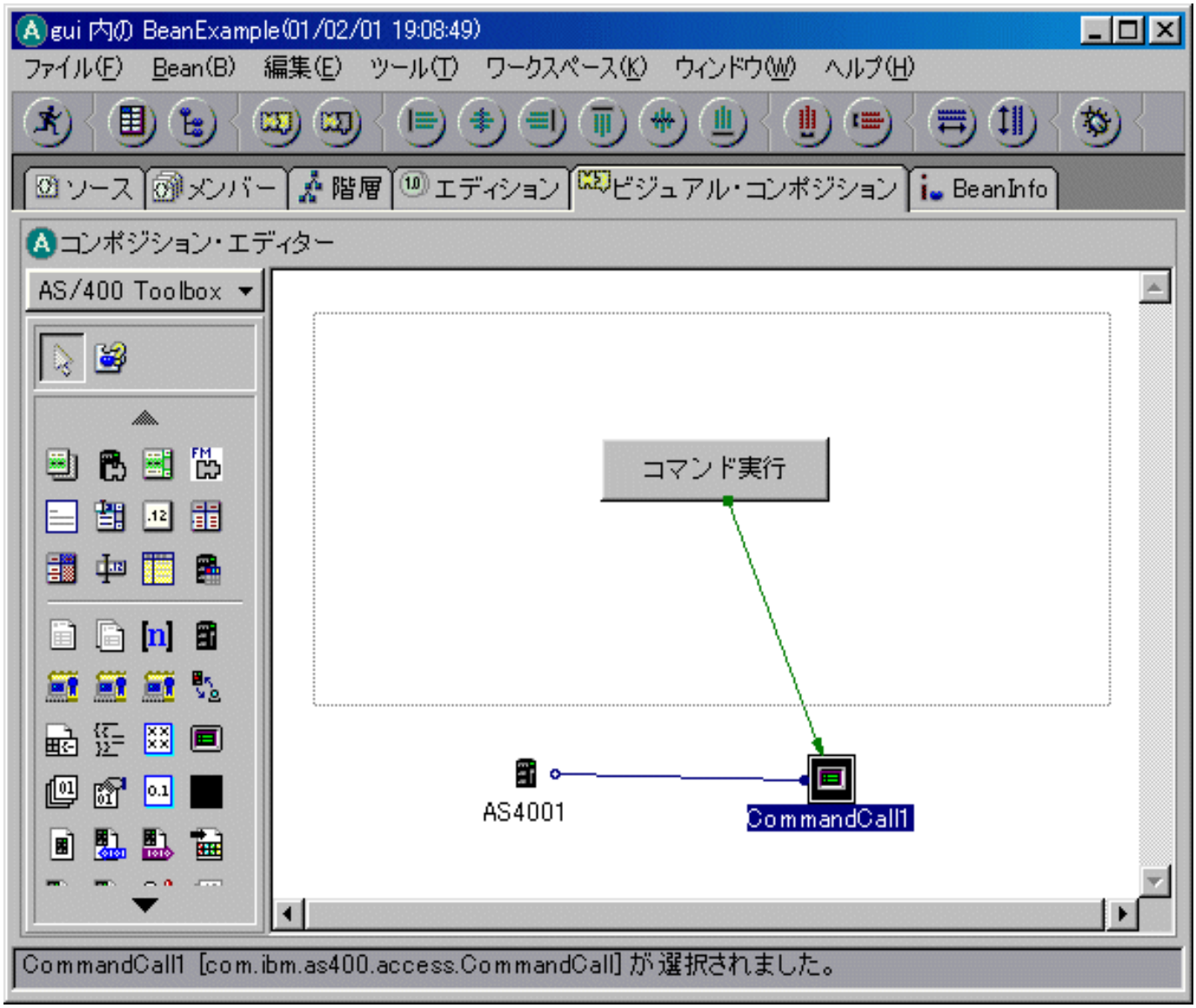
- ボタンを CommandCall bean に接続します。
  - Button bean を選択してから右マウス・ボタンをクリックする。
  - 「接続」を選択する。
  - 「actionPerformed」を選択する。
  - CommandCall bean を選択する。
  - 表示されるポップアップ・メニューから「接続可能なフィーチャー」を選択する。
  - 図 6 に示されているように、メソッドのリストで「run」を選択します。

    図 6: ボタンへのメソッドの接続



上記の実行後、「VisualAge ビジュアル・コンポジション・エディター」ウィンドウは図 7 のようになります。

図 7: 「VisualAge ビジュアル・コンポジション・エディター」ウィンドウ - bean の例の完了


gui 内の BeanExample(01/02/01 19:08:49)
ファイル(F) Bean(B) 編集(E) ツール(T) ワークスペース(K) ウィンドウ(W) ヘルプ(H)
ソース
メンバー
階層
エディション
ビジュアル・コンポジション
BeanInfo
コンポジション・エディター
AS/400 Toolbox
コマンド実行
AS4001
CommandCall1
CommandCall1 [com.ibm.as400.access.CommandCall] が選択されました。

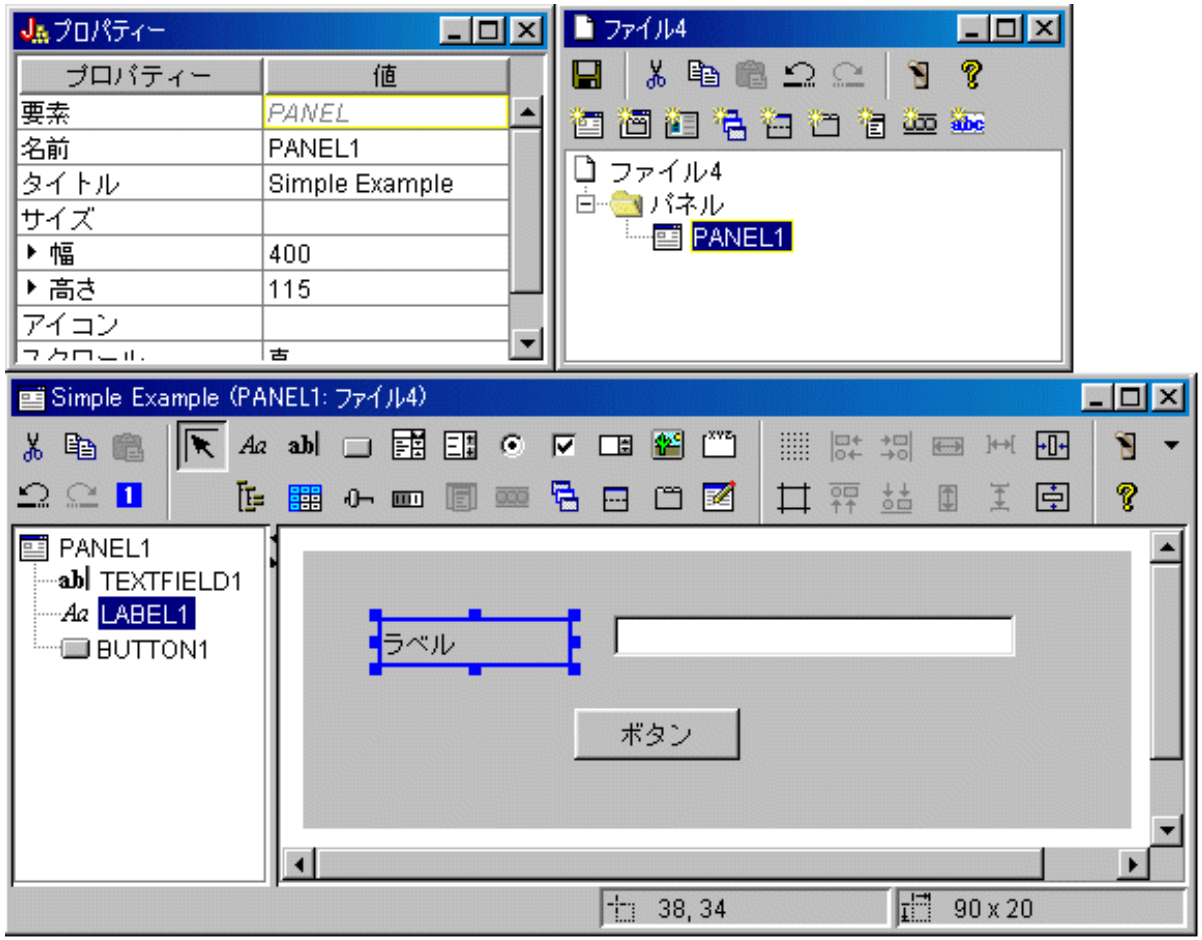
例: GUI ビルダーでパネルを組み立てる

この例では、簡単なパネルを作成することにより、Graphical Toolbox の使用方法を示します。 これは Graphical Toolbox 環境の基本機能および操作を示す概説です。 この例では、パネルの作成方法を示した後、 そのパネルを表示する小さな Java アプリケーションの作成方法を示します。 この例では、ユーザーがデータをテキスト・フィールドに入力して、「閉じる」ボタンをクリックします。 アプリケーションはその後、 そのデータを Java コンソールにエコー表示します。

パネルの構成

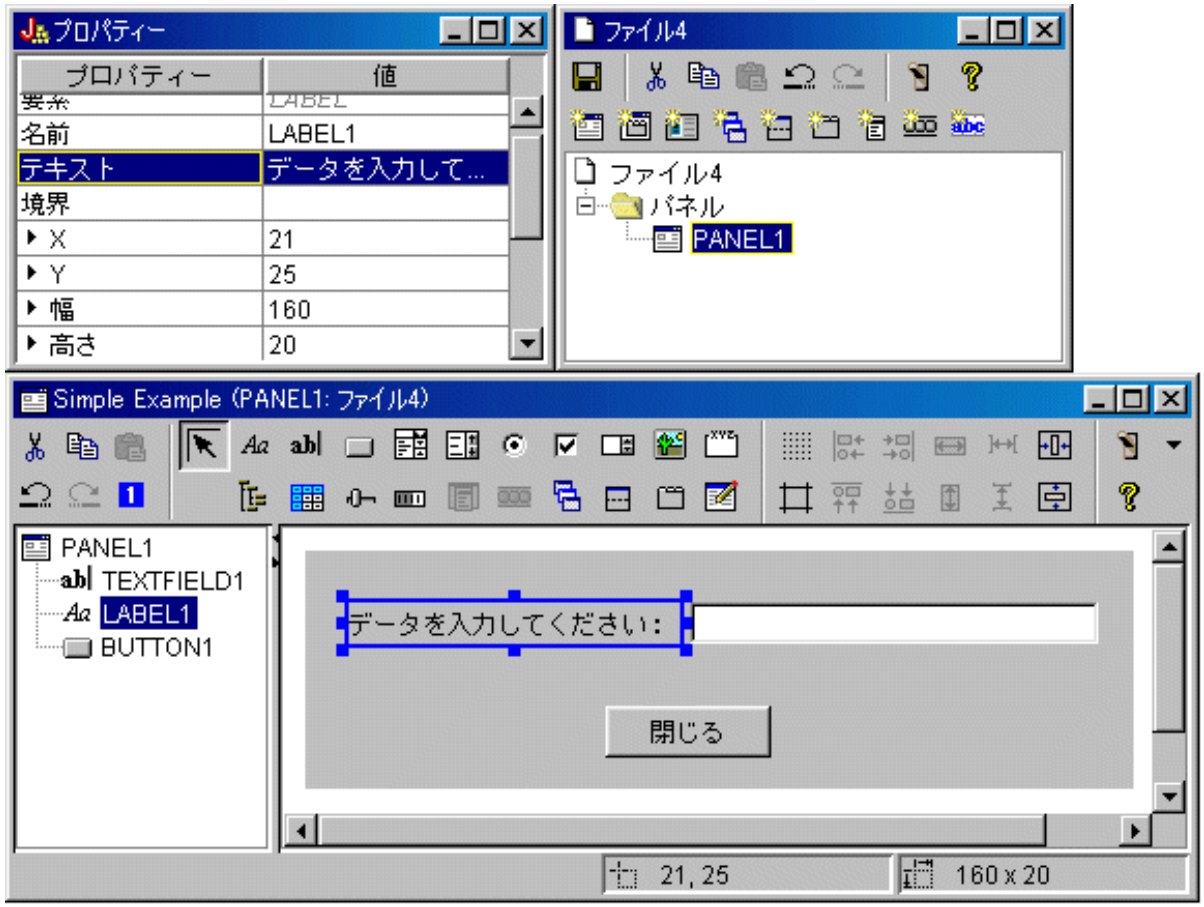
GUI ビルダーを起動すると、「プロパティー」ウィンドウおよび GUI ビルダーのウィンドウが表示されます。 "MyGUI.pdml" という名前で新しいファイルを作成してください。 この例では、新規パネルを挿入します。 「ファイル・ビルダー」ウィンドウ内の「パネルの挿入」アイコンをクリックします。 その名前は「PANEL1」です。 「プロパティー」ウィンドウ内の情報を変更してタイトルを変更します。 「タイトル」フィールドに「Simple Example」と入力します。 3 つのデフォルトのボタンをマウスで選択して「削除」を押すことにより、 それらのボタンを除去します。 「パネル・ビルダー」ウィンドウ内のボタンを使用して、 図 1 に示す 3 つの要素 (ラベル、テキスト・フィールド、およびプッシュボタン) を追加します。

図 1: GUI ビルダーのウィンドウ: パネルの作成を開始する



ラベルを選択すると、そのテキストを「プロパティー」ウィンドウ内で変更できます。 この例では、プッシュボタンにも同じことが行われ、プッシュボタンのテキストが「閉じる」に変更されています。

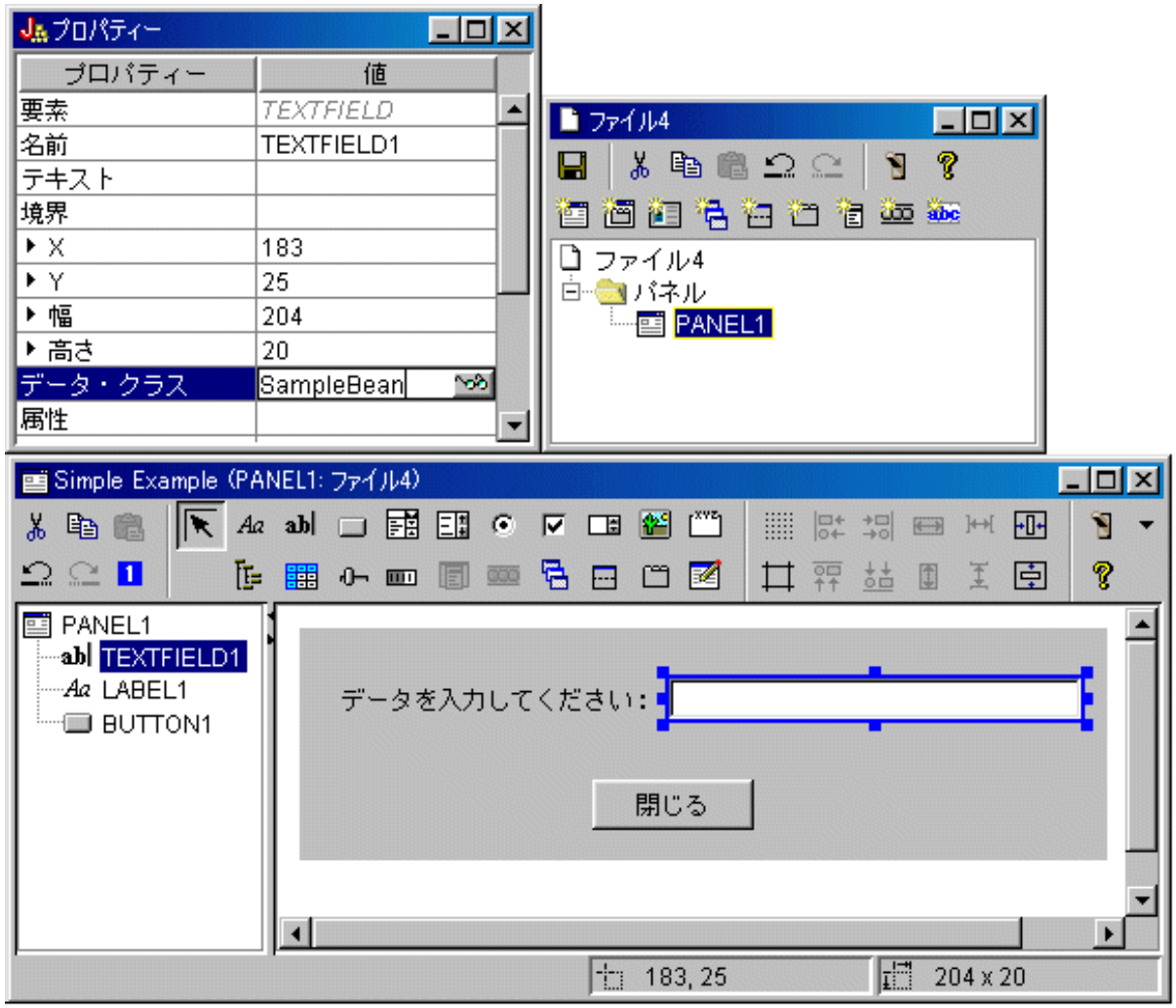
図 2: GUI ビルダーのウィンドウ: 「プロパティー」ウィンドウでテキストを変更する



テキスト・フィールド

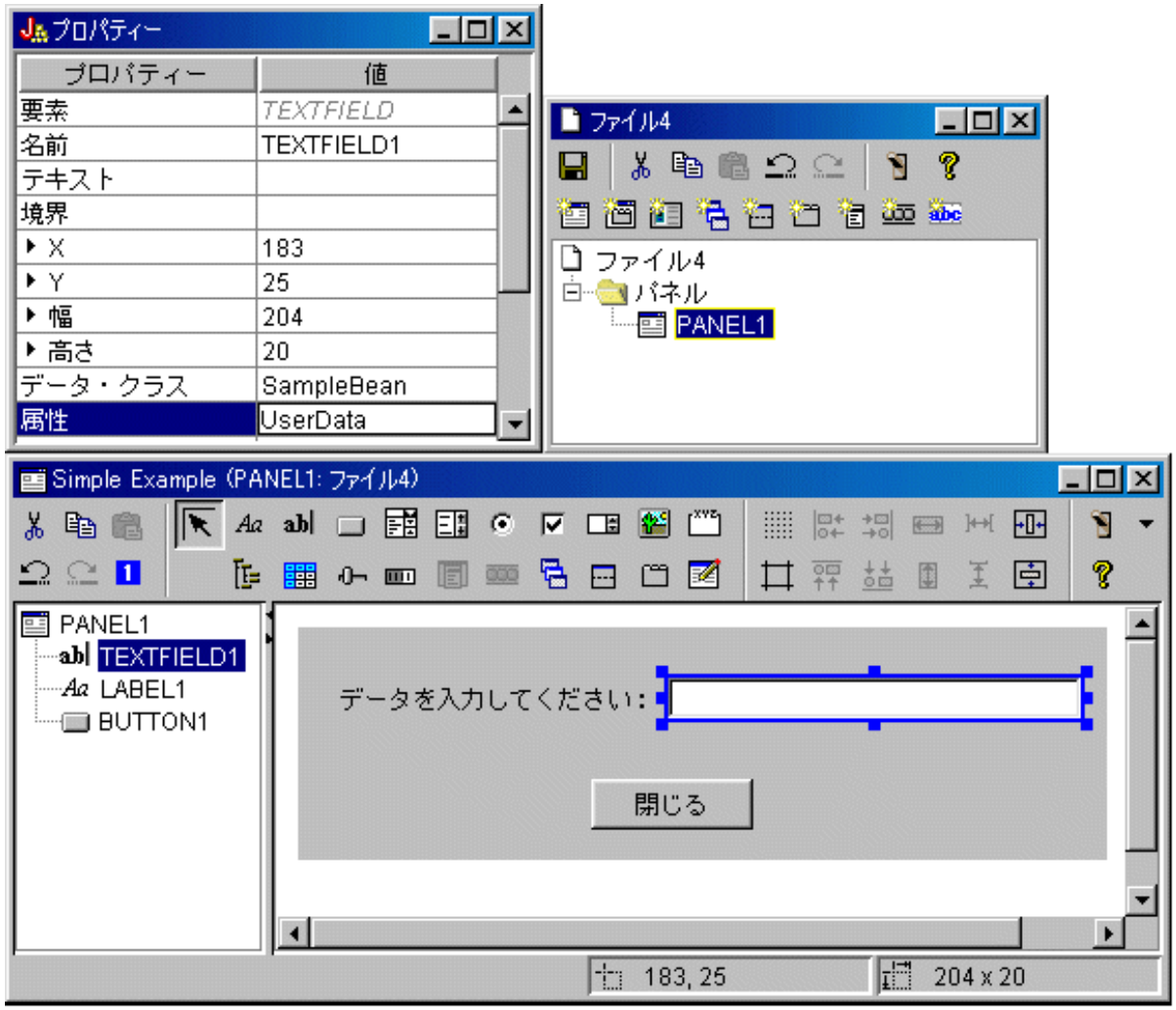
テキスト・フィールドにはデータが含まれるので、 GUI ビルダーが何種類かの追加の作業を実行するためのプロパティーを設定できます。 この例では、データ・クラス・プロパティーを bean クラスの名前である SampleBean に設定します。 この databean は、このテキスト・フィールドにデータを供給します。

図 3: GUI ビルダーのウィンドウ: データ・クラス・プロパティーを設定する



属性プロパティーを、 データが入れられる bean プロパティーの名前に設定します。 この例では、その名前は UserData です。

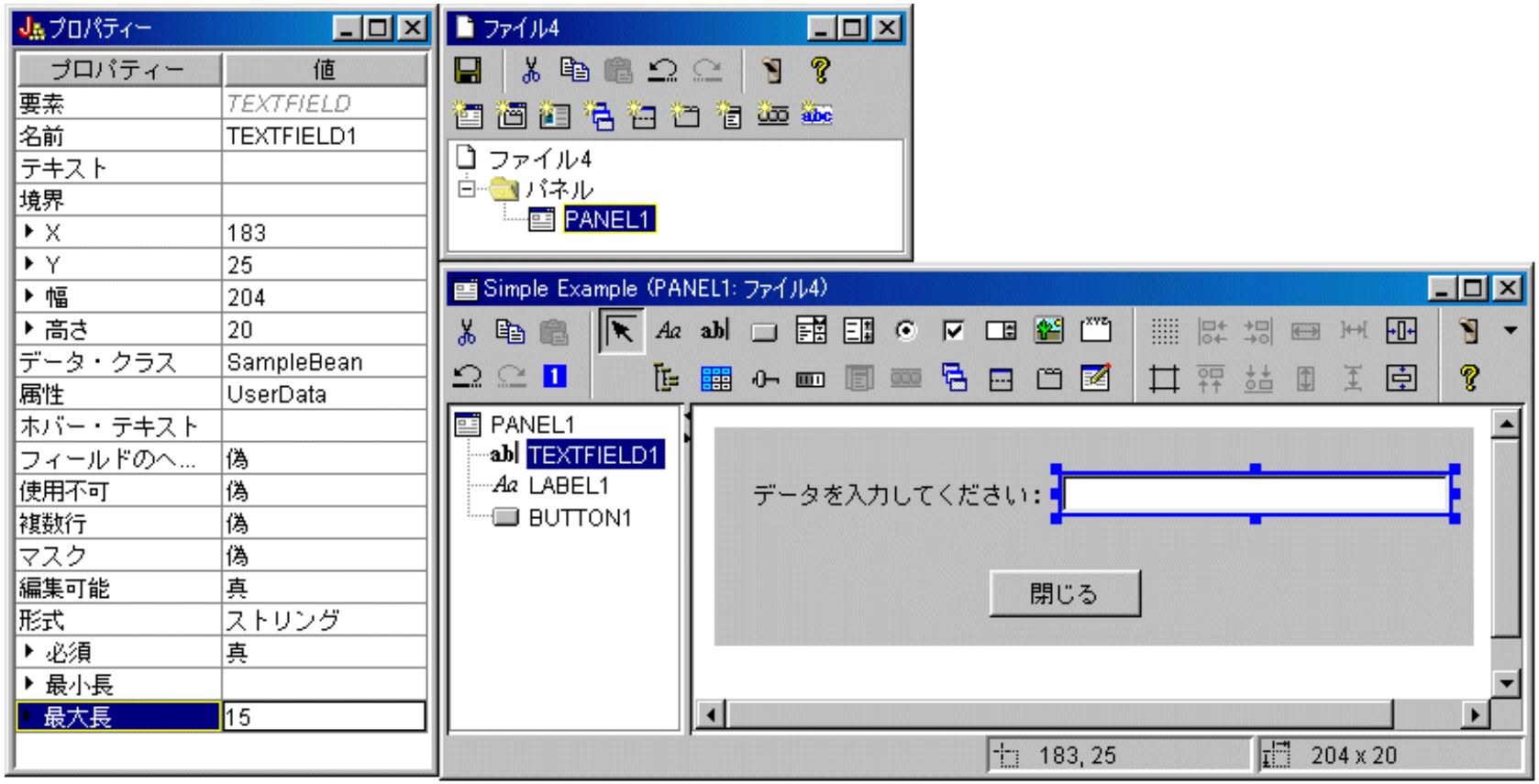
図 4: GUI ビルダーのウィンドウ: 属性プロパティーを設定する



上記のステップを実行すると、 UserData プロパティーがこのテキスト・フィールドにバインドされます。 実行時に Graphical Toolbox は SampleBean.getUserData を呼び出して、 このフィールドの初期値を取得します。 その後、SampleBean.setUserData を呼び出してパネルがクローズするときに、 変更された値がアプリケーションに送り返されます。

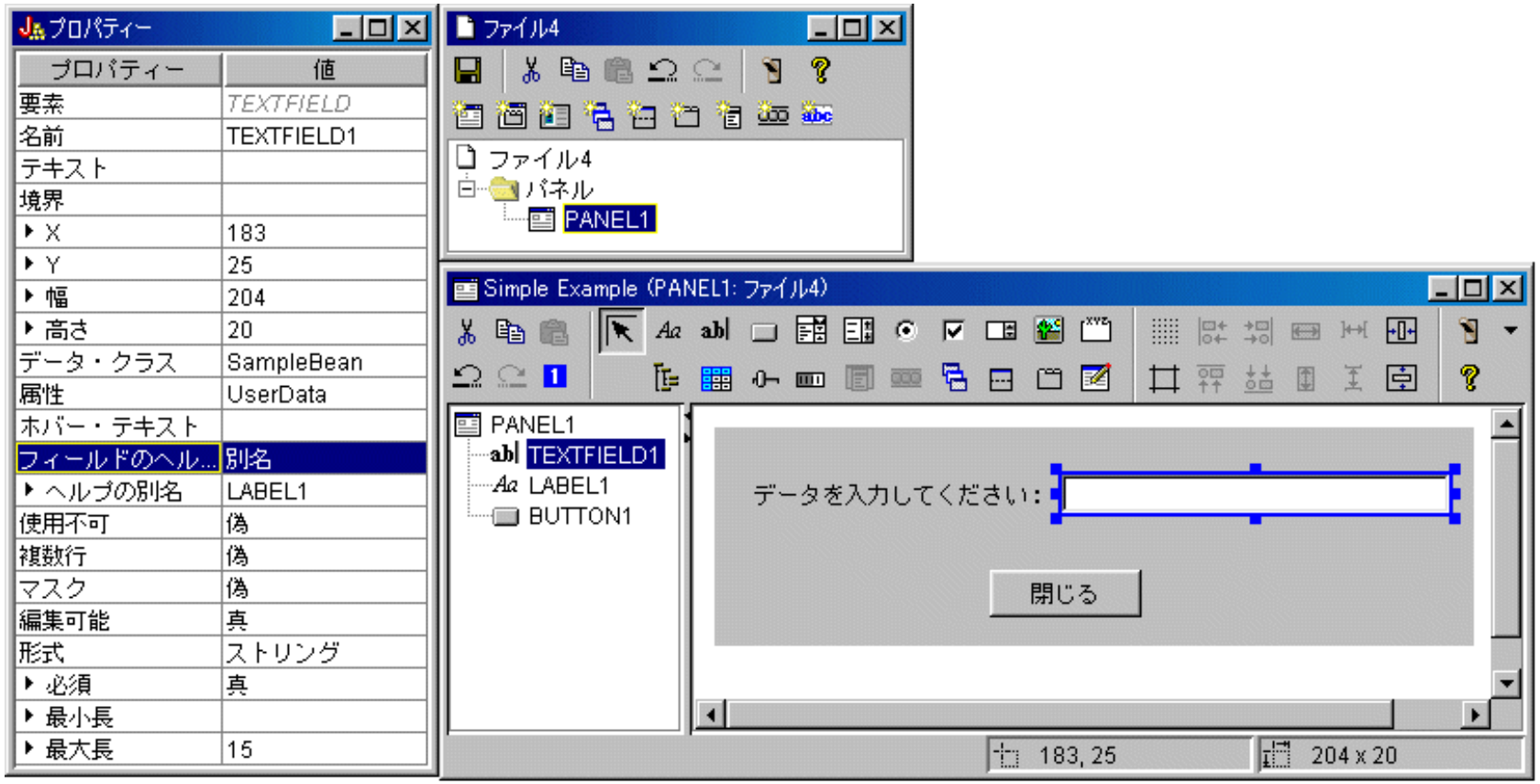
ユーザーによるデータ入力が必要であることや、 そのデータは最大長 15 文字のストリングでなければならないことを指定しています。

図 5: GUI ビルダーのウィンドウ: テキスト・フィールドの最大長を設定する



テキスト・フィールドのコンテキストに依存したヘルプ・トピックが、 ラベル「データを入力してください」と関連付けられていることを指定します。

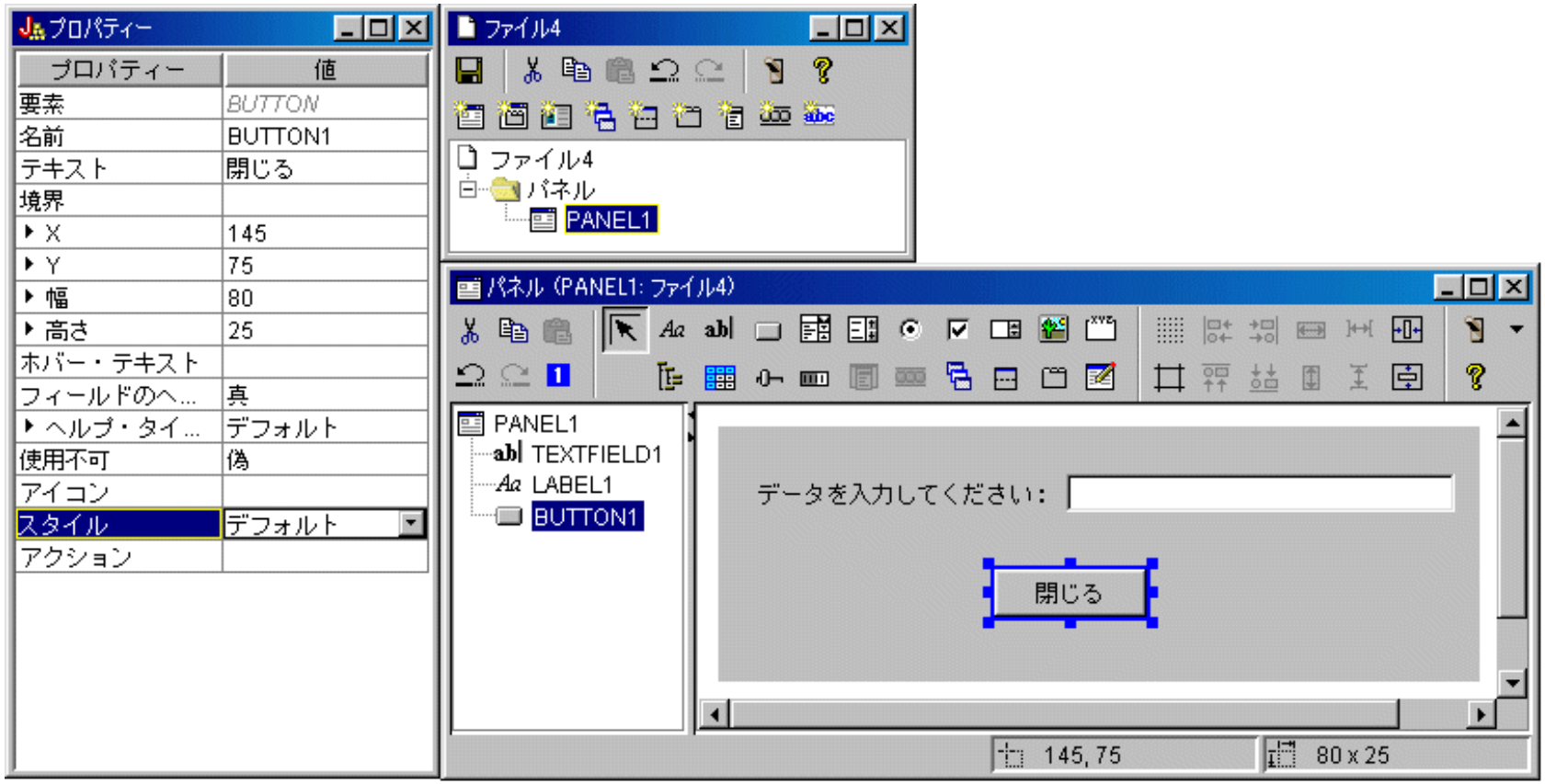
図 6: GUI ビルダーのウィンドウ: テキスト・フィールドにコンテキストに依存したヘルプを設定する



ボタン

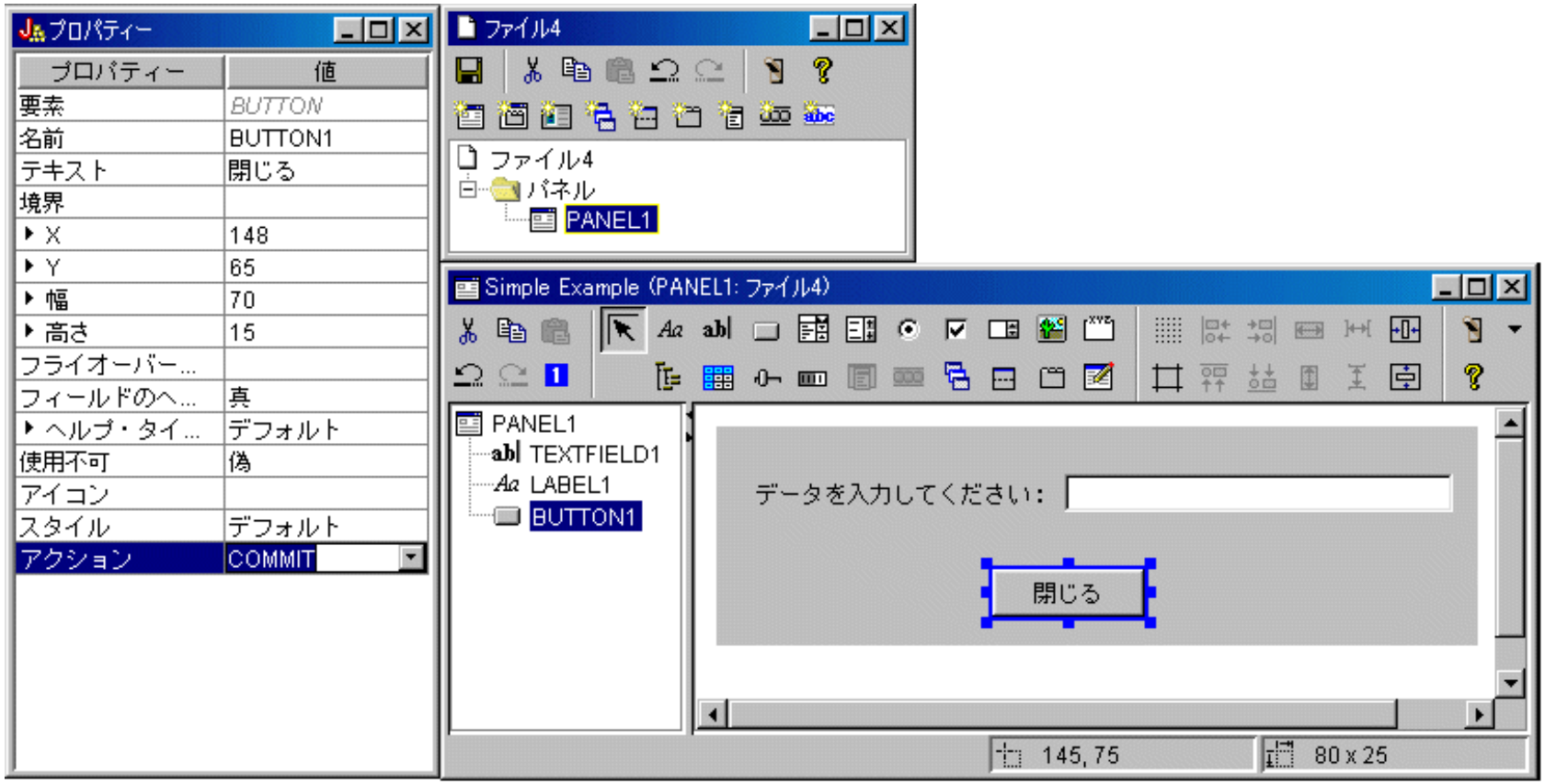
スタイル・プロパティーを変更して、ボタンのデフォルト強調を設定します。

図 7: GUI ビルダーのウィンドウ: スタイル・プロパティーを設定してボタンをデフォルト強調する



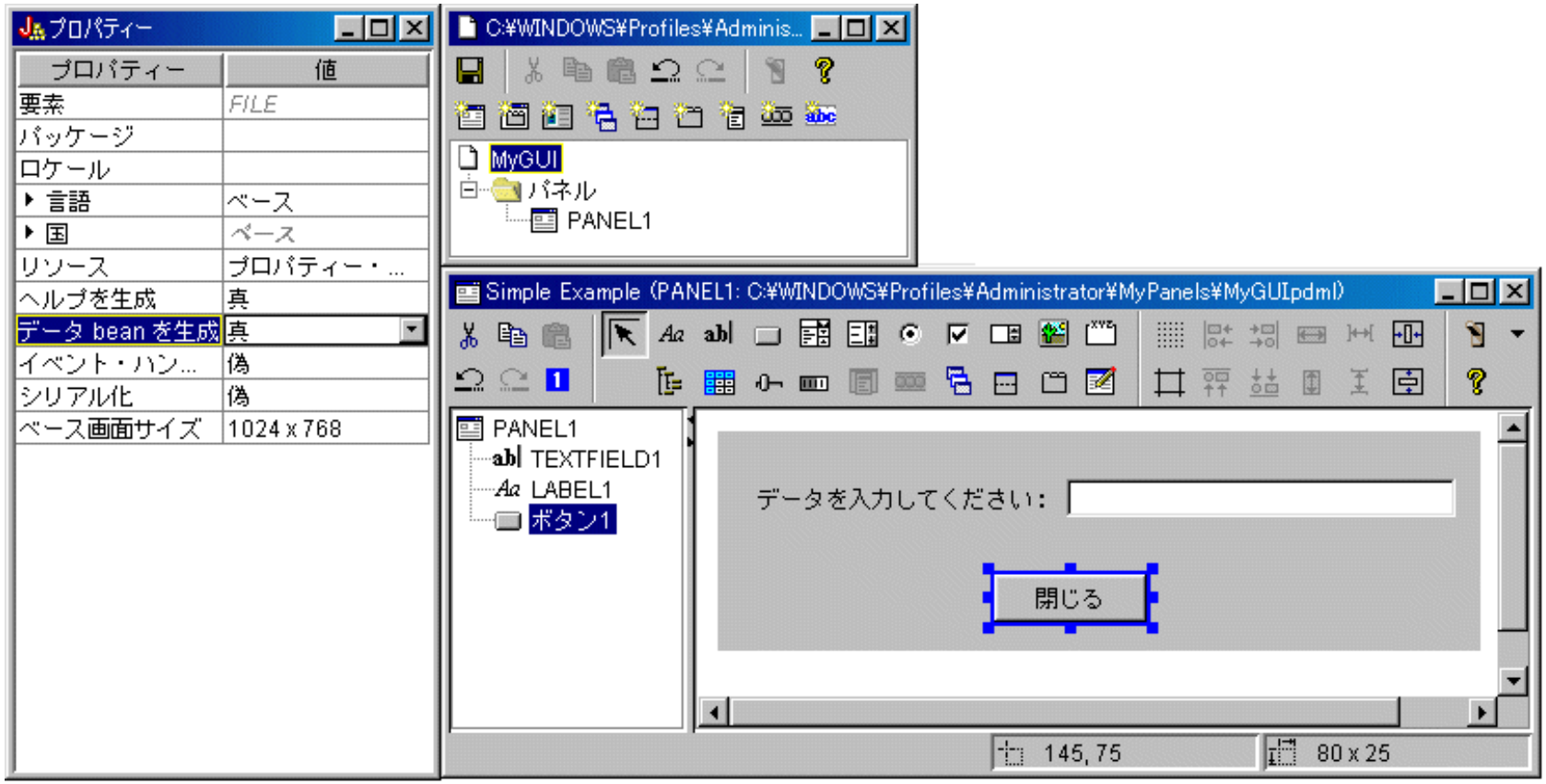
アクション・プロパティーを COMMIT に設定します。 この設定により、ボタンを選択すると bean の setUserData メソッドが呼び出されます。

図 8: GUI ビルダーのウィンドウ: アクション・プロパティーを COMMIT に設定する



パネルを保管する前に、PDML ファイルのレベルでプロパティーを設定して、 オンライン・ヘルプの骨組みと Java bean を生成します。 次に、「GUI ビルダー」ウィンドウ内の アイコンをクリックしてファイルを保管します。 プロンプトが表示されたら、ファイル名 MyGUI.pdml を指定します。

図 9: GUI ビルダーのウィンドウ: プロパティーを設定してオンライン・ヘルプの骨組みと Java bean を生成する



生成されるファイル

パネル定義を保管した後で、GUI ビルダーによって生成されたファイルを見ることができます。 PDML ファイル パネル定義マークアップ言語の動作方法が分かるように、 MyGUI.pdml の内容をここに示します。 PDML の処理はすべて Graphical Toolbox のツールを使用して行うので、 このファイルの形式を詳しく知っておく必要があります。

```
<!-- Generated by GUI Builder -->
<PDML version="2.0" source="JAVA" basescreensize="1280x1024">
```

```
 <PANEL name="PANEL1">
  <TITLE>PANEL1</TITLE>
  <SIZE>351,162</SIZE>
  <LABEL name="LABEL1"">
   <TITLE>PANEL1.LABEL1</TITLE>
   <LOCATION>18,36</LOCATION>
   <SIZE>94,18</SIZE>
   <HELPLINK>PANEL1.LABEL1</HELPLINK>
  </LABEL>
  <TEXTFIELD name="TEXTFIELD1">
   <TITLE>PANEL1.TEXTFIELD1</TITLE>
   <LOCATION>125,31</LOCATION>
   <SIZE>191,26</SIZE>
   <DATACLASS>SampleBean</DATACLASS>
   <ATTRIBUTE>UserData</ATTRIBUTE>
   <STRING minlength="0" maxlength="15"/>
   <HELPALIAS>LABEL1</HELPALIAS>
  </TEXTFIELD>
  <BUTTON name="BUTTON1">
   <TITLE>PANEL1.BUTTON1</TITLE>
   <LOCATION>125,100</LOCATION>
   <SIZE>100,26</SIZE>
   <STYLE>DEFAULT</STYLE>
   <ACTION>COMMIT</ACTION>
   <HELPLINK>PANEL1.BUTTON1</HELPLINK>
  </BUTTON>
 </PANEL>

</PDML>
```

リソース・バンドル

すべての PDML ファイルには、リソース・バンドルが関連付けられています。 この例では、MyGUI.properties と呼ばれる変換可能リソースが PROPERTIES ファイルに保管されています。 PROPERTIES ファイルには、 GUI ビルダーのカスタマイズ・データも含まれていることに注意してください。

```
##Generated by GUI Builder
BUTTON_1=Close
TEXT_1=
@GenerateHelp=1
@Serialize=0
@GenerateBeans=1
LABEL_1=Enter some data:
```

```
PANEL_1.Margins=18,18,18,18,18,18
PANEL_1=Simple Example
```

JavaBean

この例では、JavaBean オブジェクトの Java ソース・コードの骨組みも生成されます。 SampleBean.java の内容をここに示します。

```
import com.ibm.as400.ui.framework.java.*;

public class SampleBean extends Object
    implements DataBean
{
    private String m_sUserData;

    public String getUserData()
    {
        return m_sUserData;
    }

    public void setUserData(String s)
    {
        m_sUserData = s;
    }

    public Capabilities getCapabilities()
    {
        return null;
    }

    public void verifyChanges()
    {
    }

    public void save()
    {
    }

    public void load()
    {
        m_sUserData = "";
    }
}
```

UserData プロパティーの getter メソッドと setter メソッドが骨組みにすでに実装されていることに注意してください。 他のメソッドは DataBean インターフェースによって定義されるもので、必須です。

GUI ビルダーにより骨組みの Java コンパイラーがすでに起動されており、 対応するクラス・ファイルがすでに作成されています。 これは簡単

な例なので、実装されている bean に修正を加える必要はありません。 実際の Java アプリケーションでは、 通常は load メソッドと save メソッドに修正を加えて、 外部データ・ソースからデータが転送されるようにします。 他の 2 つのメソッドは、多くの場合デフォルトで実装されるもので十分です。 詳細については、 PDML 実行時フレームワーク用の javadoc で DataBean インターフェースに関する資料を参照してください。

ヘルプ・ファイル

GUI ビルダーは、ヘルプ文書と呼ばれる HTML フレームワークも作成します。 ヘルプの作成者は、このファイルを編集することによってヘルプ情報を簡単に管理できます。 詳細は、以下のトピックを参照してください。

- ヘルプ文書の作成
- GUI ビルダーによって生成されたヘルプ文書の編集

アプリケーションの構築

パネル定義および生成済みファイルを保管したら、 アプリケーションを構築することができます。 必要なものは、アプリケーションのメイン・エントリー・ポイントを含む新しい Java ソース・ファイルだけです。 この例では、このファイルは SampleApplication.java です。 このファイルには以下のコードが含まれます。

```
import com.ibm.as400.ui.framework.java.*;
import java.awt.Frame;

public class SampleApplication
{
    public static void main(String[] args)
    {
        // Instantiate the bean object that supplies data to the panel
        SampleBean bean = new SampleBean();

        // Initialize the object
        bean.load();

        // Set up to pass the bean to the panel manager
        DataBean[] beans = { bean };

        // Create the panel manager.Parameters:
        // 1. PDML file as a resource name
        // 2. Name of panel to display
        // 3. List of data objects that supply panel data
        // 4. An AWT Frame to make the panel modal

        PanelManager pm = null;
        try { pm = new PanelManager("MyGUI", "PANEL_1", beans, new Frame()); }
        catch (DisplayManagerException e)
        {
            // Something didn't work, so display a message and exit
```

```
            e.displayUserMessage(null);
            System.exit(1);
        }

        // Display the panel - we give up control here
        pm.setVisible(true);

        // Echo the saved user data
        System.out.println("SAVED USER DATA: '" + bean.getUserData() + "'");

        // Exit the application
        System.exit(0);
    }
}
```

呼び出しプログラムで load が呼び出されることにより、 bean オブジェクトが初期化されます。 パネルのデータが複数の bean オブジェクトによって成り立っている場合は、 個々のオブジェクトを初期化してから Graphical Toolbox 環境に渡さなければなりません。

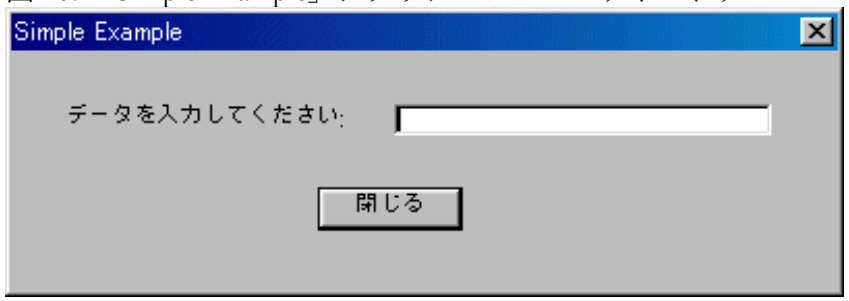
クラス com.ibm.as400.ui.framework.java.PanelManager には、 スタンドアロン型のウィンドウとダイアログを表示するための API が備えられています。 コンストラクターに備えられている PDML ファイルの名前は、 Graphical Toolbox ではリソース名として扱われます。 PDML を含むディレクトリー、ZIP ファイル、または JAR ファイルはクラスパスにより識別されます。

Frame オブジェクトはコンストラクター上に備えられるので、 このウィンドウは順序指定ダイアログとして使用されます。 実際の Java アプリケーションでは、このオブジェクトはダイアログの適切な親ウィンドウから取得されます。 このウィンドウは順序指定形式なので、ウィンドウをクローズするまで制御権はアプリケーションに戻りません。 クローズした時点で、アプリケーションが修正済みのユーザー・データをエコーして、終了します。

アプリケーションの実行

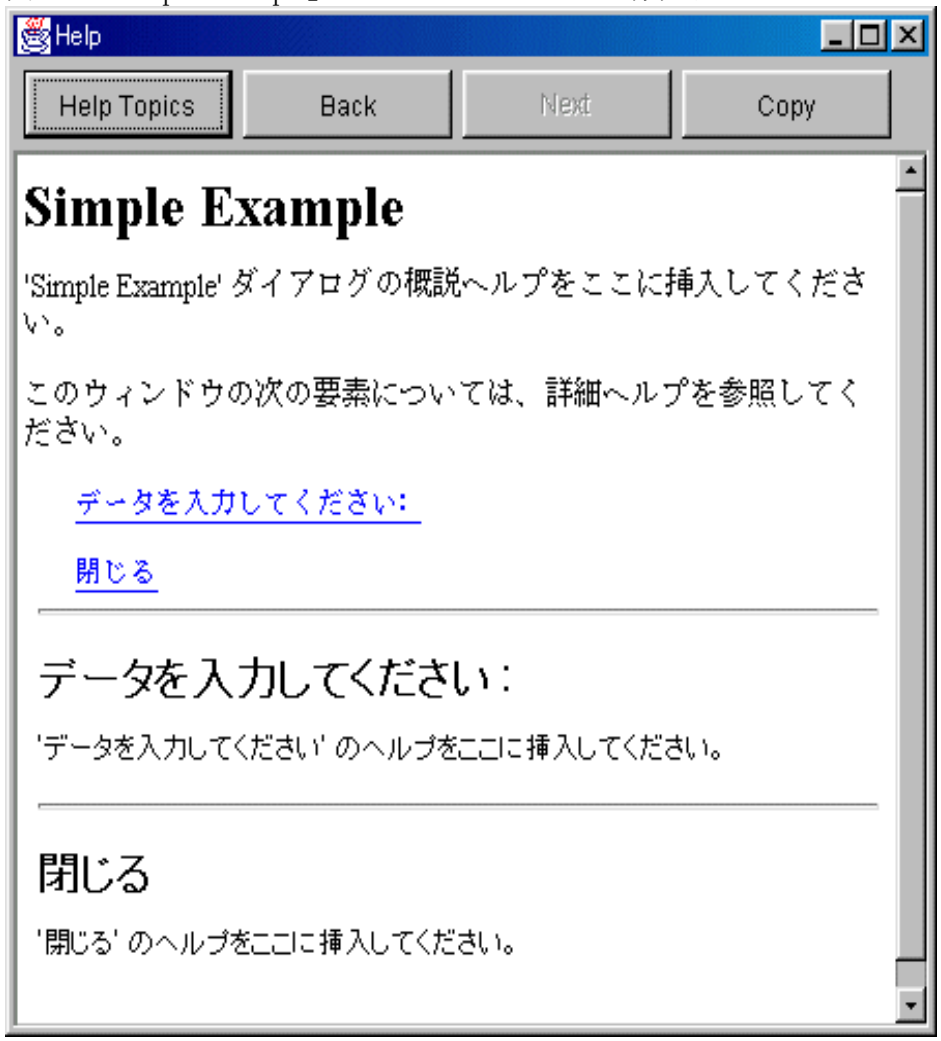
アプリケーションがコンパイルし実行すると、以下のようなウィンドウが表示されます。

図 10: 「Simple Example」アプリケーション・ウィンドウ



フォーカスがテキスト・フィールドにある時点で F1 を押すと、 Graphical Toolbox にヘルプ・ブラウザーが表示され、 そこに GUI ビルダーで生成されたオンライン・ヘルプの骨組みが入ります。

図 11: 「Simple Example」オンライン・ヘルプの骨組み



HTML を編集して、そこのヘルプ・トピックに実際のヘルプ項目を追加できます。

テキスト・フィールド内のデータが無効な場合 (たとえば、値を入力しないで「閉じる」ボタンを押した場合)、 Graphical Toolbox にはエラー・メッセージが表示され、フォーカスがそのフィールドに戻るので、データを入力できます。

図 12: 「データ入力エラー (Data Entry Error)」メッセージ

この例をアプレットとして実行する方法については、ブラウザーで Graphical Toolbox を使用するを参照してください。

編集可能コンボ・ボックス

bean 生成プログラムが編集可能 ComboBox の gettor および settor を生成する際、デフォルトでは、String を settor で戻し、ストリング・パラメーターを gettor で取ります。 settor を変更して Object クラスを取るようにしたり、 gettor が Object タイプを戻すようにすると便利かもしれません。これにより、ChoiceDescriptor を使用してユーザー選択が可能になります。

gettor および settor で Object タイプが検出される場合、システムはフォーマット済みストリングではなくて ChoiceDescriptor または Object タイプを受け取ることを予期します。
例

Editable が 編集可能な ComboBox で、 Double 値を持っているか、システム値を使用しているか、設定されていないと想定しています。

```
public Object getEditable()
{
    if (m_setting == SYSTEMVALUE)
    {
       return new ChoiceDescriptor("choice1","System Value");
    }
    else if (m_setting == NOTSET)
    {
       return new ChoiceDescriptor("choice2","Value not set");
    }
    else
    {
       return m_doubleValue;
    }
}
```

同様に、gettor および settor で Object タイプが検出される場合、システムによって Object が戻されます。この Object は、選択された選択項目が入っている ChoiceDescriptor か Object タイプです。

```
public void setEditable(Object item)
{
    if (ChoiceDescriptor.class.isAssignableForm(obj.getClass()))
    {
       if (((ChoiceDescriptor)obj).getName().equalsIgnoreCase("choice1"))
           m_setting = SYSTEMVALUE;
       else
```

```
            m_setting = NOTSET;
        }
        else if (Double.class.isAssignableFrom(obj.getClass()))
        {
           m_setting = VALUE;
           m_doubleValue = (Double)obj;
        }
        else
        { /* error processing */ }
    }
```

GUI ビルダーによるパネルの作成

GUI ビルダーを使用すれば、パネルを簡単に作成できます。 GUI ビルダーのメイン・ウィンドウのメニュー・バーから、 「ファイル」 --> 「新規ファイル」を選択します。

GUI ビルダーの「ファイル」ウィンドウのメニュー・バーから、 「新規パネルの挿入 (Insert New Panel)」アイコン  をクリックして、 パネル用コンポーネントを挿入できるパネル・ビルダーを表示させます。 「パネル」ウィンドウ上のツールバー・ボタンは、パネルに追加できるさまざまなコンポーネントを表します。 必要なコンポーネントを選択してから、それを配置したい場所をクリックしてください。

以下の画像では、提供されているいくつかのオプションを使って作成されたパネルを示しています。

図 1: GUI ビルダーによるサンプル・パネルの作成

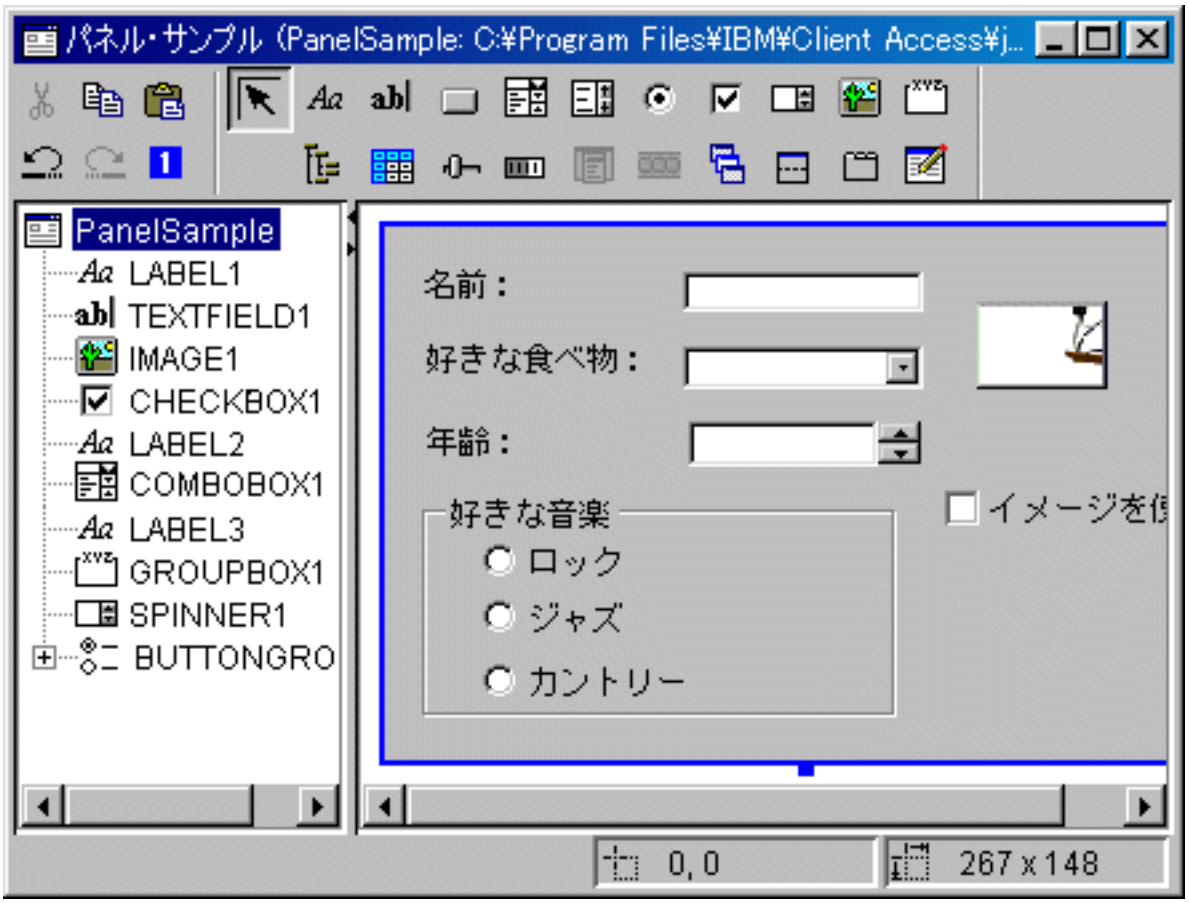


図 1 のサンプル・パネルは、以下のような DataBean コードを使用して、 さまざまなコンポーネントを組み合わせます。

```
import com.ibm.as400.ui.framework.java.*;

public class PanelSampleDataBean extends Object
    implements DataBean
{
    private String m_sName;
    private Object m_oFavoriteFood;
    private ChoiceDescriptor[] m_cdFavoriteFood;
    private Object m_oAge;
    private String m_sFavoriteMusic;

    public String getName()
    {
        return m_sName;
    }

    public void setName(String s)
    {
        m_sName = s;
    }

    public Object getFavoriteFood()
    {
        return m_oFavoriteFood;
    }

    public void setFavoriteFood(Object o)
    {
        m_oFavoriteFood = o;
    }

    public ChoiceDescriptor[] getFavoriteFoodChoices()
    {
        return m_cdFavoriteFood;
    }

    public Object getAge()
    {
        return m_oAge;
    }

    public void setAge(Object o)
```

```
{
    m_oAge = o;
}

public String getFavoriteMusic()
{
    return m_sFavoriteMusic;
}

public void setFavoriteMusic(String s)
{
    m_sFavoriteMusic = s;
}

public Capabilities getCapabilities()
{
    return null;
}

public void verifyChanges()
{
}

public void save()
{
    System.out.println("Name = " + m_sName);
    System.out.println("Favorite Food = " + m_oFavoriteFood);
    System.out.println("Age = " + m_oAge);
    String sMusic = "";
    if (m_sFavoriteMusic != null)
    {
        if (m_sFavoriteMusic.equals("RADIOBUTTON1"))
            sMusic = "Rock";
        else if (m_sFavoriteMusic.equals("RADIOBUTTON2"))
            sMusic = "Jazz";
        else if (m_sFavoriteMusic.equals("RADIOBUTTON3"))
            sMusic = "Country";
    }
    System.out.println("Favorite Music = " + sMusic);
}

public void load()
```

```
    {
        m_sName = "Sample Name";
        m_oFavoriteFood = null;
        m_cdFavoriteFood = new ChoiceDescriptor[0];
        m_oAge = new Integer(50);
        m_sFavoriteMusic = "RADIOBUTTON1";
    }
}
```

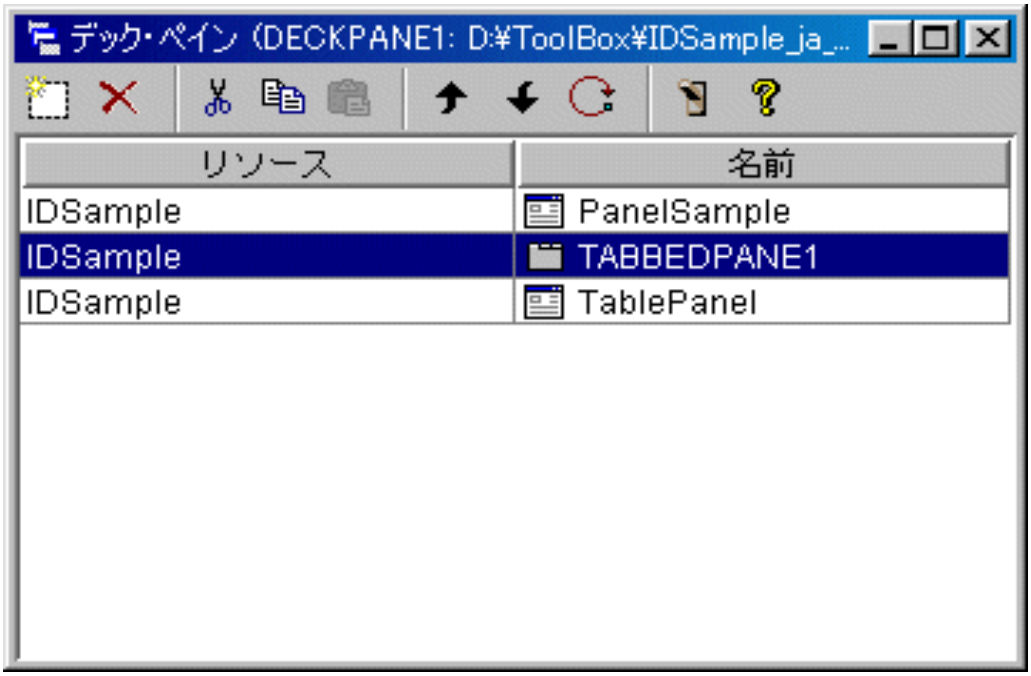
パネルは GUI ビルダーで提供されているコンポーネントの中で最も単純なものですが、 単純なパネルから、大規模な UI アプリケーションを構築できます。

GUI ビルダーを使用してデック・ペインを作成する

GUI ビルダーを使用すると、デック・ペインを簡単に作成することができます。「GUI ビルダー」ウィンドウのメニュー・バーから、「ファイル」 --> 「新規ファイル」を選択します。

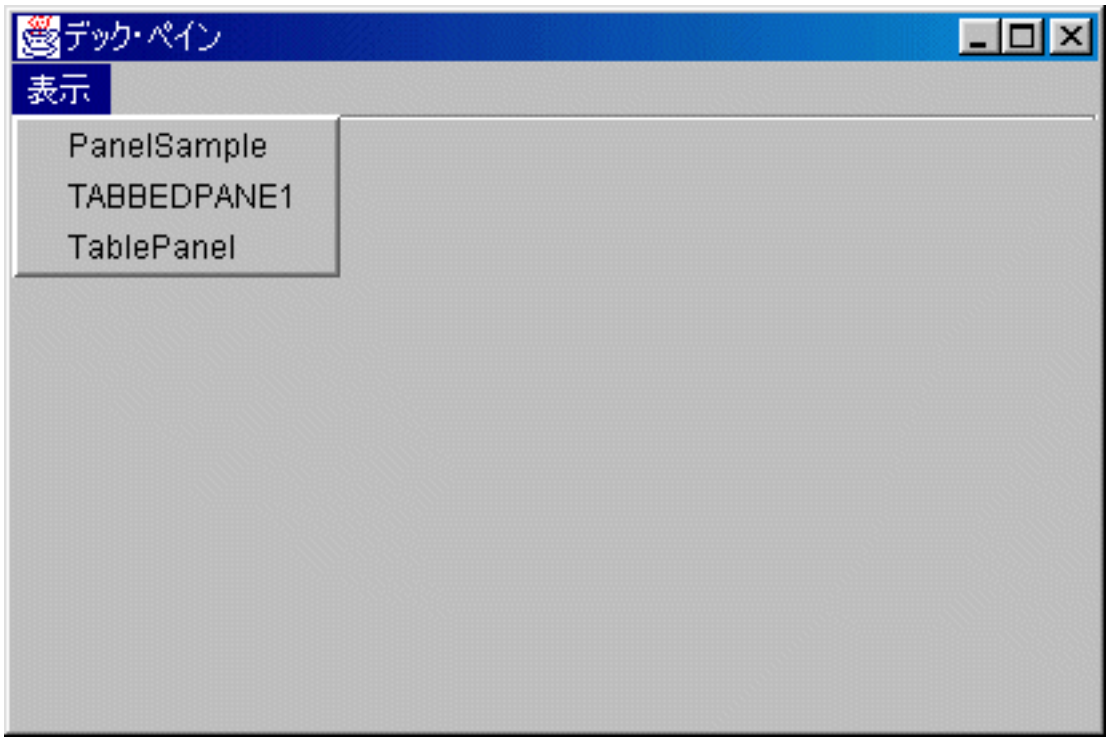
GUI ビルダーの「ファイル」ウィンドウのメニュー・バーから、「デック・ペインの挿入 (Insert Deck Pane)」ツール・ボタンをクリックし、デック・ペイン用のコンポーネントを挿入できるパネル・ビルダーを表示します。以下の例では、3 つのコンポーネントが追加されます。

図 1: GUI ビルダーを使用してデック・ペインを作成する



デック・ペインを作成した後、「プレビュー (Preview)」ツール・ボタンをクリックして、プレビューします。デック・ペインは、「表示」メニューを選択するまでは何もないように見えます。

図 2: GUI ビルダーを使用してデック・ペインをプレビューする

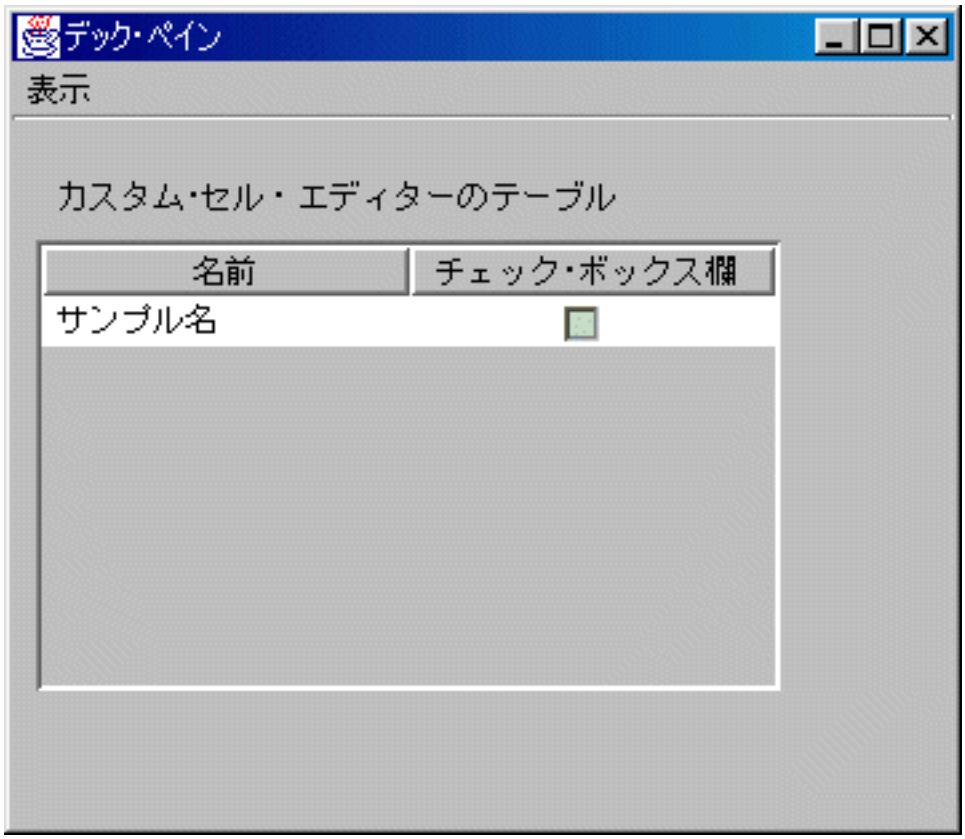


デック・ペインの「表示」メニューから、表示するコンポーネントを選択します。この例では、PanelSample、TABBEDPANE1、または TablePanel を表示するようにできます。以下の図は、これらのコンポーネントを表示する際に見えるものを示しています。

図 3: GUI ビルダーを使用して PanelSample を表示する

図 4: GUI ビルダーを使用して TABBEDPANE1 を表示する

図 5: GUI ビルダーを使用して TablePanel を表示する

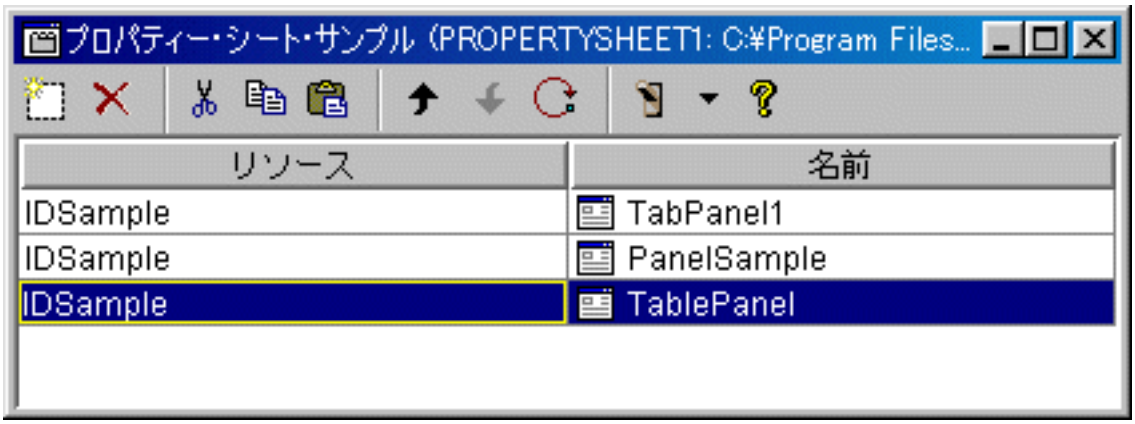

デック・ペイン
表示
カスタム・セル・エディターのテーブル
名前
チェック・ボックス欄
サンプル名

GUI ビルダーによるプロパティー・シートの作成

GUI ビルダーを使用すると、プロパティー・シートを容易に作成することができます。 GUI ビルダーのメイン・ウィンドウのメニュー・バーから、 「ファイル」 --> 「新規ファイル」を選択します。

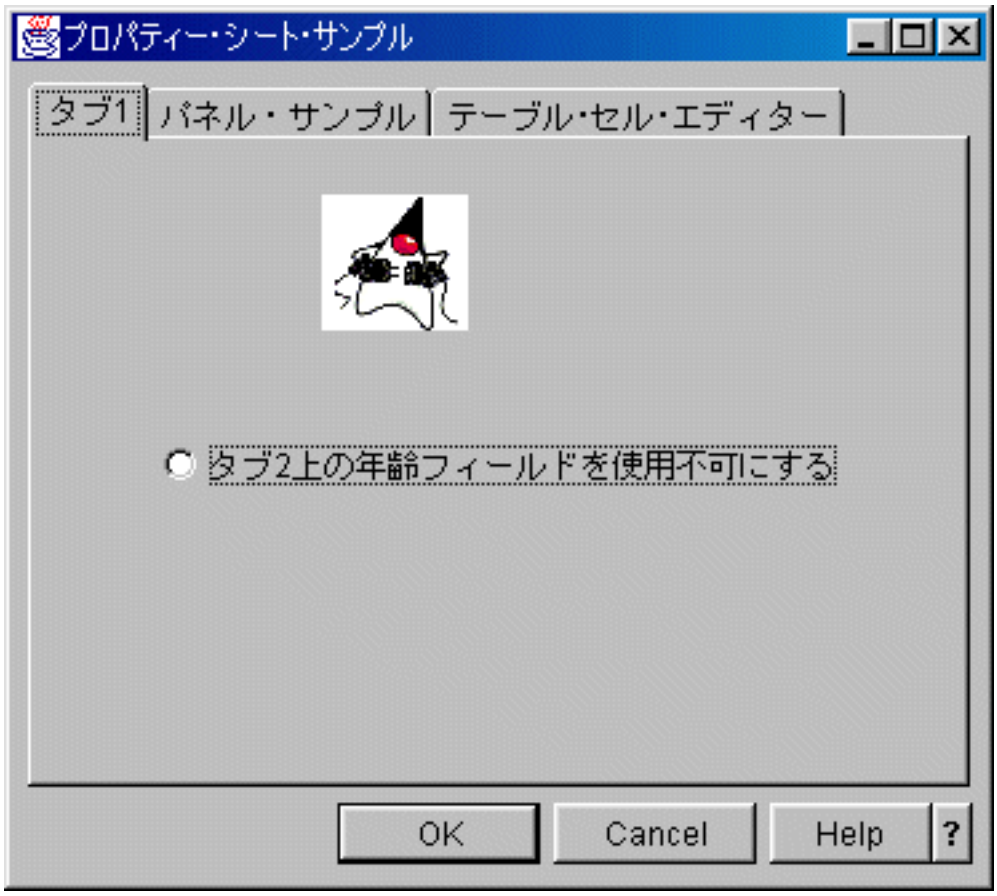
GUI ビルダーの「ファイル」ウィンドウのメニュー・バーから、 「プロパティー・シートの挿入」アイコン  をクリックして、 プロパティー・シートのコンポーネントを挿入できるパネル・ビルダーを表示させます。

図 1: GUI ビルダーによるプロパティー・シートの作成



プロパティー・シートを作成したら、  アイコンを使用してそれをプレビューします。 この例では、以下の 3 つのタブから選択できます。

図 2: GUI ビルダーによるプロパティー・シートのプレビュー

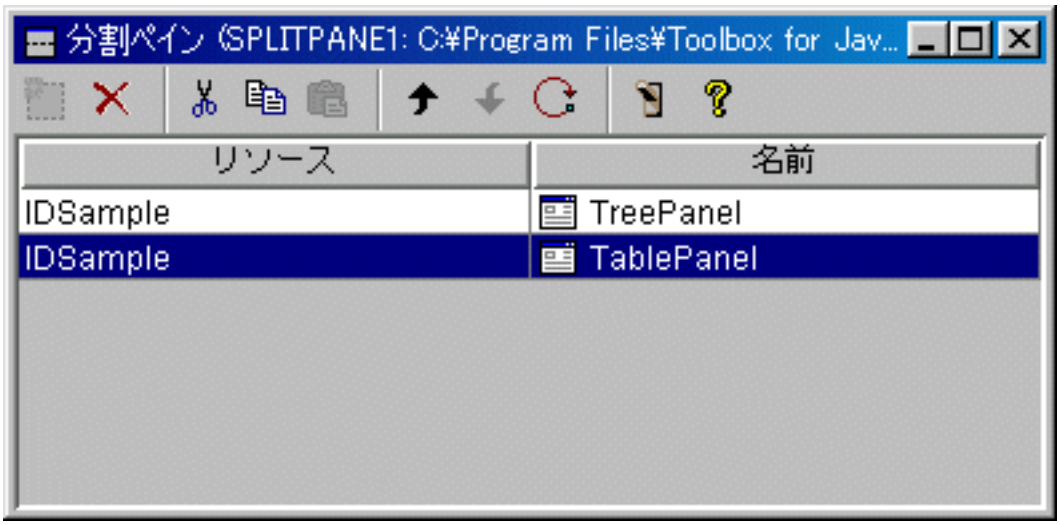

プロパティー・シート・サンプル
タブ1
パネル・サンプル
テーブル・セル・エディター
タブ2上の年齢フィールドを使用不可にする
OK
Cancel
Help
?

GUI ビルダーを使用して分割ペインを作成する

GUI ビルダーを使用すると、分割ペインを簡単に作成することができます。GUI ビルダーのメイン・ウィンドウのメニュー・バーから、「ファイル」 --> 「新規ファイル」を選択します。

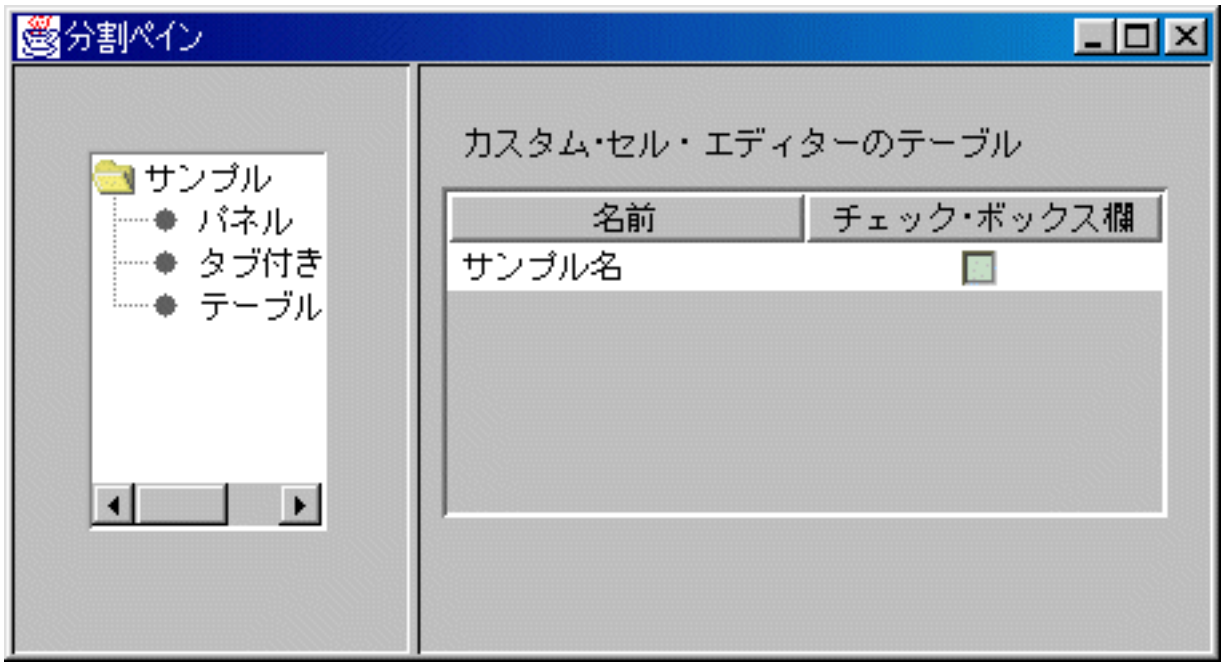
GUI ビルダーの「ファイル」ウィンドウのメニュー・バーから、「分割ペインの挿入 (Insert Split Pane)」ツール・ボタン  をクリックして、分割ペインに含めたいコンポーネントを挿入できるパネル・ビルダーを表示させます。以下の例では、2 つのコンポーネントが追加されています。

図 1: GUI ビルダーによる分割ペインの作成



分割ペイン作成後、図 2 に示されているように、「プレビュー (Preview)」ツール・ボタン  アイコンをクリックし、それをプレビューします。

図 2: GUI ビルダーによる分割ペインのプレビュー

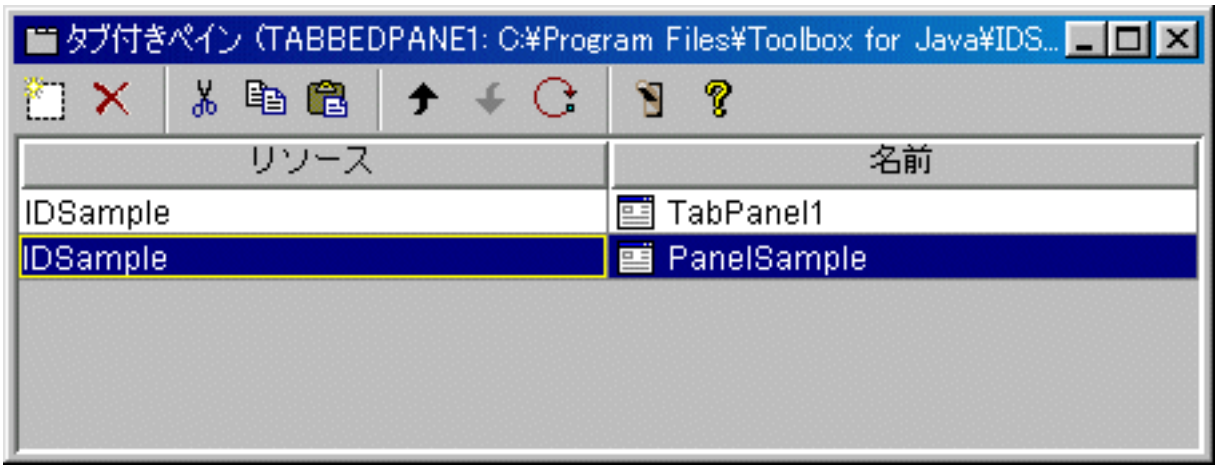

分割ペイン
サンプル
パネル
タブ付き
テーブル
カスタム・セル・エディターのテーブル
名前
チェック・ボックス欄
サンプル名

GUI ビルダーを使用してタブ付きペインを作成する

GUI ビルダーを使用すると、タブ付きペインを簡単に作成することができます。 GUI ビルダーのメイン・ウィンドウのメニュー・バーから、 「ファイル」 --> 「新規ファイル」を選択します。

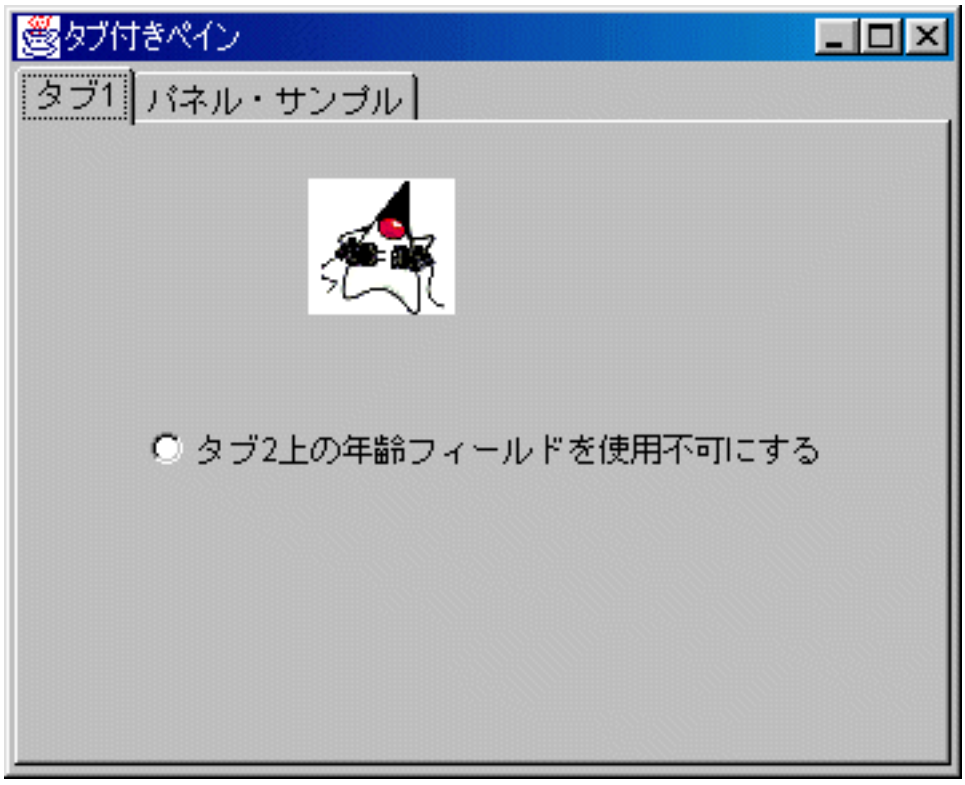
GUI ビルダーの「ファイル」ウィンドウのメニュー・バーから、 「タブ付きペインの挿入 (Insert Tabbed Pane)」アイコン  をクリックして、 タブ付きペインのコンポーネントを挿入できるパネル・ビルダーを表示させます。 以下の例では、2 つのコンポーネントが追加されています。

図 1: GUI ビルダーにおけるタブ付きペインの作成



タブ付きペインを作成した後、「プレビュー (Preview)」ツール・ボタン  をクリックし、 それをプレビューします。

図 2: GUI ビルダーによるタブ付きペインのプレビュー

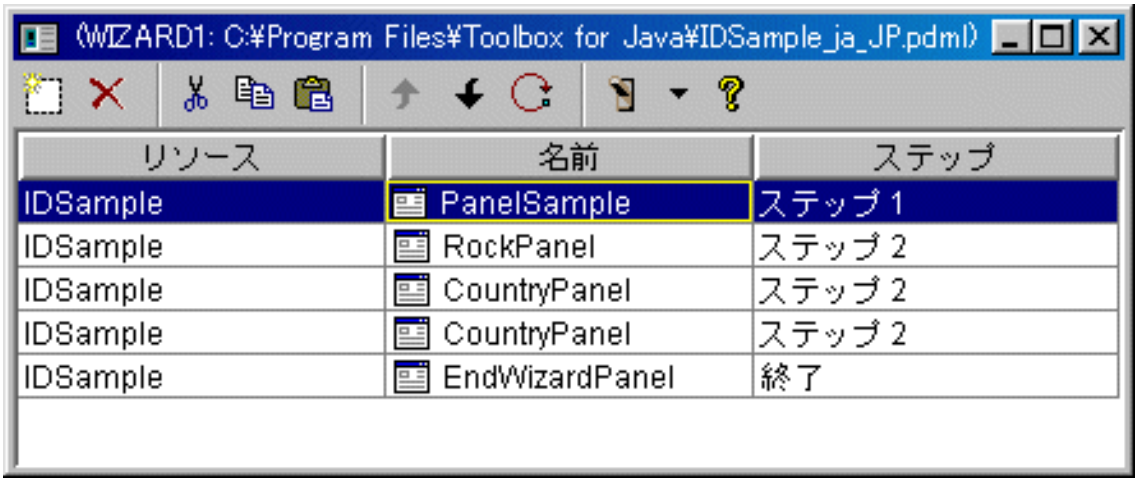

タブ付きペイン
タブ1
パネル・サンプル
タブ2上の年齢フィールドを使用不可にする

GUI ビルダーでウィザードを作成する

GUI ビルダーを使用すると、 ウィザード・インターフェースを容易に作成できます。 「GUI ビルダー」ウィンドウのメニュー・バーから、 「ファイル」 --> 「新規ファイル」を選択します。

GUI ビルダーの「ファイル」ウィンドウのメニュー・バーから、 「ウィザードの挿入 (Insert Wizard)」ツール・ボタン をクリックし、 パネルをウィザードに追加できるパネル・ビルダーを表示させます。

図 1: GUI ビルダーによるウィザードの作成



ウィザード作成後、 「プレビュー (Preview)」ツール・ボタン を使用し、 それをプレビューします。 図 2 は、 この例で最初に表示されるパネルを示しています。

図 2: GUI ビルダーによる最初のウィザード・パネルのプレビュー

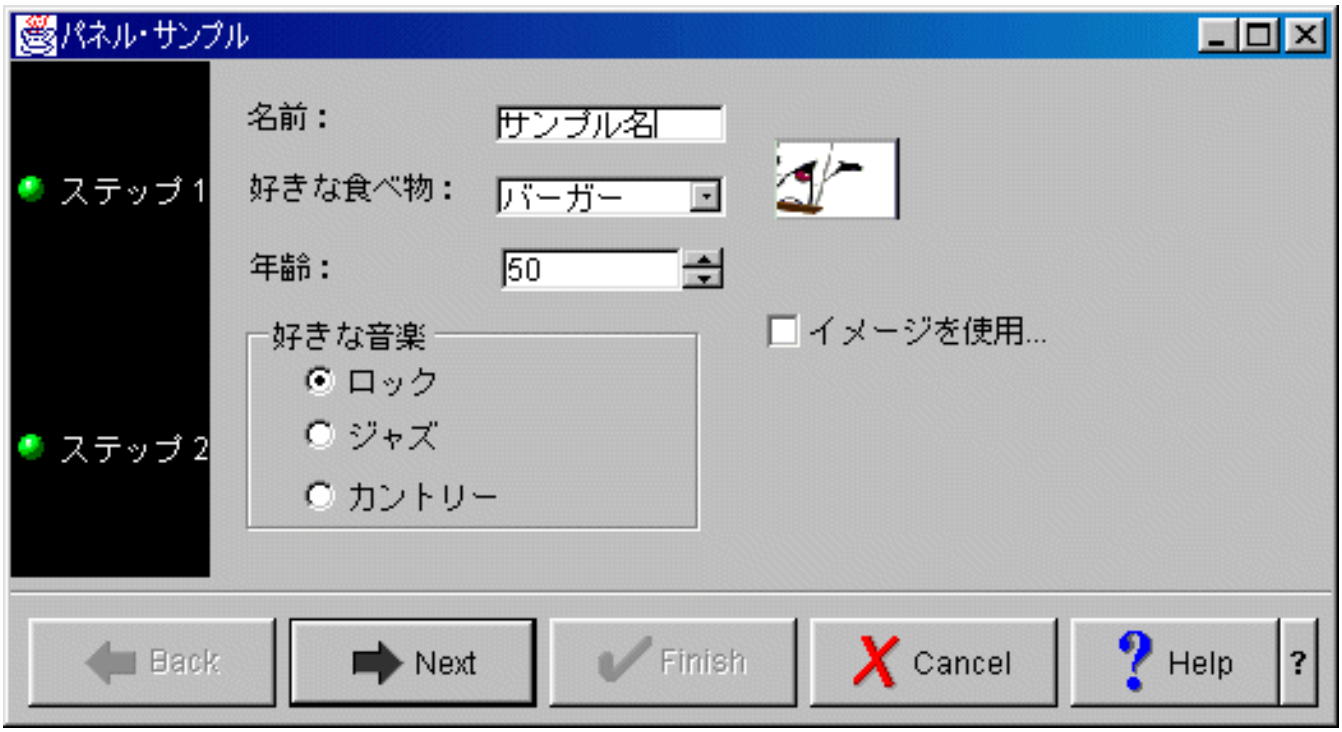
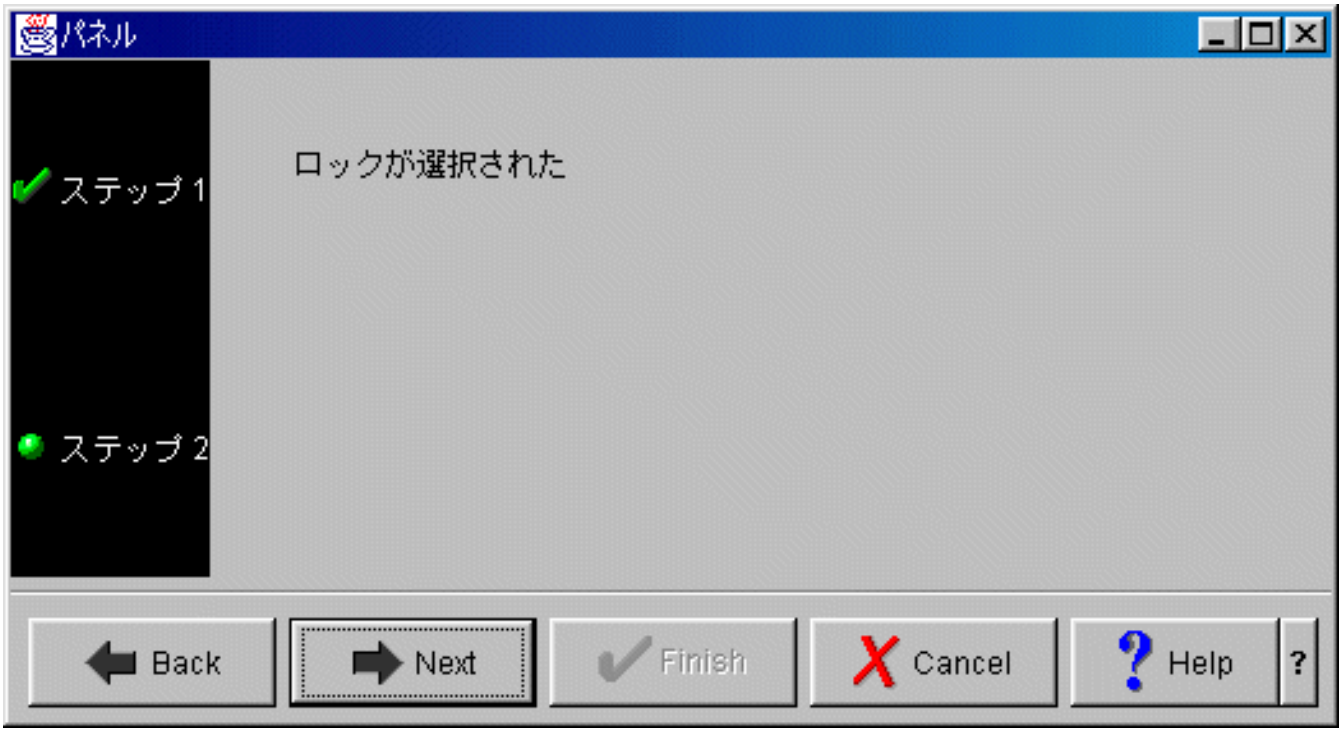


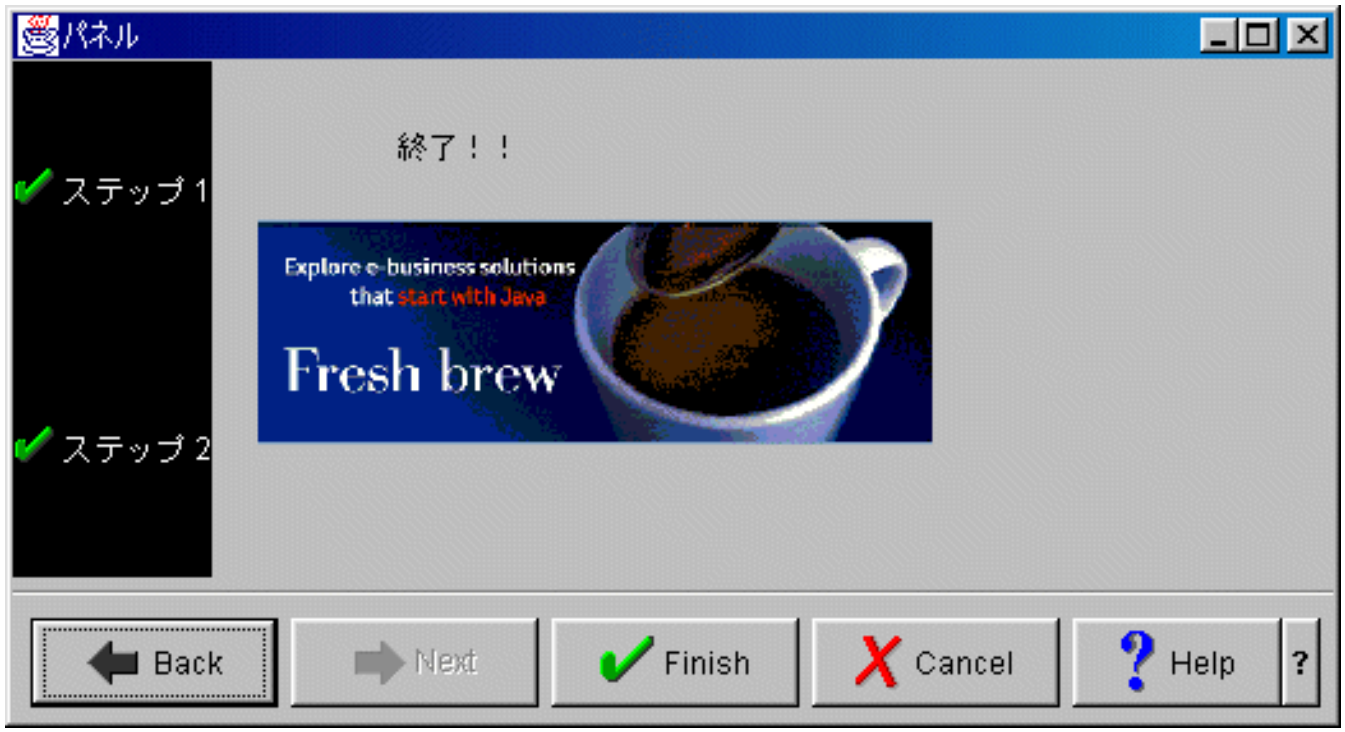
図 2 は、ユーザーが「ロック」を選択して、 「次へ (Next)」をクリックした時に表示される 2 番目のパネルを示しています。

図 2: GUI ビルダーによる 2 番目のウィザード・パネルのプレビュー



2 番目のウィザード・パネルで「次へ (Next)」をクリックすると、 図 3 に示されているように、最後のパネルが表示されます。

図 3: GUI ビルダーによる最後のウィザード・パネルのプレビュー

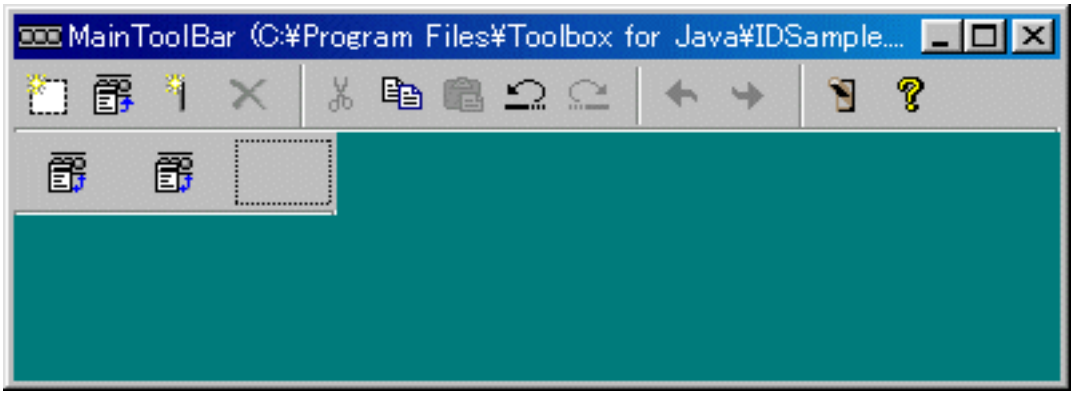

パネル
ステップ 1
ステップ 2
終了！！
Explore e-business solutions
that start with Java
Fresh brew
Back
Next
Finish
Cancel
Help
?

GUI ビルダーを使用してツールバーを作成する

GUI ビルダーを使用すると、ツールバーを簡単に作成することができます。「GUI ビルダー」ウィンドウのメニュー・バーから、「ファイル」 --> 「新規ファイル」を選択します。

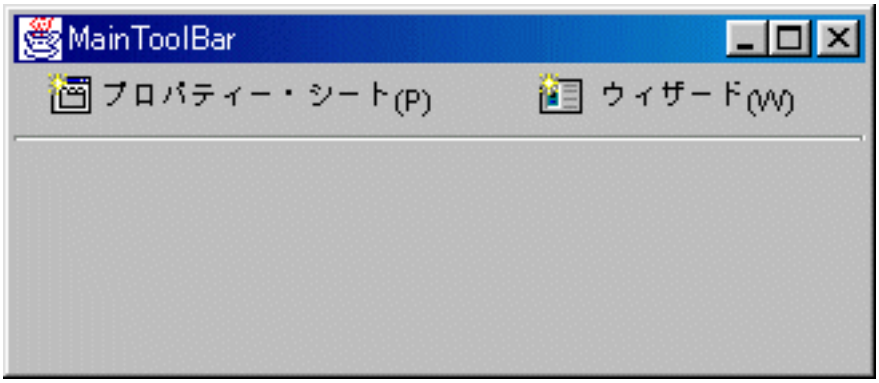
GUI ビルダーの「ファイル」ウィンドウのメニュー・バーから、「ツールバーの挿入 (Insert Tool Bar)」ツール・ボタンをクリックし、ツールバーのコンポーネントを挿入できるパネル・ビルダーを表示させます。

図 1: GUI ビルダーによるツールバーの作成



ツールバー作成後、「プレビュー (Preview)」ツール・ボタン  をクリックし、それをプレビューします。この例では、プロパティー・シートまたはウィザードのいずれかを表示するためにツールバーを使用することができます。

図 2 : GUI ビルダーによるツールバーのプレビュー

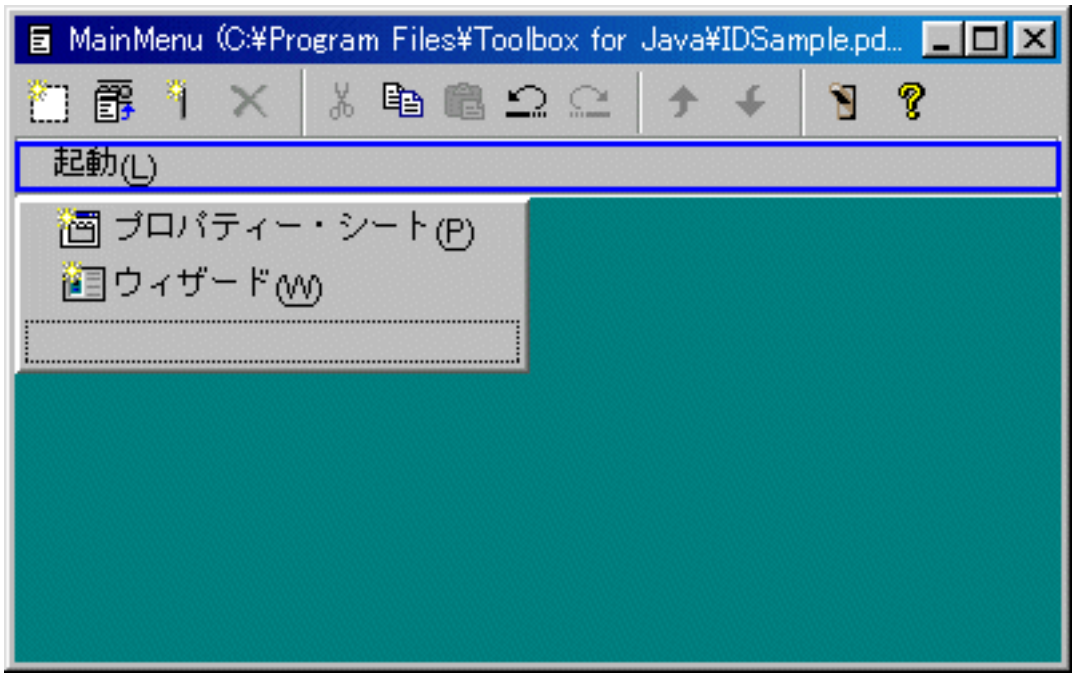


GUI ビルダーでメニュー・バーを作成する

GUI ビルダーを使用すると、メニュー・バーを簡単に作成することができます。 「GUI ビルダー」ウィンドウのメニュー・バーから、 「ファイル」 --> 「新規ファイル」を選択します。

GUI ビルダーの「ファイル」ウィンドウのツールバーから、 「メニューの挿入 (Insert Menu)」ツール・ボタンをクリックし、 メニュー用のコンポーネントを挿入できるパネル・ビルダーを作成します。

図 1: GUI ビルダー: メニューを作成する



メニューを作成した後、「プレビュー (Preview)」ツール・ボタン を使用してプレビューします。 この例では、新しく作成した「起動」メニューから、「プロパティー・シート」または 「ウィザード」のいずれかが選択可能です。 以下の図は、これらのメニュー項目を選択した際に見えるものを示しています。

図 2: GUI ビルダー: 「起動」メニューの「プロパティー・シート」を表示する

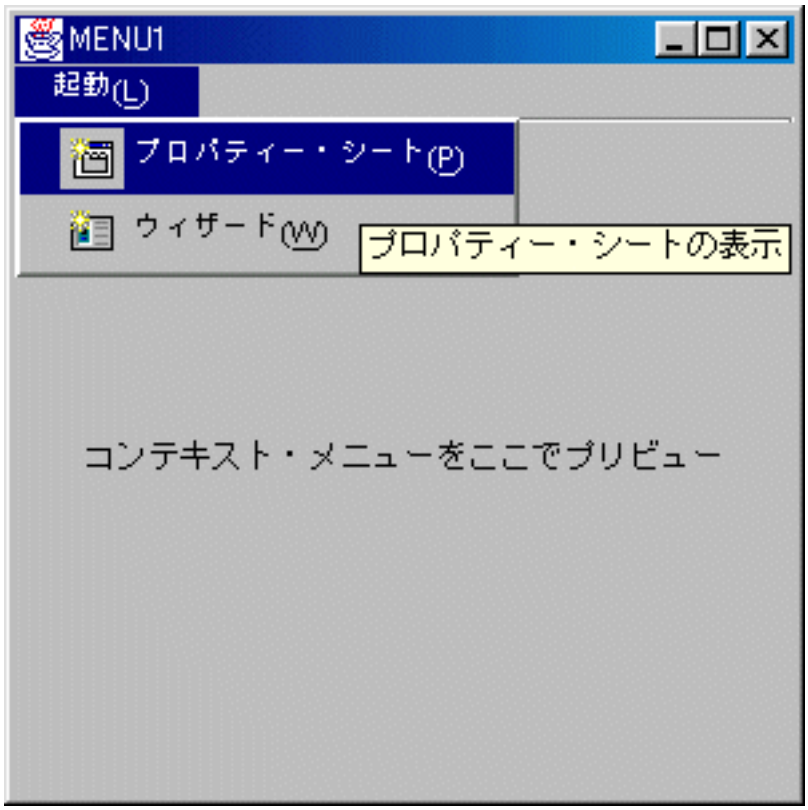
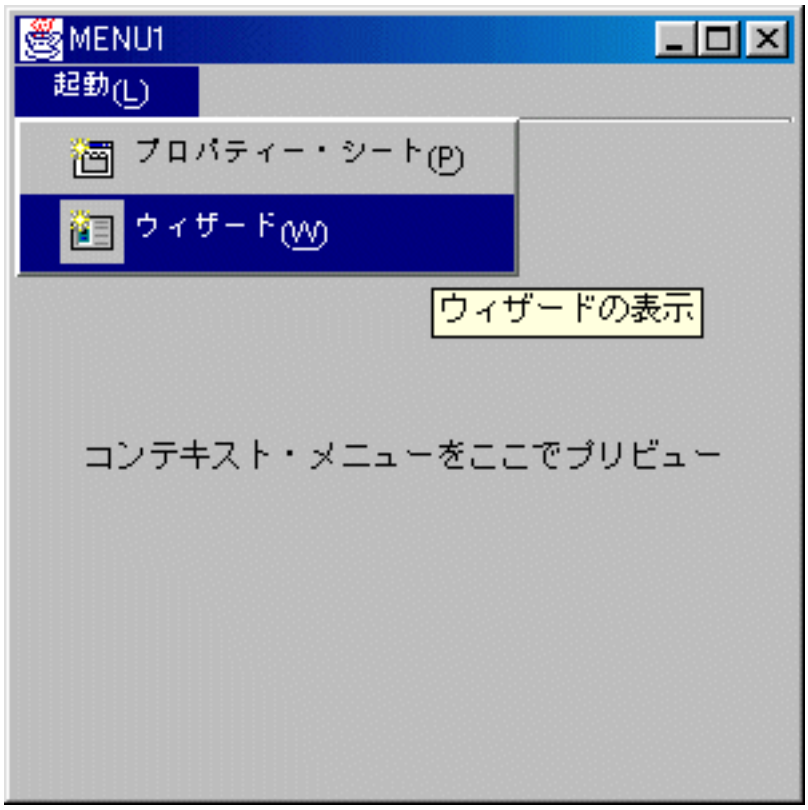


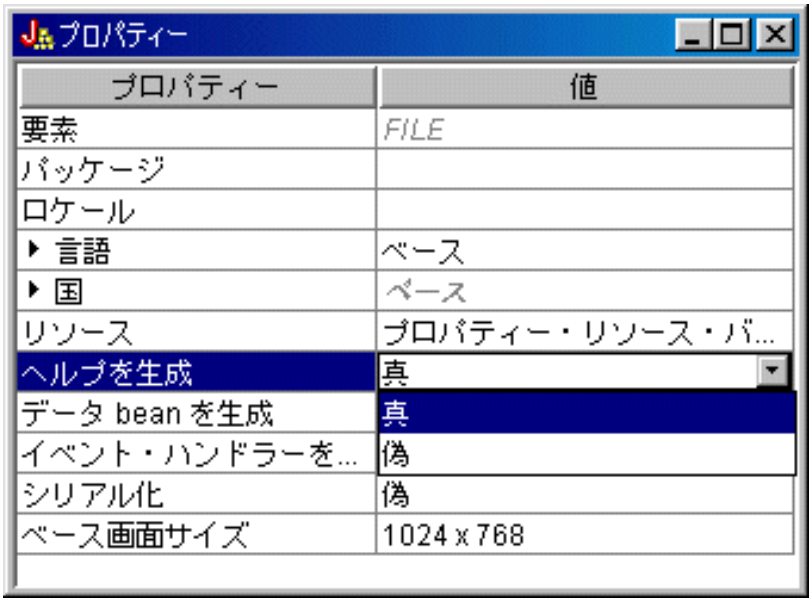
図 3 GUI ビルダー: 「起動」メニューの「ウィザード」を表示する



例: ヘルプ文書の作成

GUI ビルダーを使用すると、ヘルプ・ファイルを簡単に作成できます。作業しているファイルのプロパティー・パネルで、「ヘルプを生成」を真に設定してください。

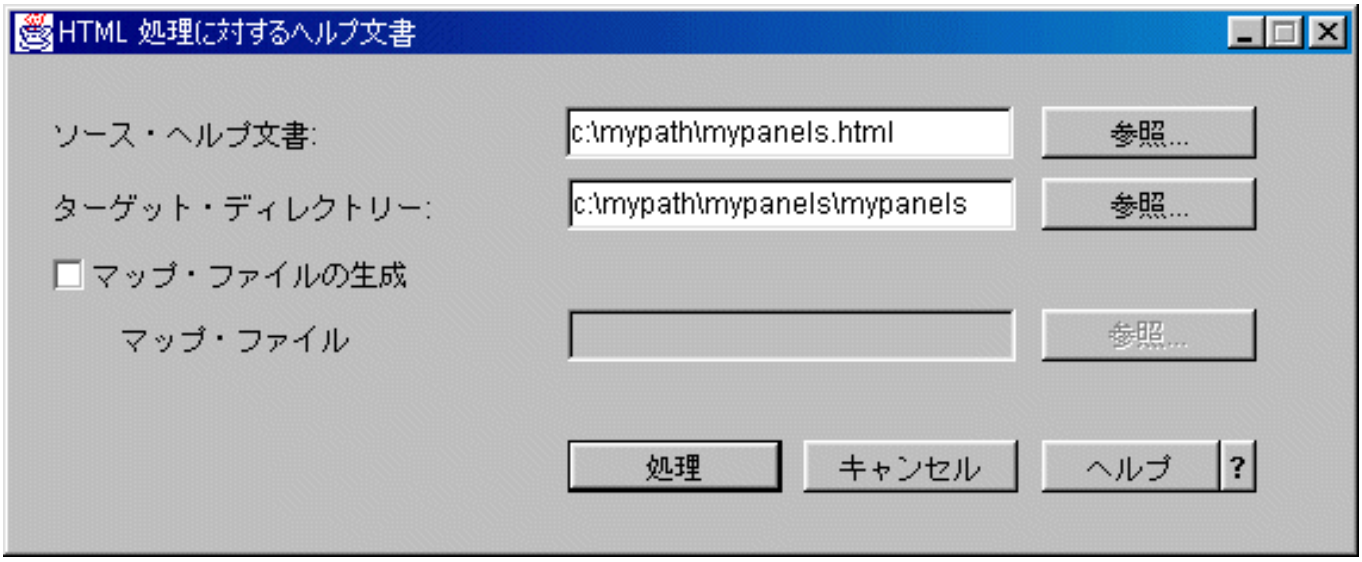
図 1: GUI ビルダーの「プロパティー」パネルの「ヘルプを生成」プロパティーを設定する



GUI ビルダーはヘルプ文書という HTML フレームワークを作成します。この文書を編集することができます。

実行時に使用するには、 PDML ファイル内のトピックを個々の HTML ファイルに分離する必要があります。 「HTML 処理に対するヘルプ文書」 を実行すると、トピックは個々のファイルに分けられてから、ヘルプ文書および PDML ファイルにちなんで名前が付けられたサブディレクトリーに置かれます。実行時環境では、個々の HTML ファイルはヘルプ文書および PDML ファイルと同じ名前のサブディレクトリー内に入っていなければなりません。 「HTML 処理に対するヘルプ文書」ダイアログは、必要な情報を収集してから、HelpDocSplitter プログラムを呼び出して以下の処理を行います。

図 2: 「HTML 処理に対するヘルプ文書」ダイアログ



「HTML 処理に対するヘルプ文書」を開始するには、コマンド・プロンプトで以下のように入力します。

```
jre com.ibm.as400.ui.tools.hdoc2htmViewer
```

このコマンドを実行するには、 クラスパスを正しく設定する必要があります。

「HTML 処理に対するヘルプ文書」を使用するには、まず、PDML ファイルと同じ名前のヘルプ文書を選択します。それから、出力されるヘルプ文書および PDML ファイルと同じ名前のサブディレクトリーを指定します。最後に「処理」を選択して処理を完了します。

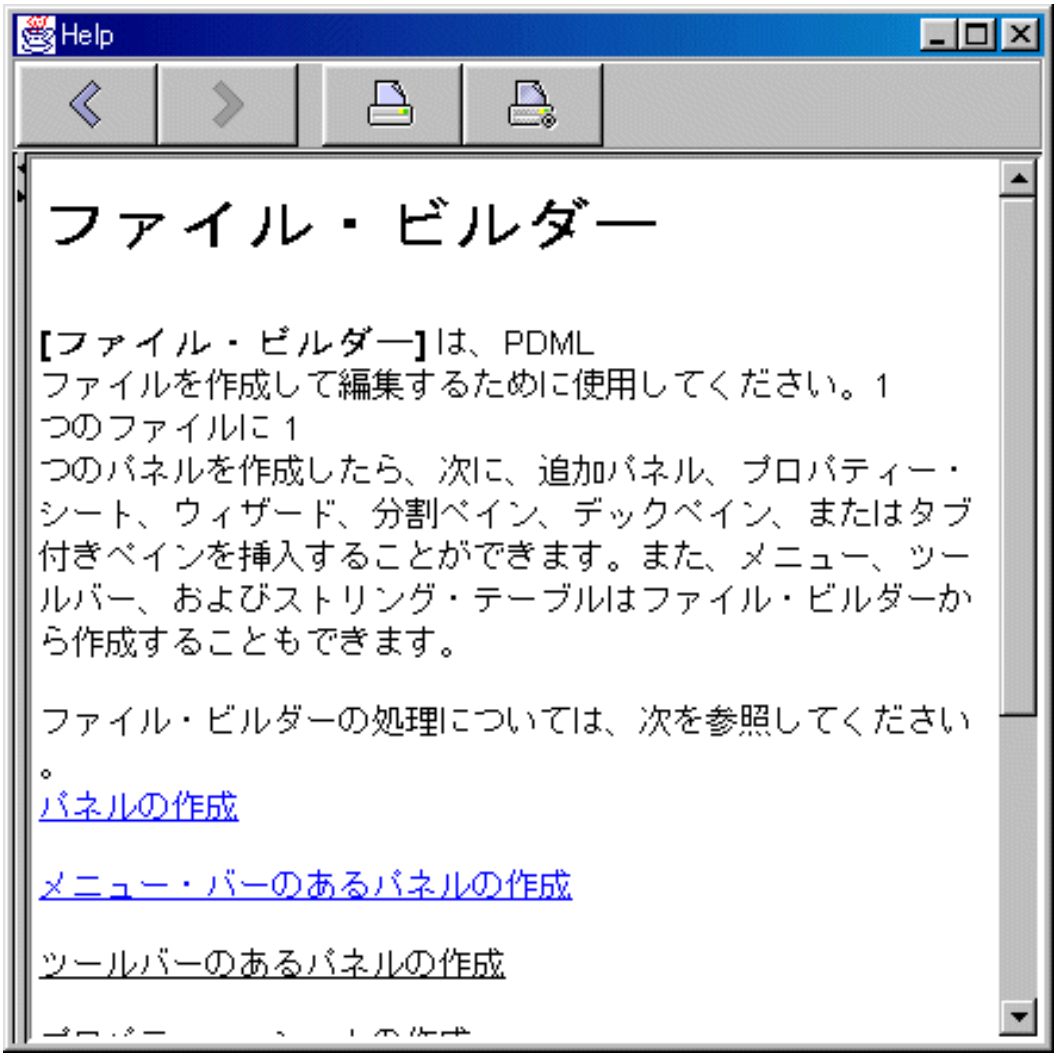
ヘルプ文書を分割するには、コマンド行で以下のようなコマンドを使います。

```
jre com.ibm.as400.ui.tools.HelpDocSplitter "helpdocument.htm" [output directory]
```

このコマンドはファイルを分割する処理を実行します。ここでは、ヘルプ文書の名前を入力してください (任意選択で出力ディレクトリーの名前を入力することもできます)。デフォルトでは、ヘルプ文書と同じ名前のサブディレクトリーが作成されて、結果ファイルはそのディレクトリーに入れられます。

この例は、ヘルプ・ファイルがどのようなものかを示します。

図 3: GUI ビルダーのヘルプ・ファイルの例

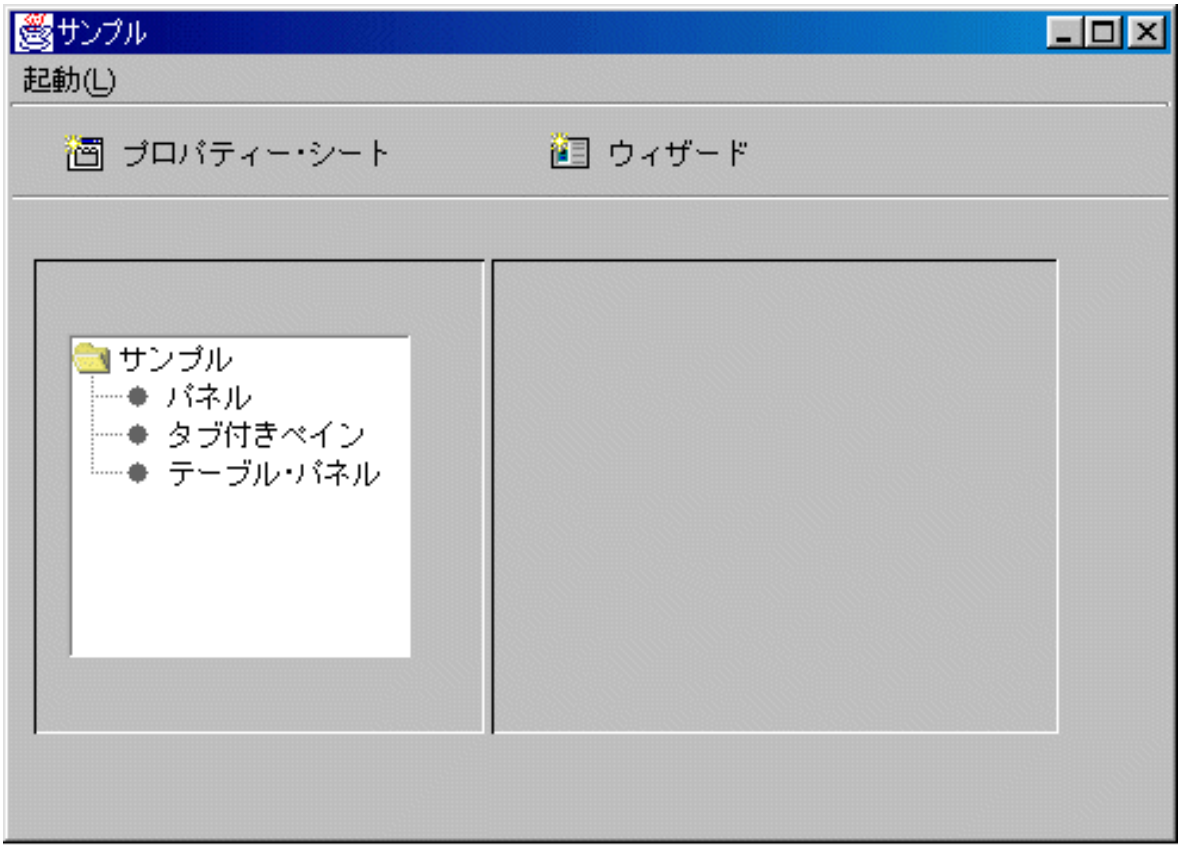


例: GUI ビルダーを使用する

ここに示されている例と、背景で機能している正確なデータ bean を一緒にすると、 完全な GUI アプリケーションになります。

図 1 は、この例を実行する時に表示される最初のパネルを示しています。

図 1: GUI ビルダーのサンプル・メイン・ウィンドウ



この画面では、動的パネル管理機能を使用できることに注意してください。 図 2 および図 3 は、ウィンドウのサイズをどのように変更できるかを示しています。

図 2: GUI ビルダーのサンプル・メイン・ウィンドウのサイズ変更 (大きくする)

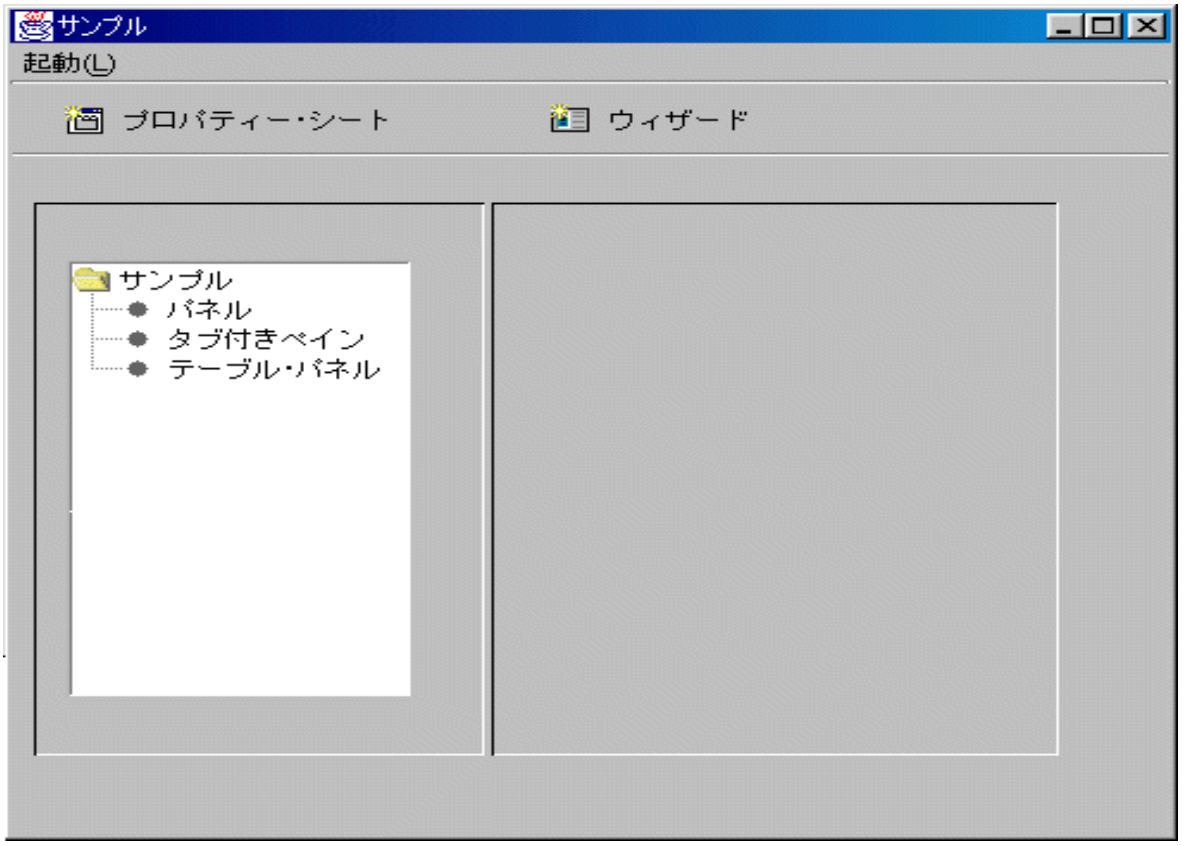
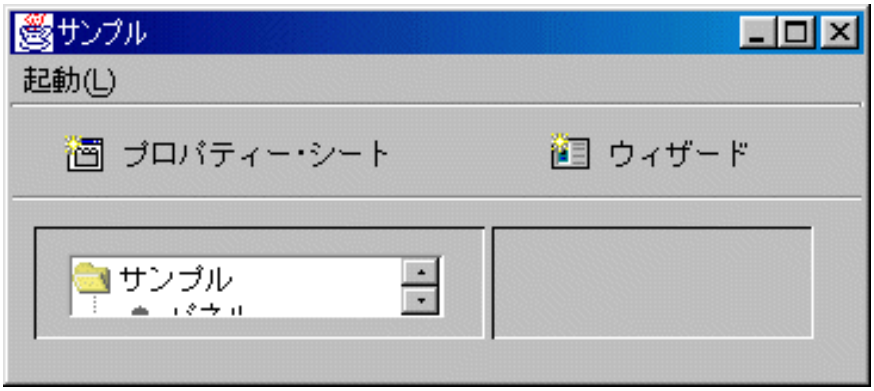


図 3: GUI ビルダーのサンプル・メイン・ウィンドウのサイズ変更 (小さくする)



動的パネル管理機能を使用すると、 パネルとパネル・コントロールのサイズは変わりますが、テキストのサイズは変わりません。

このパネルを使用すると、以下の処理を行うことが可能になります。

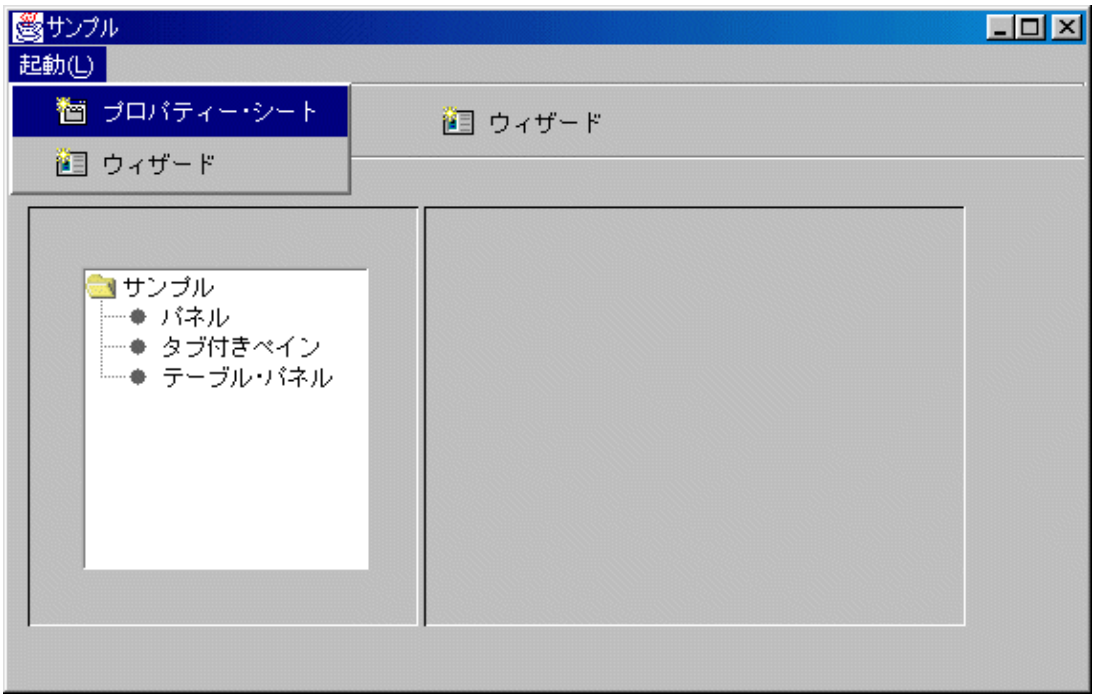
- プロパティー・シートを起動する
- ウィザードを起動する
- 左側のペインにリストされているサンプルの表示

プロパティー・シートの起動

ツールバーの「プロパティー・シート」ボタンをクリックするか、 「起動」メニューを使用することにより、プロパティー・シートを起動できます。 ツールバーかメニューのどち

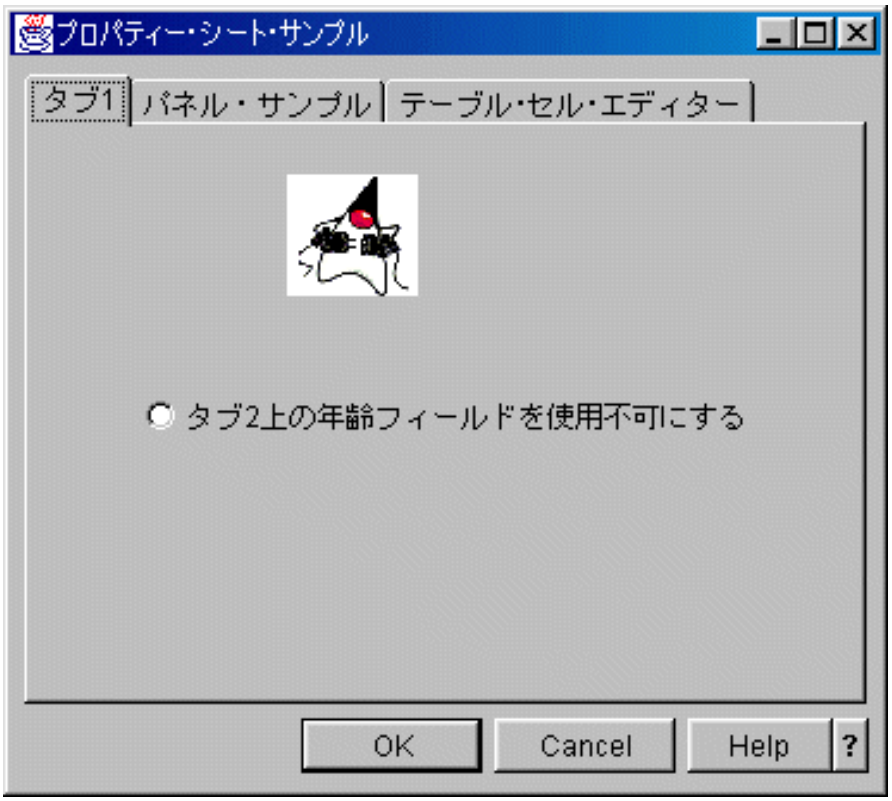
らかを選択できるということは、メニュー項目をリンクするということを意味しています。 図 4 は、GUI ビルダーのサンプル・メイン・ウィンドウの「起動」メニューから、「プロパティー・シート」が選択されている様子を示しています。

図 4: 「起動」メニューからの「プロパティー・シート」の選択



「プロパティー・シート」を選択すると、図 5 のパネルが表示されます。

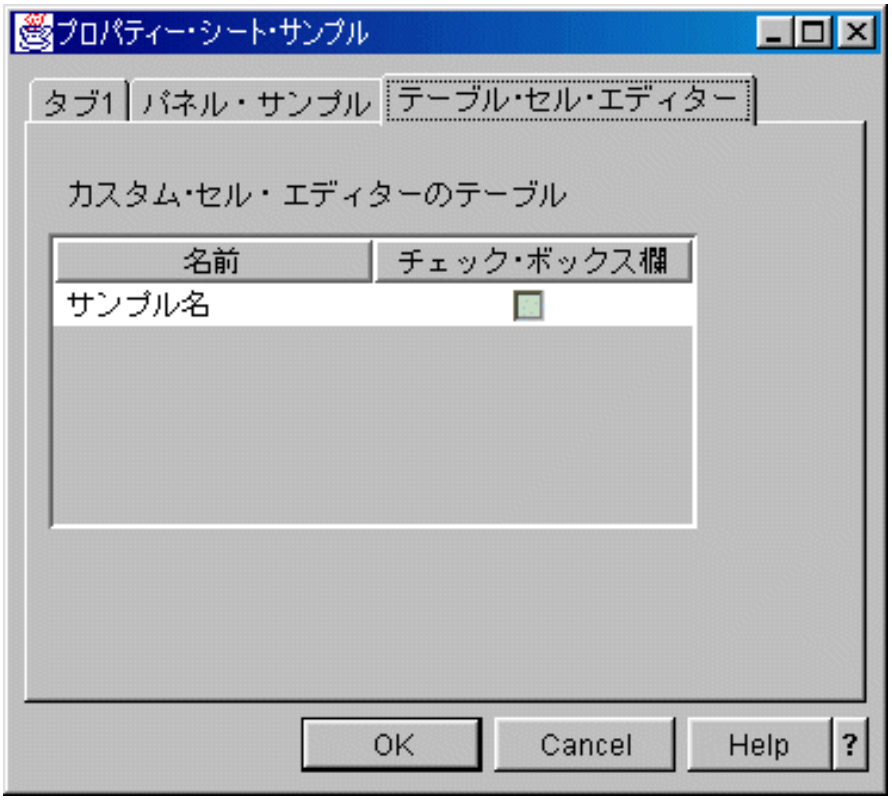
図 5: 「プロパティー・シート・サンプル」ダイアログ



「プロパティー・シート・サンプル」が最初に表示される時、「タブ 1」がデフォルトで表示されます。 図 6 および図 7 は、他のタブを選択するとどのようにパネル表示が変更されるかを示しています。

図 6: 「パネル・サンプル」タブの選択

図 7: 「テーブル・セル・エディター」タブの選択



ウィザードの起動

ツールバーの「ウィザード」ボタンをクリックするか、「起動」メニューを使用することにより、ウィザードを起動できます。ツールバーかメニューのどちらかを選択できるということは、メニュー項目をリンクするということを意味しています。図 8 は、GUI ビルダーのサンプル・メイン・ウィンドウの「起動」メニューから、「ウィザード」が選択されている様子を示しています。

「起動」メニューからの「ウィザード」の選択

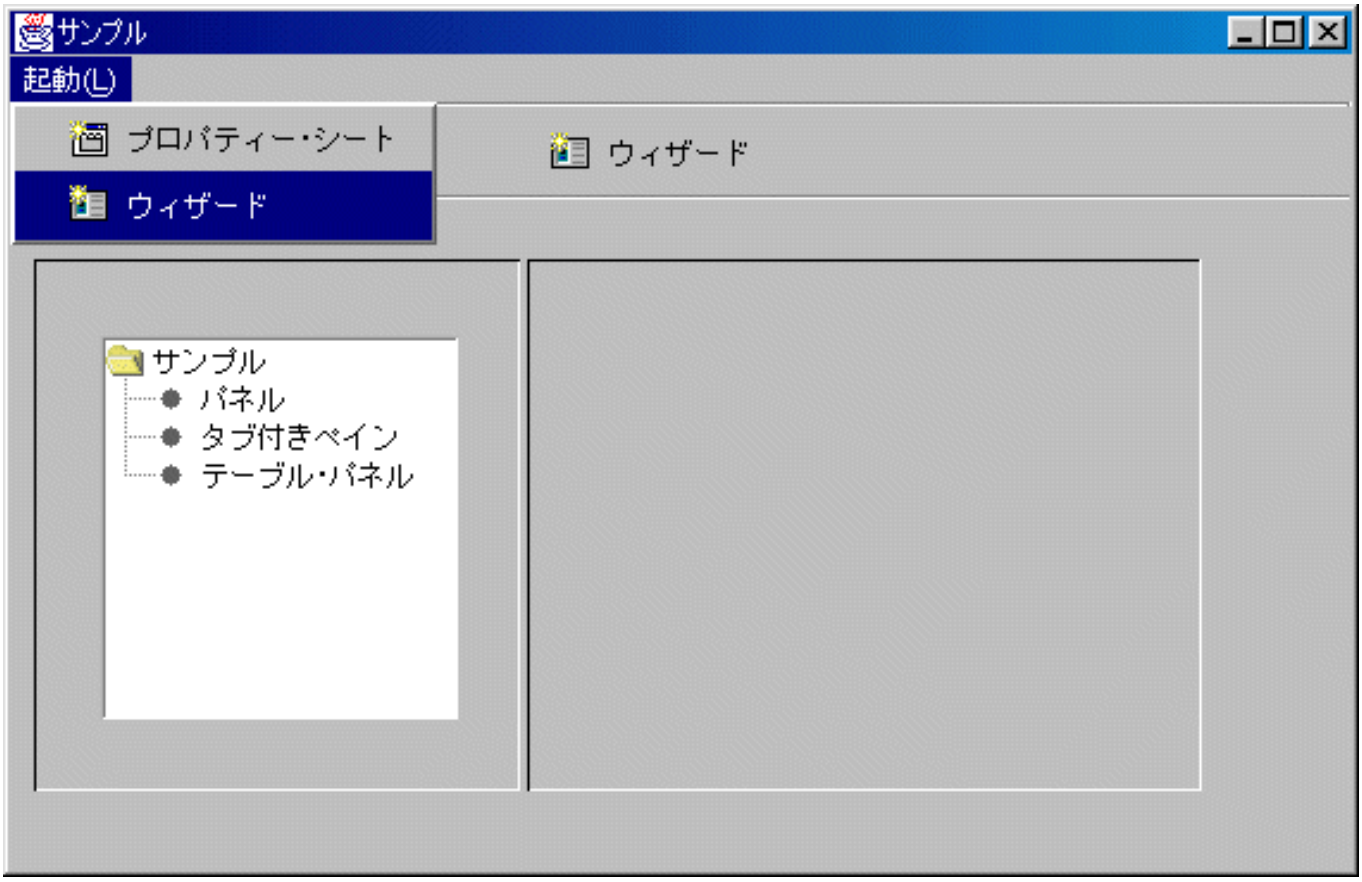
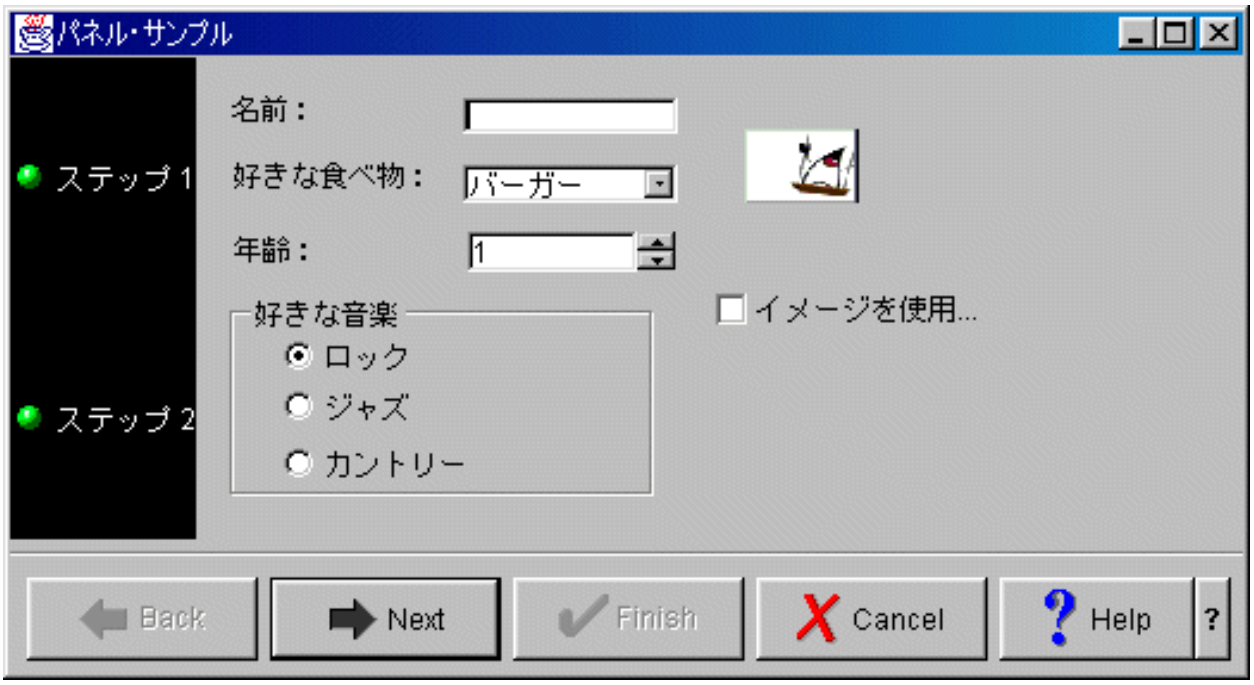


図 9 は、どのように最初のウィザード・ダイアログで多くのオプションが提供されるかを示しています。

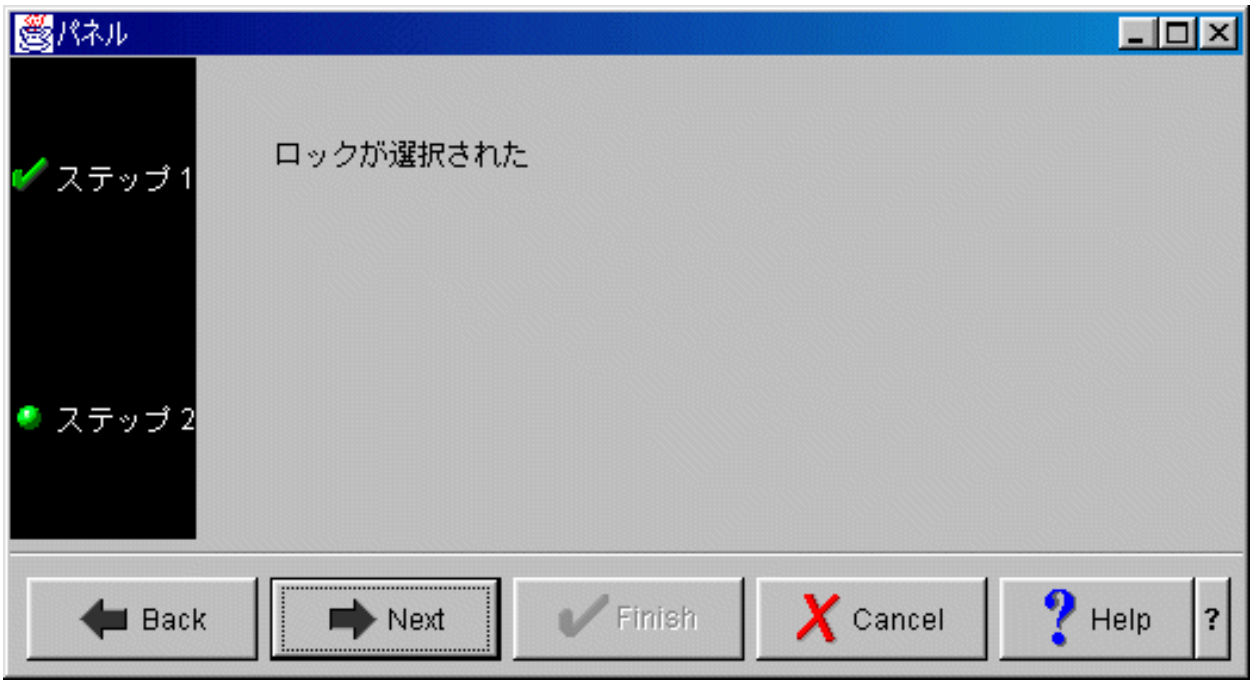
最初のウィザード・ダイアログでの「ロック」の選択



最初のウィザード・ダイアログで、「ロック」を選択し、「Next」をクリックすると、図

10 に示されているような 2 番目のウィザード・ダイアログが表示されます。

図 10: 2 番目のウィザード・ダイアログ (「ロック」を選択した後)



2 番目のウィザード・ダイアログで、「Next」をクリックすると、 図 11 に示されているような最後のウィザード・ダイアログが表示されます。

図 11: 最後のウィザード・ダイアログ

ただし、この例は、ループするようにプログラムされています。 最初のウィザード・ダイアログ (図 12) にある「カントリー」を選択し、 「Next」をクリックすると、2 番目のウィザード・ダイアログ (図 13) が表示されます。 2 番目のウィザード・ダイアログで「Next」

をクリックすると、ループバックし、 最後のウィザード・ダイアログの代わりに、最初のダイアログ (図 14) をまた表示します。

図 12: 最初のウィザード・ダイアログでの「カントリー」の選択

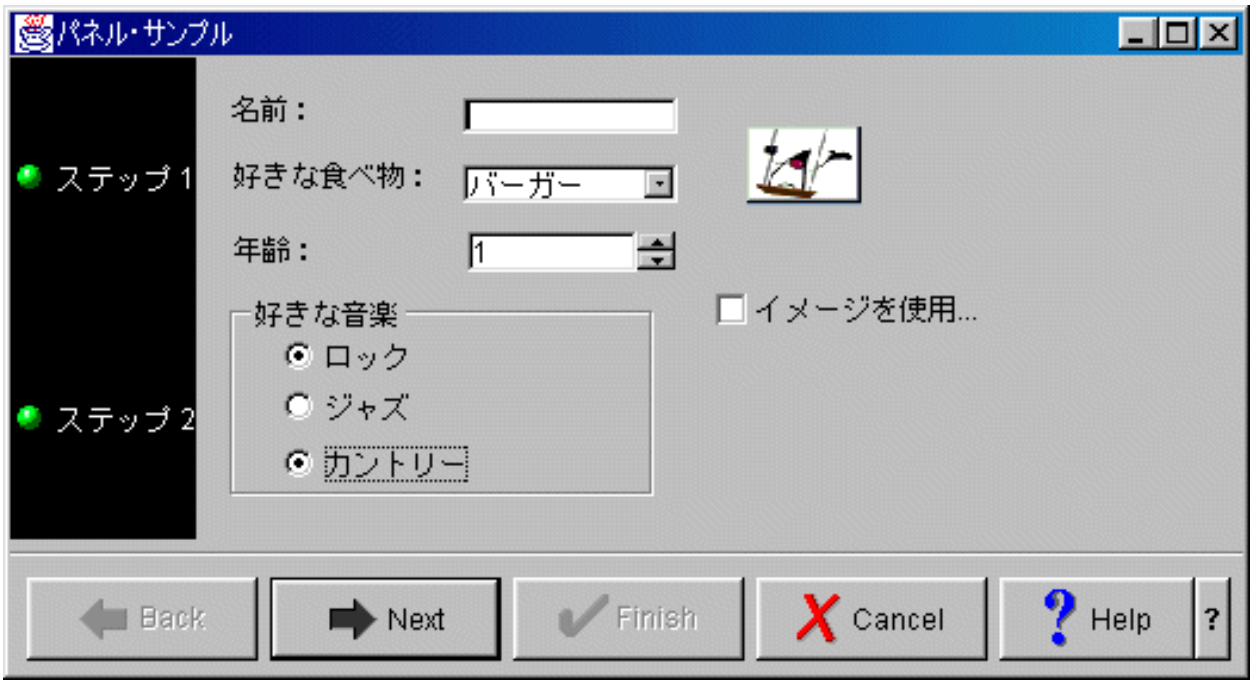


図 13: 2 番目のウィザード・ダイアログ (「カントリー」を選択した後)

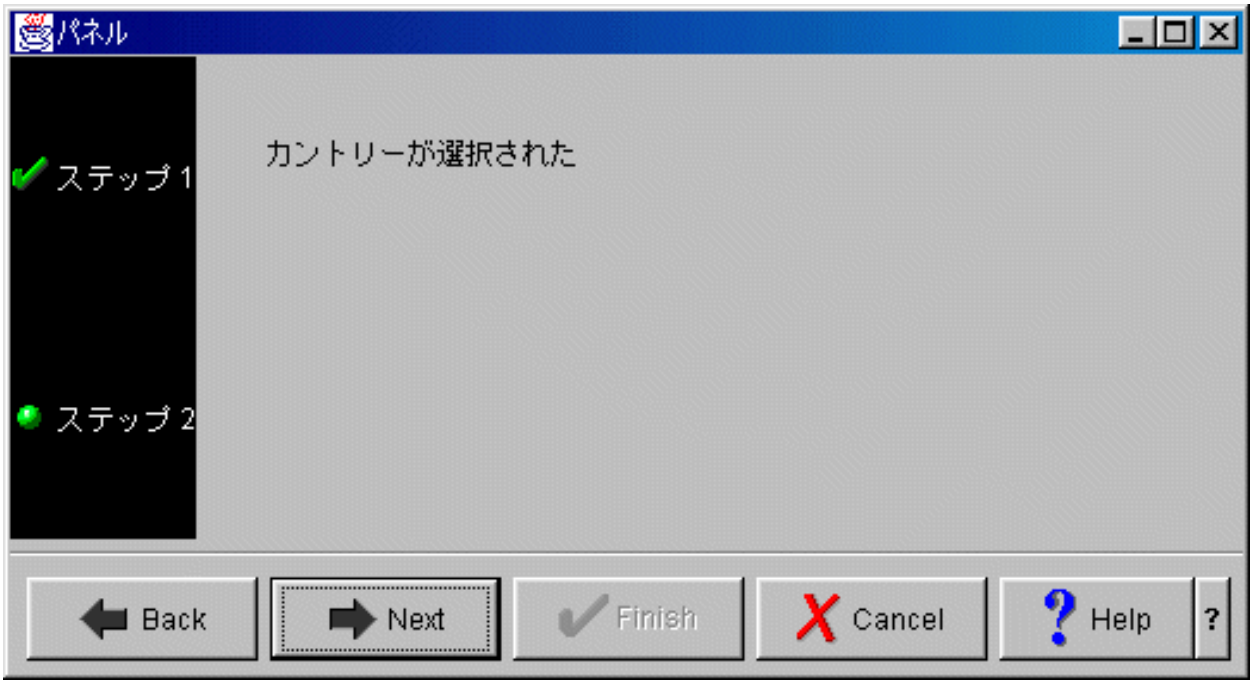
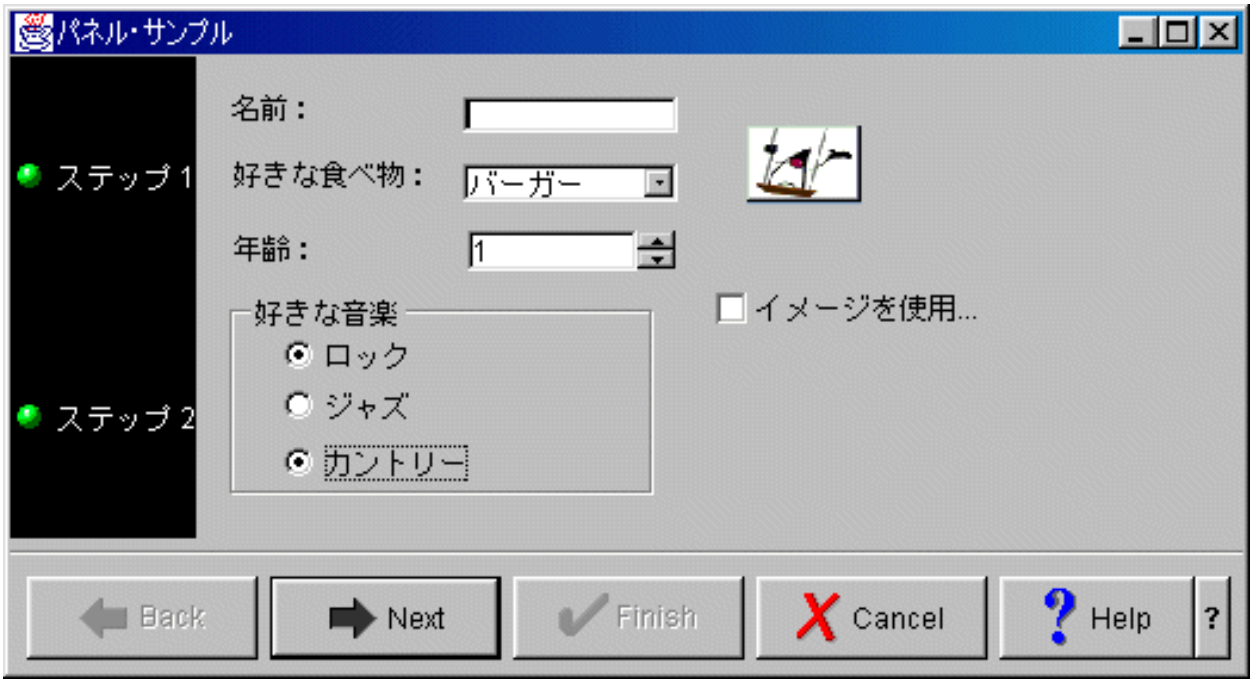


図 14: 最初のウィザード・ダイアログへのループバック

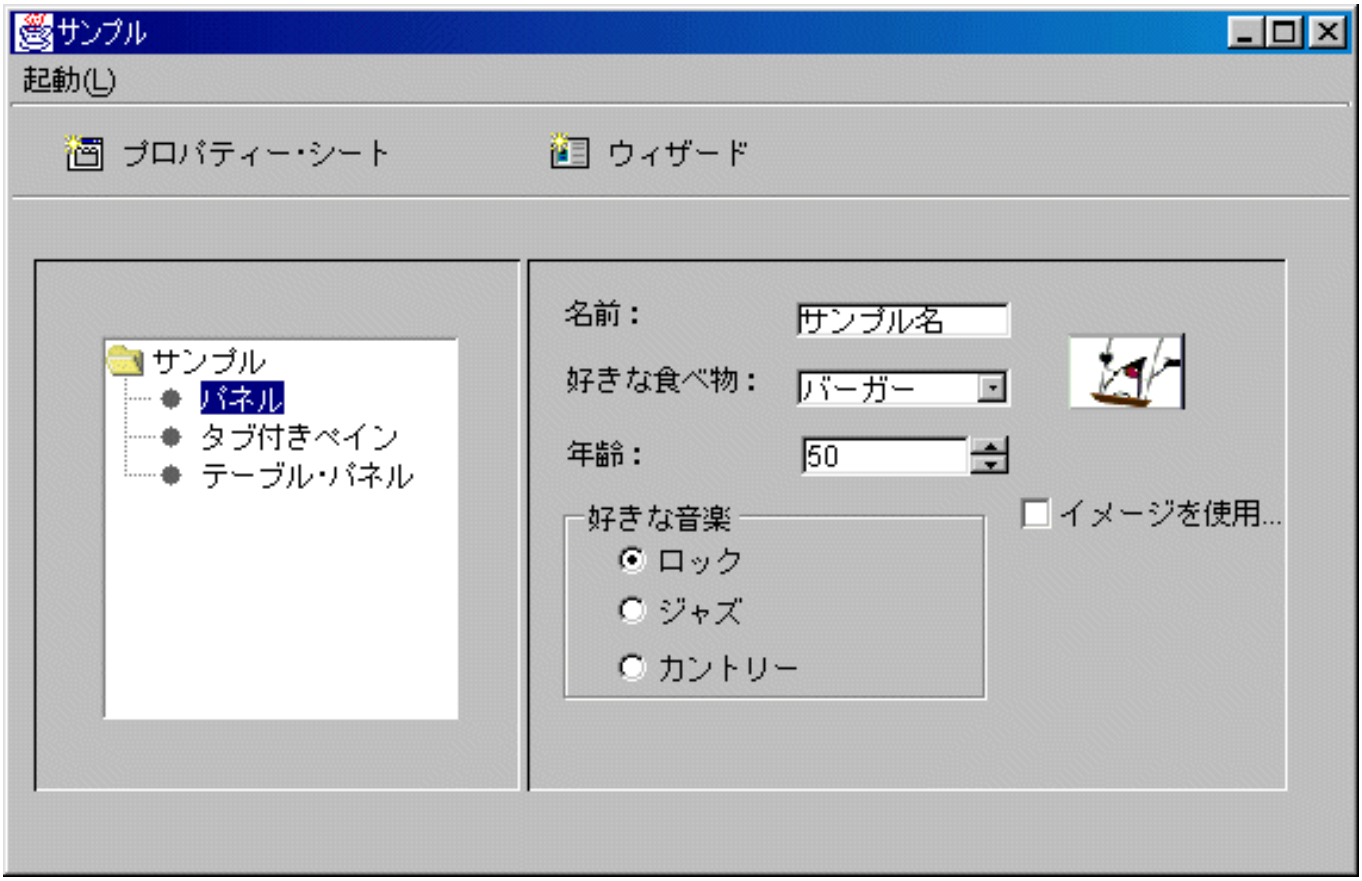


つまり、お気に入りの音楽ジャンルとしてカントリーを選択できないようになっています。

サンプルの表示

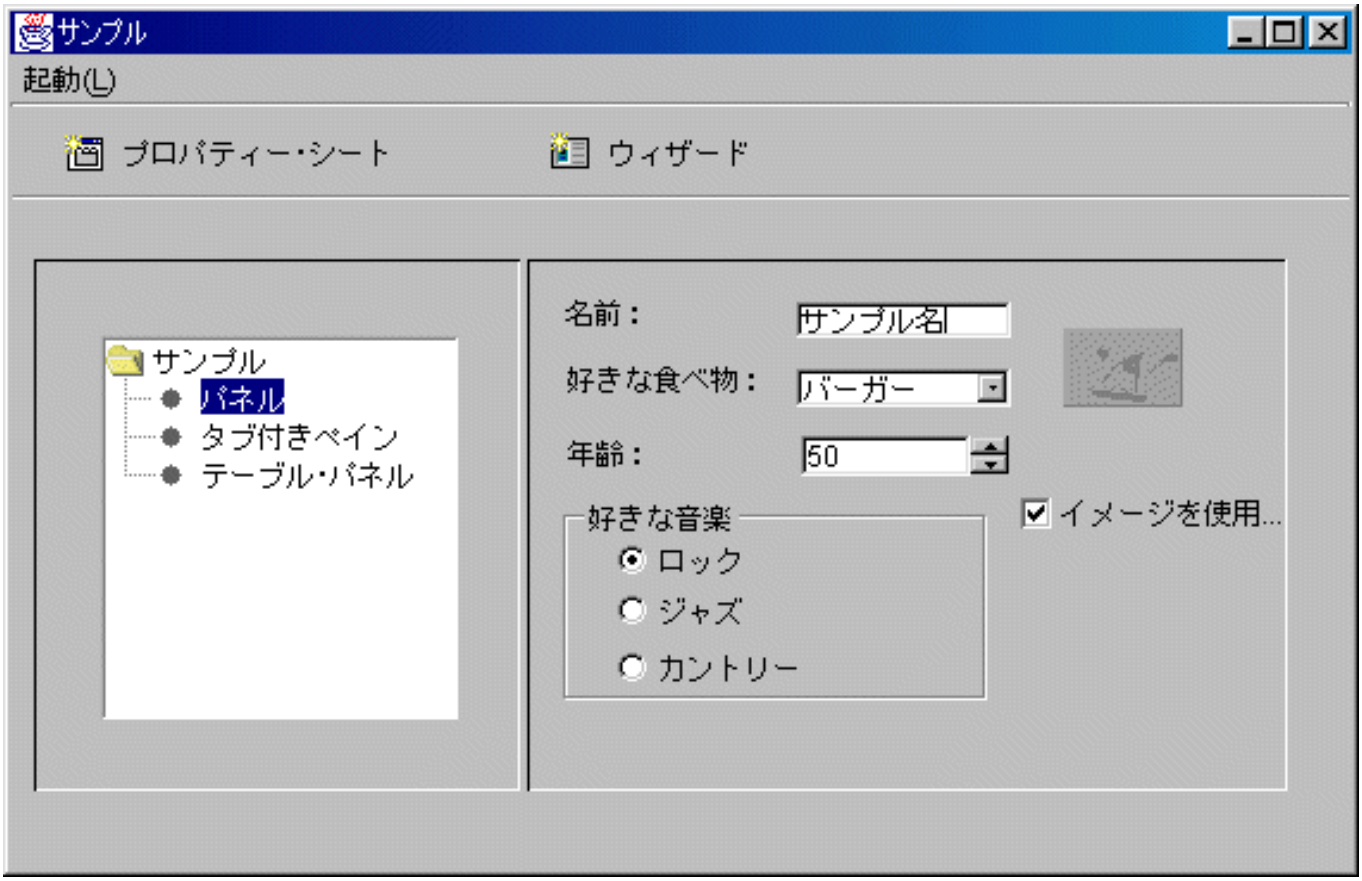
GUI ビルダーのサンプル・メイン・ウィンドウで、ツールバーの下の左側のペインから、その他の機能を選択することもできます。 図 15 は、左側のペインで「パネル」を選択すると、 どのように右側のペインにパネル・サンプルが表示されるかを示しています。

図 15: 左側のペインでの「パネル」の選択



パネル・サンプルは、イメージを使用不可にするためのオプション付きでプログラムされています。 イメージをグレー化して同じ画面を表示するには、「イメージを使用不可にする」を選択してください。

図 16: 右側のペインでの「イメージを使用不可にする」の選択



このパネル・サンプルでは、図 17 に示されているように、 ドロップダウン・リスト・ボックス・オプションも示されています。

図 17: 右側のペインでの「好きな食べ物」リストからの項目の選択

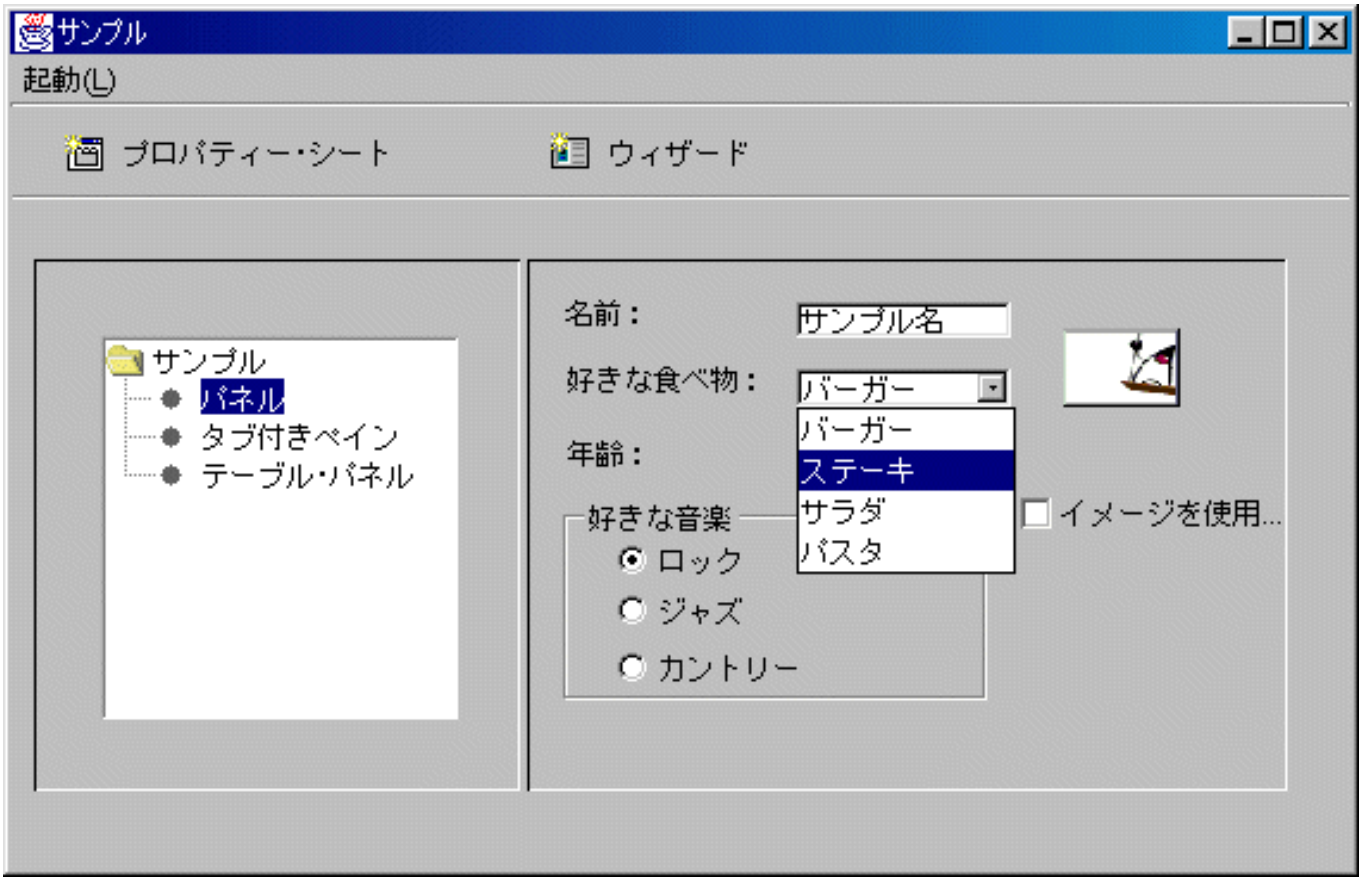


図 18 では、 GUI ビルダーのサンプル・メイン・ウィンドウの左側のペインでの「タブ付きペイン」の選択により、 どのように右側のペインにタブ付きペインのサンプルが表示されるかを示しています。

図 18: 左側のペインでの「タブ付きペイン」の選択

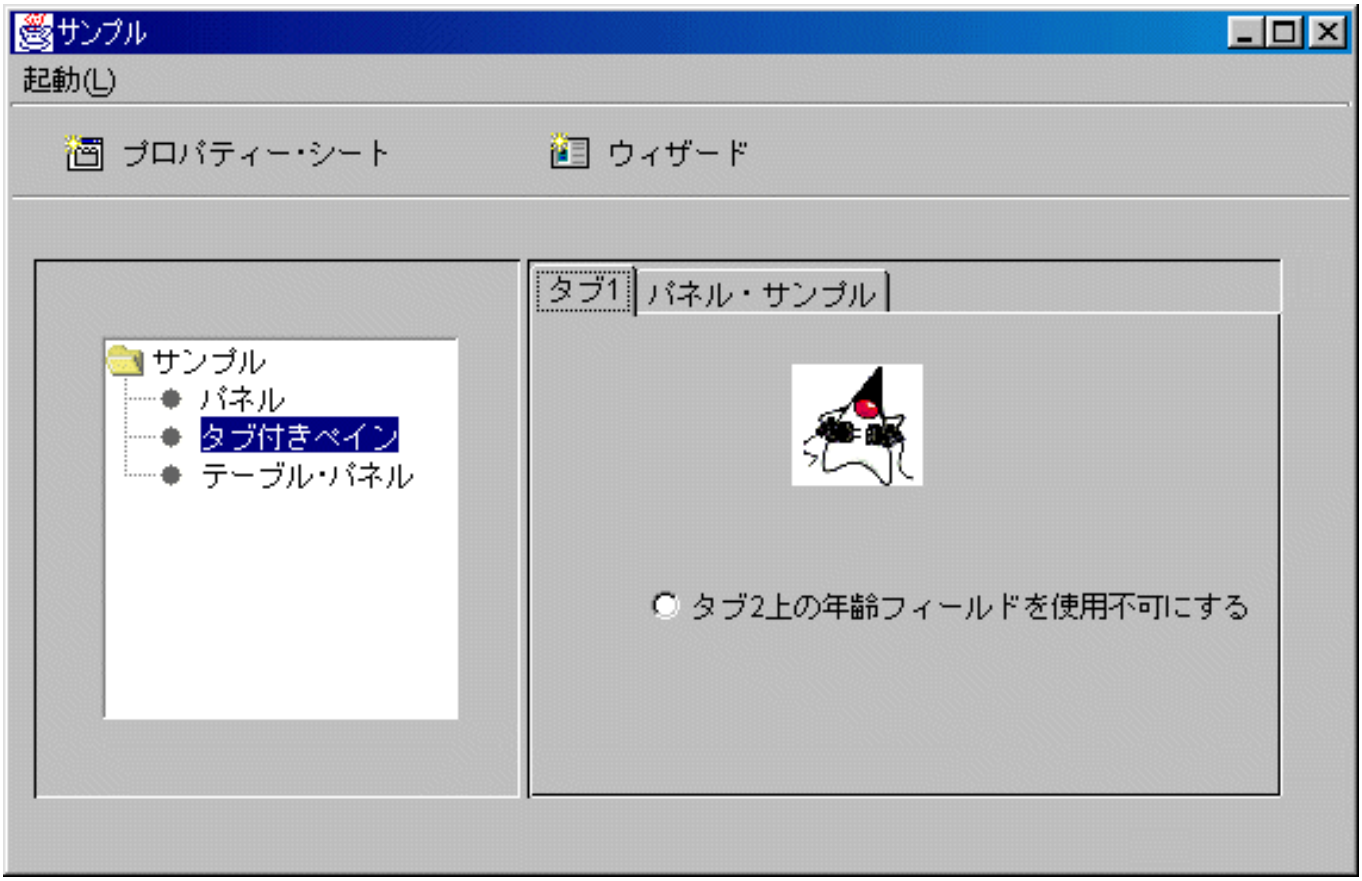
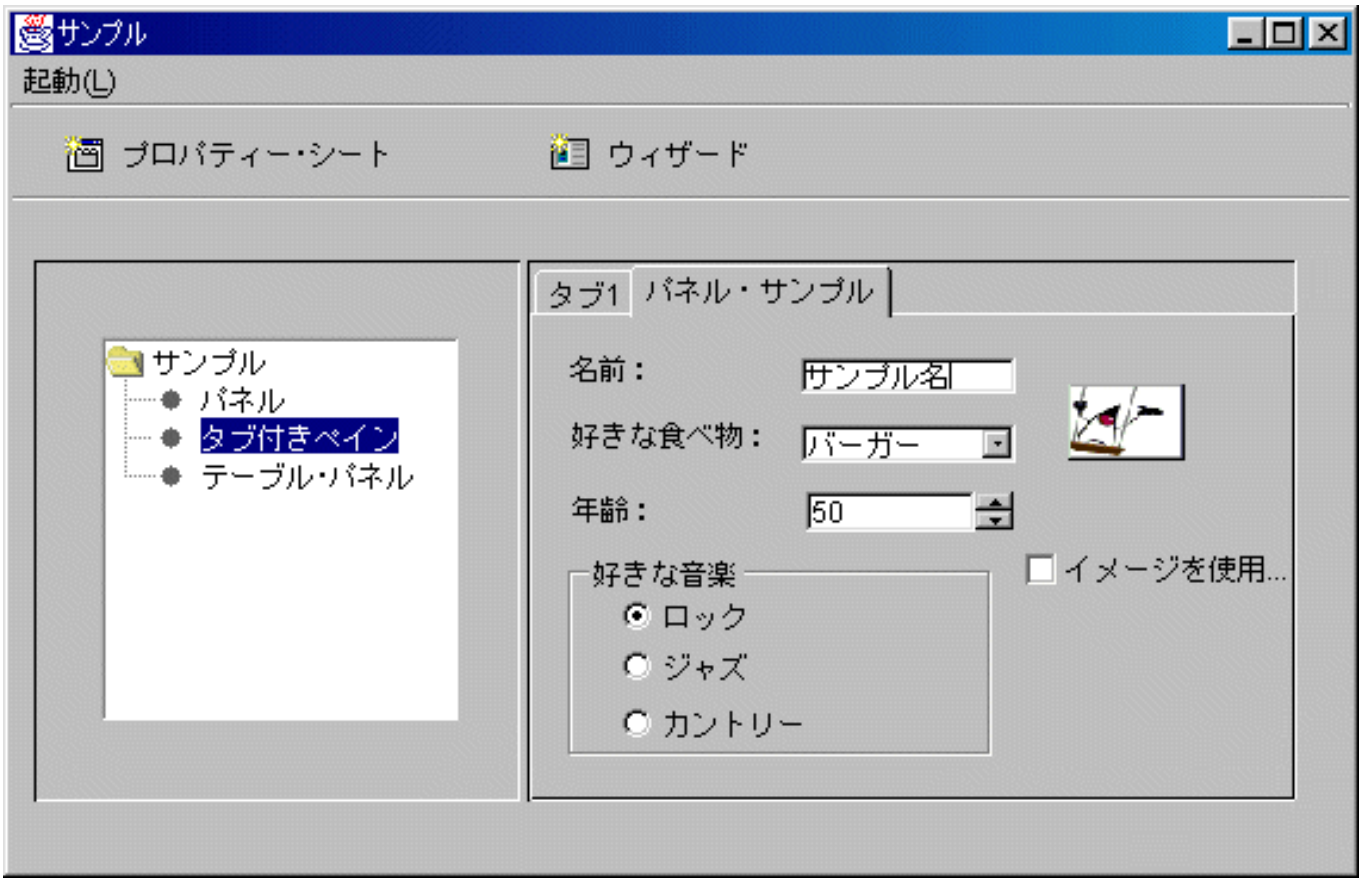


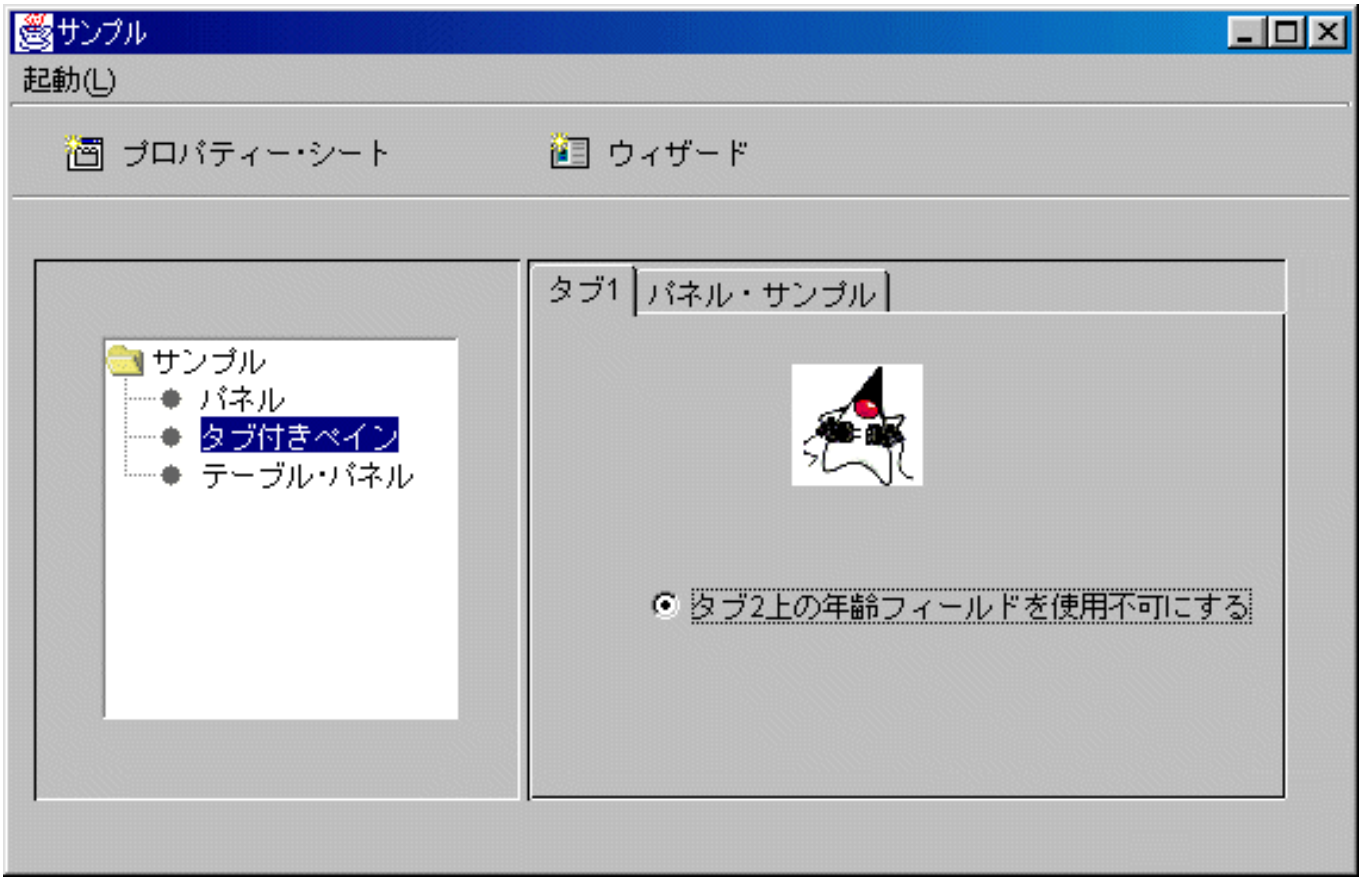
図 19 では、右側のペインでの「パネル・サンプル」タブの選択の結果を示しています。

図 19: 右側のペインでの「パネル・サンプル」タブの選択



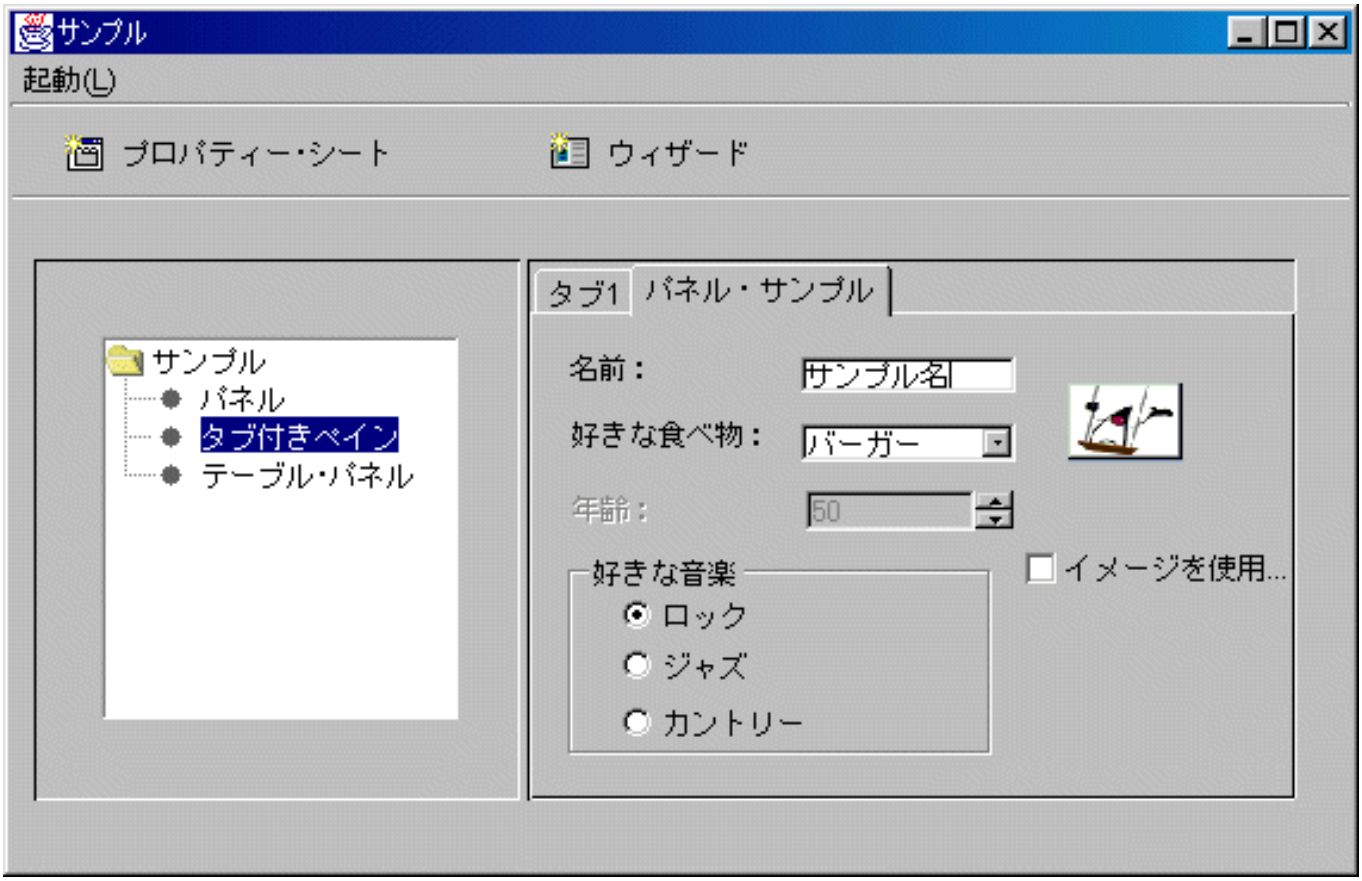
(右側のペインの)「タブ 1」を再度選択し、「タブ 2 上の年齢フィールドを使用不可にする」をクリックし、チェックを外します。

図 20: 右側のペインでの「タブ 2 上の年齢フィールドを使用不可にする」の選択



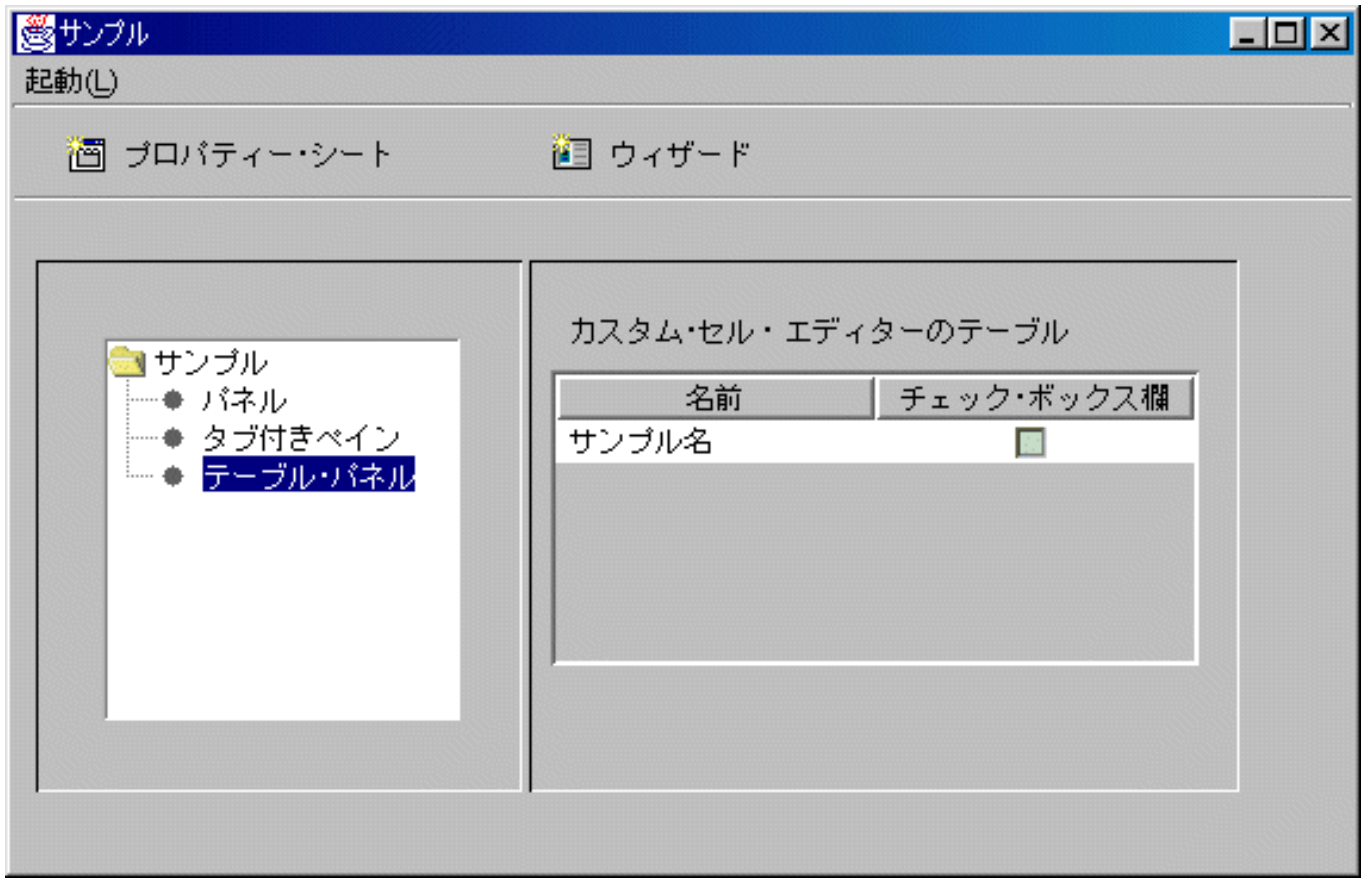
「タブ 2 上の年齢フィールドを使用不可にする」オプションを選択すると、 図 21 に示されているように、 「パネル・サンプル」タブの「年齢」フィールドが使用不能になり、 ぼかし表示になります。

図 21: 「パネル・サンプル」タブの「年齢」を使用不可にした結果



GUI ビルダー・サンプル・メインウィンドウの左側のペインで 「テーブル・パネル」を選択すると、 図 22 に示されているように、カスタム・レンダラーおよびカスタム・セル・エディターとともに、 テーブル・パネルの使用が示されます。

図 22: 左側のペインでの「テーブル・パネル」の選択


サンプル
起動(L)
プロパティー・シート
ウィザード
サンプル
パネル
タブ付きペイン
テーブル・パネル
カスタム・セル・エディターのテーブル
名前
チェック・ボックス欄
サンプル名

HTML クラスの例

以下の例では、HTML クラスのいくつかの使用法を示します。

- 例: BidiOrdering クラスを使用する
- 例: HTMLAlign オブジェクトを作成する
- 例: HTML フォーム・クラスを使用する
- 入力クラスの例:
  - 例: ButtonFormInput オブジェクトを作成する
  - 例: FileFormInput オブジェクトを作成する
  - 例: HiddenFormInput オブジェクトを作成する
  - 例: ImageFormInput オブジェクトを作成する
  - 例: ResetFormInput オブジェクトを作成する
  - 例: SubmitFormInput オブジェクトを作成する
  - 例: TextFormInput オブジェクトを作成する
  - 例: PasswordFormInput オブジェクトを作成する
  - 例: RadioFormInput オブジェクトを作成する
  - 例: CheckboxFormInput オブジェクトを作成する
- 例: HTMLHeading オブジェクトを作成する
- 例: HTMLHyperlink クラスを使用する
- 例: HTMLImage クラスを使用する
- HTMLList の例
  - 例: 順序リストを作成する
  - 例: 順不同リストを作成する
  - 例: ネストしたリストを作成する
- 例: HTMLMeta タグを作成する
- 例: HTMLParameter タグを作成する
- 例: HTMLServlet タグを作成する
- 例: HTMLText クラスを使用する
- HTMLTree の例
  - 例: HTMLTree クラスを使用する
  - 例: 走査可能な統合ファイル・システム・ツリーを作成する

- レイアウト・フォーム・クラス:
  - 例: GridLayoutFormPanel クラスを使用する
  - 例: LineLayoutFormPanel クラスを使用する
- 例: TextAreaFormElement クラスを使用する
- 例: LabelFormOutput クラスを使用する
- 例: SelectFormElement クラスを使用する
- 例: SelectOption クラスを使用する
- 例: RadioFormInputGroup クラスを使用する
- 例: RadioFormInput クラスを使用する
- 例: HTMLTable クラスを使用する
  - 例: HTMLTableCell クラスを使用する
  - 例: HTMLTableRow クラスを使用する
  - 例: HTMLTableHeader クラスを使用する
  - 例: HTMLTableCaption クラスを使用する

この例にあるように、 HTML クラスと servlet クラスを一緒に使用することもできます。

以下の特記事項は、IBM Toolbox for Java のすべての例に適用されます。

例: HTML フォーム・クラスを使用する

以下の例は、HTML フォーム・クラスを使用する方法を示しています。また、このコードを実行することによって、 出力例を表示することもできます。 "showHTML" メソッドで使用される HTML クラスは、 太字で表されています。

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
////////////////////////////////////////////////////////////////////////////
//
// This source is an example of using the IBM Toolbox for Java HTML
// package classes, which allow you to easily build HTML Forms.
//
////////////////////////////////////////////////////////////////////////////

package customer;

import java.io.*;
import java.awt.Color;

import javax.servlet.*;
import javax.servlet.http.*;

import com.ibm.as400.access.*;
import com.ibm.as400.util.html.*;


public class HTMLExample extends HttpServlet
{

  private static boolean found = false;         // Determines if user already exists in
                                                // the list of registrants.

  String regPath = "c://registration.txt";      // Registration information will be stored here


  public void init(ServletConfig config)
  {

    try
    {
       super.init(config);

    }
    catch(Exception e)
    {
       e.printStackTrace();
    }

  }

  /**
   *  Process the GET request.
   *  @param req The request.
   *  @param res The response.
   **/
   public void doGet (HttpServletRequest req, HttpServletResponse res)
      throws ServletException, IOException
   {
```

```
      res.setContentType("text/html");
      ServletOutputStream out = res.getOutputStream();

      // Display the Web using the new HTML classes
      out.println(showHTML());
      out.close();
   }


    public void doPost (HttpServletRequest req, HttpServletResponse res)
     throws ServletException, IOException
    {
        String nameStr   = req.getParameter("name");
        String emailStr = req.getParameter("email");
        String errorText= "";

        // Output stream to write to the servlet
        ServletOutputStream out = res.getOutputStream();

        res.setContentType("text/html");

        // Check name & e-mail parameters for valid values
        if (nameStr.length() == 0)
           errorText += "Customer Name not entered.  ";
        if (emailStr.length() == 0)
           errorText += "E-mail not entered.  " ;

        // If name & e-mail have both been provided, continue.
        if (errorText.length() == 0)
        {

           try
           {
              //Create the registration.txt file
              FileWriter f = new FileWriter(regPath, true);
              BufferedWriter output = new BufferedWriter(f);

              //buffered reader for searching the file
              BufferedReader in = new BufferedReader(new FileReader(regPath));

              String line = in.readLine();

              // reset the found flag
              found = false;

              // Check to see if this customer has already registered
              //  or has already used the same e-mail address
              while (!found)
              {
                 // if file is empty or end of file reached.
                 if (line == null)
                    break;

                 // if customer already registered
                 if ((line.equals("Customer Name: " + nameStr)) || (line.equals("Email address: " +
emailStr)))
                 {
                    // Output a message to the customer saying they have already
                    // registered
                    out.println ("<HTML> " +
```

```
                              "<TITLE> Toolbox Registration</TITLE> " +
                              "<META HTTP-EQUIV=/"pragma/" content=/"no-cache/"> " +
                              "<BODY BGCOLOR=/"blanchedalmond/" TEXT=/"black/"> " );
                  out.println ("<P><HR>" +
                              "<P>" + nameStr + "</B>, you have already registered using that " +
                              "<B>Name</B> or <B>E-mail address</B>." +
                              "<P> Thank You!...<P><HR>");

                  // Create a HTMLHyperlink object and display it
                  out.println ("<UL><LI>" + new HTMLHyperlink("./customer.HTMLExample", "Back to
Registration Form") + "</UL></BODY></HTML>");
                  found = true;
                  break;

               }
               else    // read the next line
                  line = in.readLine();

            }

            // String object to hold data submitted from the HTML Form
            String data;

            // If the users name or e-mail aren't found in our text file, continue.
            if (!found)
            {
               //-------------------------------------------------------------------------------
               // Insert the new customer info into a file
               output.newLine();
               output.write("Customer Name: " + nameStr);
               output.newLine();
               output.write("Email address: " + emailStr);
               output.newLine();
               //-------------------------------------------------------------------------------


               //-------------------------------------------------------------------------------
               //Getting "USE" checkbox from form
               data = req.getParameter("use");
               if (data != null)
               {
                  output.write("Currently Using Toolbox: " + data);
                  output.newLine();
               }
               //-------------------------------------------------------------------------------


               //-------------------------------------------------------------------------------
               //Getting "More Information" checkbox from form
               data = req.getParameter("contact");
               if (data != null)
               {
                  output.write("Requested More Information: " + data);
                  output.newLine();
               }
               //-------------------------------------------------------------------------------


               //-------------------------------------------------------------------------------
```

```
//Getting "AS400 Version" from form
data = req.getParameter("version");
if (data != null)
{
   if (data.equals("multiple versions"))
   {
      data = req.getParameter("MultiList");
      output.write("Multiple Versions: " + data);
   }
   else
      output.write("AS400 Version: " + data);

   output.newLine();
}
//--------------------------------------------------------------------------------


//--------------------------------------------------------------------------------
//Getting "Current Projects" from form
data = req.getParameter("interest");
if (data != null)
{
   output.write("Using Java or Interested In: " + data);
   output.newLine();
}
//--------------------------------------------------------------------------------


//--------------------------------------------------------------------------------
//Getting "Platforms" from form
data = req.getParameter("platform");
if (data != null)
{
   output.write("Platforms: " + data);
   output.newLine();
   if (data.indexOf("Other") >= 0)
   {
      output.write("Other Platforms: " + req.getParameter("OtherPlatforms"));
      output.newLine();
   }
}
//--------------------------------------------------------------------------------


//--------------------------------------------------------------------------------
//Getting "Number of iSeries or AS/400e servers" from form
data = req.getParameter("list1");
if (data != null)
{
   output.write("Number of iSeries servers: " + data);
   output.newLine();
}
//--------------------------------------------------------------------------------


//--------------------------------------------------------------------------------
//Getting "Comments" from form
data = req.getParameter("comments");
if (data != null && data.length() > 0)
{
```

```
                output.write("Comments: " + data);
                output.newLine();
             }
             //-------------------------------------------------------------------------


             //-------------------------------------------------------------------------
             //Getting "Attachment"
             data = req.getParameter("myAttachment");
             if (data != null && data.length() > 0)
             {
                output.write("Attachment File: " + data);
                output.newLine();
             }
             //-------------------------------------------------------------------------


             //-------------------------------------------------------------------------
             //Getting Hidden "Copyright" infomation
             data = req.getParameter("copyright");
             if (data != null)
             {
                output.write(data);
                output.newLine();
             }
             //-------------------------------------------------------------------------


             output.flush();
             output.close();

             // Print a thanks to the customer
             out.println("<HTML>");
             out.println("<TITLE>Thank You!</TITLE>");
             out.println("<META HTTP-EQUIV=/"pragma/" content=/"no-cache/"> ");
             out.println("<BODY BGCOLOR=/"blanchedalmond/">");
             out.println("<HR><P>Thank You for Registering, <B>" + nameStr + "</B>!<P><HR>");

             // Create a HTMLHyperlink object and display it
             out.println("<UL><LI>" + new HTMLHyperlink("./customer.HTMLExample", "Back to
Registration Form"));
             out.println("</UL></BODY></HTML>");

          }

       }
       catch (Exception e)
       {
          // Show error in browser
          out.println("<HTML>");
          out.println("<TITLE>ERROR!</TITLE>");
          out.println("<META HTTP-EQUIV=/"pragma/" content=/"no-cache/"> ");
          out.println("<BODY BGCOLOR=/"blanchedalmond/">");
          out.println("<BR><B>Error Message:</B><P>");
          out.println(e + "<P>");

          // Create a HTMLHyperlink object and display it
          out.println("<UL><LI>" + new HTMLHyperlink("./customer.HTMLExample", "Back to
Registration Form"));
```

```
            out.println("</UL></BODY></HTML>");

            e.printStackTrace();
         }
      }
      else
      {
         // Output a message to the customer saying customer name & e-mail not entered. Please
         // try again
         out.println ("<HTML> " +
                      "<TITLE>Invalid Registration Form</TITLE> " +
                      "<META HTTP-EQUIV=/"pragma/" content=/"no-cache/"> " +
                      "<BODY BGCOLOR=/"blanchedalmond/" TEXT=/"black/"> " );

         out.println ("<HR><B>ERROR</B> in customer data - <P><B>" +
                      errorText + "</B><P> Please Try Again... <HR>");


         // Create a HTMLHyperlink object and display it
         out.println("<UL><LI>" + new HTMLHyperlink("./customer.HTMLExample", "Back to Registration
Form") + "</UL></BODY></HTML>");
      }
       // Close the writer
       out.close();


   }




   public void destroy(ServletConfig config)
   {
      // do nothing
   }



   public String getServletInfo()
   {
       return "My Product Registration";
   }

   private String showHTML()
   {
     // String Buffer to hold HTML Page
     StringBuffer page = new StringBuffer();

     // Create the HTML Form object
     HTMLForm form = new HTMLForm("/servlet/customer.HTMLExample");;
     HTMLText txt;

     // Build the beginning of the HTML Page and add it to the String Buffer
     page.append("<HTML>/n");
     page.append("<TITLE> Welcome!!</TITLE>/n");
     page.append("<HEAD><SCRIPT LANGUAGE=/"JavaScript/">function test(){alert(/"This is a sample
script executed with a ButtonFormInput./")}</SCRIPT></HEAD>");
     page.append("<META HTTP-EQUIV=/"pragma/" content=/"no-cache/">/n");
     page.append("<BODY BGCOLOR=/"blanchedalmond/" TEXT=/"black/"><BR>/n");
```

```
try
{
   //-------------------------------------------------------------------------------------
   // Create page title using HTML Text
  txt = new HTMLText("Product Registration");
  txt.setSize(5);
  txt.setBold(true);
   txt.setColor(new Color(199, 21, 133));
   txt.setAlignment(HTMLConstants.CENTER);

   // Add HTML Text to the String Buffer
   page.append(txt.getTag(true) + "<HR><BR>/n");
   //-------------------------------------------------------------------------------------

   //-------------------------------------------------------------------------------------
   // Create a Line Layout
   LineLayoutFormPanel line = new LineLayoutFormPanel();
   txt = new HTMLText("Enter your name and e-mail address:");
   txt.setSize(4);
   line.addElement(txt);

   // Add the Line Layout to String Buffer
   page.append(line.toString());
   page.append("<BR>");
   //-------------------------------------------------------------------------------------

   //-------------------------------------------------------------------------------------
   // Set the HTML Form METHOD
  form.setMethod(HTMLForm.METHOD_POST);
   //-------------------------------------------------------------------------------------

   //-------------------------------------------------------------------------------------
   // Create a Text input for the name.
   TextFormInput user = new TextFormInput("name");
   user.setSize(25);
   user.setMaxLength(40);

   // Create a Text input for the email address.
   TextFormInput email = new TextFormInput("email");
   email.setSize(30);
   email.setMaxLength(40);

   // Create a ImageFormInput
   ImageFormInput img = new ImageFormInput("Submit Form", "..//images//myPiimages/c.gif");
   img.setAlignment(HTMLConstants.RIGHT);
   //-------------------------------------------------------------------------------------

   //-------------------------------------------------------------------------------------
   // Create a LineLayoutFormPanel object for the name & e-mail address
   LineLayoutFormPanel line2 = new LineLayoutFormPanel();

   // Add elements to the line form
   line2.addElement(new LabelFormElement("Name:"));
   line2.addElement(user);
   // Create and add a Label Element to the Line Layout
   line2.addElement(new LabelFormElement("E-mail:"));
   line2.addElement(email);
   line2.addElement(img);
   //-------------------------------------------------------------------------------------
```

```
        //-------------------------------------------------------------------------------------
        // Create Questions line layout
        LineLayoutFormPanel line3 = new LineLayoutFormPanel();

        // Add elements to the line layout
        line3.addElement(new LineLayoutFormPanel());
        line3.addElement(new CheckboxFormInput("use", "yes", "Do you currently use the Toolbox?",
false));
        line3.addElement(new LineLayoutFormPanel());
        line3.addElement(new CheckboxFormInput("contact", "yes", "Would you like information on
future Toolbox releases?", true));
        line3.addElement(new LineLayoutFormPanel());
        //-------------------------------------------------------------------------------------

        //-------------------------------------------------------------------------------------
        // Create Version Radio Group
        RadioFormInputGroup group = new RadioFormInputGroup("version");

        // Add Radio Form Inputs to the Group
        group.add(new RadioFormInput("version", "v3r2", "V3R2", false));
        group.add(new RadioFormInput("version", "v4r1", "V4R1", false));
        group.add(new RadioFormInput("version", "v4r2", "V4R2", false));
        group.add(new RadioFormInput("version", "v4r3", "V4R3", false));
        group.add(new RadioFormInput("version", "v4r4", "V4R4", false));
        group.add(new RadioFormInput("version", "multiple versions", "Multiple Versions? Which
ones:", false));

        //Create a Select Form Element
        SelectFormElement mlist = new SelectFormElement("MultiList");
        mlist.setMultiple(true);
        mlist.setSize(3);

        //Create the Options for the Select Form Element
        SelectOption option1 = mlist.addOption("V3R2", "v3r2");
        SelectOption option2 = mlist.addOption("V4R1", "v4r1");
        SelectOption option3 = mlist.addOption("V4R2", "v4r2");
        SelectOption option4 = mlist.addOption("V4R3", "v4r3");
        SelectOption option5 = mlist.addOption("V4R4", "v4r4");

        // Create HTML text
       txt = new HTMLText("Current Server Level:");
        txt.setSize(4);

        // Create Grid Layout
        GridLayoutFormPanel grid1 = new GridLayoutFormPanel(3);

        // Add radio group & select form element to the grid
        grid1.addElement(txt);
        grid1.addElement(group);
        grid1.addElement(mlist);
        //-------------------------------------------------------------------------------------

        //-------------------------------------------------------------------------------------
        // Create Grid Layout for interests
        GridLayoutFormPanel grid2 = new GridLayoutFormPanel(1);
        txt = new HTMLText("Current Projects or Area of Interest: (check all that apply)");
        txt.setSize(4);

        // Add elements to Grid Layout
```

```
 grid2.addElement(new LineLayoutFormPanel());
 grid2.addElement(txt);
 // Create and add a Checkbox to the Grid Layout
 grid2.addElement(new CheckboxFormInput("interest", "applications", "Applications", true));
 grid2.addElement(new CheckboxFormInput("interest", "applets", "Applets", false));
 grid2.addElement(new CheckboxFormInput("interest", "servlets", "Servlets", false));
 //-------------------------------------------------------------------------------------

 //-------------------------------------------------------------------------------------
 // Create Line Layout for platforms
 LineLayoutFormPanel line4 = new LineLayoutFormPanel();
 txt = new HTMLText("Client Platforms Used: (check all that apply)");
 txt.setSize(4);

 // Add elements to Line Layout
 line4.addElement(new LineLayoutFormPanel());
 line4.addElement(txt);
 line4.addElement(new LineLayoutFormPanel());
 line4.addElement(new CheckboxFormInput("platform", "95", "Windows95", false));
 line4.addElement(new CheckboxFormInput("platform", "98", "Windows98", false));
 line4.addElement(new CheckboxFormInput("platform", "NT", "WindowsNT", false));
 line4.addElement(new CheckboxFormInput("platform", "OS2", "OS/2", false));
 line4.addElement(new CheckboxFormInput("platform", "AIX", "AIX", false));
 line4.addElement(new CheckboxFormInput("platform", "Linux", "Linux", false));
 line4.addElement(new CheckboxFormInput("platform", "AS400", "iSeries", false));
 line4.addElement(new CheckboxFormInput("platform", "Other", "Other:", false));

 TextFormInput other = new TextFormInput("OtherPlatforms");
 other.setSize(20);
 other.setMaxLength(50);

 line4.addElement(other);
 //-------------------------------------------------------------------------------------

 //-------------------------------------------------------------------------------------
 // Create a Line Layout for number of servers
 LineLayoutFormPanel grid3 = new LineLayoutFormPanel();

txt = new HTMLText("How many iSeries or AS/400e servers do you have?   ");
 txt.setSize(4);

 // Create a Select Form Element for number of servers owned
 SelectFormElement list = new SelectFormElement("list1");
 // Create and add the Select Options to the Select Form Element List
 SelectOption opt0 = list.addOption("0", "zero");
 SelectOption opt1 = list.addOption("1", "one", true);
 SelectOption opt2 = list.addOption("2", "two");
 SelectOption opt3 = list.addOption("3", "three");
 SelectOption opt4 = list.addOption("4", "four");
 SelectOption opt5 = new SelectOption("5+", "FiveOrMore", false);
 list.addOption(opt5);

 // Add Elements to the Grid Layout
 grid3.addElement(new LineLayoutFormPanel());
 grid3.addElement(txt);
 grid3.addElement(list);
 //-------------------------------------------------------------------------------------

 //-------------------------------------------------------------------------------------
```

```
 // Create a Grid Layout for Product Comments
 GridLayoutFormPanel grid4 = new GridLayoutFormPanel(1);
 txt = new HTMLText("Product Comments:");
 txt.setSize(4);

 // Add elements to the Grid Layout
 grid4.addElement(new LineLayoutFormPanel());
 grid4.addElement(txt);
 //grid4.addElement(new LineLayoutFormPanel());
 // Create a Text Area Form
 grid4.addElement(new TextAreaFormElement("comments", 5, 75));
 grid4.addElement(new LineLayoutFormPanel());
 //-------------------------------------------------------------------------------------

 //-------------------------------------------------------------------------------------
 // Create a Grid Layout
 GridLayoutFormPanel grid5 = new GridLayoutFormPanel(2);
 txt = new HTMLText("Would you like to sign on to a server?");
 txt.setSize(4);

 // Create a Text input and Label for the system name.
 TextFormInput sys = new TextFormInput("system");
 LabelFormElement sysLabel = new LabelFormElement("System:");

 // Create a Text input and Label for the userid.
 TextFormInput uid = new TextFormInput("uid");
 LabelFormElement uidLabel = new LabelFormElement("UserID");

 // Create a Password input and Label for the password.
 PasswordFormInput pwd = new PasswordFormInput("pwd");
 LabelFormElement pwdLabel = new LabelFormElement("Password");

 // Add the Text inputs, password inputs, and Labels to the grid
 grid5.addElement(sysLabel);
 grid5.addElement(sys);
 grid5.addElement(uidLabel);
 grid5.addElement(uid);
 grid5.addElement(pwdLabel);
 grid5.addElement(pwd);
 //-------------------------------------------------------------------------------------


 //-------------------------------------------------------------------------------------
 // Add the various panels created to the HTML Form in the order you wish them to appear
form.addElement(line2);
 form.addElement(line3);
 form.addElement(grid1);
 form.addElement(grid2);
 form.addElement(line4);
 form.addElement(grid3);
 form.addElement(grid4);
 form.addElement(txt);
 form.addElement(new LineLayoutFormPanel());
 form.addElement(grid5);
 form.addElement(new LineLayoutFormPanel());
 form.addElement(new HTMLText("Submit an attachment Here: <br />"));
 // Add a File Input to the form
 form.addElement(new FileFormInput("myAttachment"));
 form.addElement(new ButtonFormInput("button", "TRY ME!", "test()"));
 // Adds a empty Line Layout, which in turn
```

```
            // adds a line break <br /> to the form
            form.addElement(new LineLayoutFormPanel());
            form.addElement(new LineLayoutFormPanel());
            form.addElement(new SubmitFormInput("submit", "Register"));
            form.addElement(new LineLayoutFormPanel());
            form.addElement(new LineLayoutFormPanel());
            form.addElement(new ResetFormInput("reset", "Reset"));

            // Add a Hidden Input to the form
            form.addElement(new HiddenFormInput("copyright", "(C) Copyright IBM Corp. 1999, 1999"));
            //-----------------------------------------------------------------------------------------

            // Add the entire HTML Form to the String Buffer
         page.append(form.toString());

         }
         catch(Exception e)
         {
            e.printStackTrace();
         }

         // Add the Ending HTML tags to the Buffer
         page.append("</BODY>/n");
         page.append("</HTML>/n");

         // Return the entire HTML page string
         return page.toString();

      }
   }
```

例: HTMLTree クラスを使用する

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
////////////////////////////////////////////////////////////////////////////
//
// This source is an example of using the IBM Toolbox for Java HTML
// package classes, which allow you to easily build HTML and File Trees.
//
////////////////////////////////////////////////////////////////////////////

import java.io.File;
import java.io.PrintWriter;
import java.io.IOException;

import java.util.Vector;
import java.util.Properties;

import javax.servlet.*;
import javax.servlet.http.*;

import com.ibm.as400.access.AS400;
import com.ibm.as400.access.Trace;
import com.ibm.as400.access.IFSJavaFile;
import com.ibm.as400.util.html.HTMLMeta;
import com.ibm.as400.util.html.HTMLTree;
import com.ibm.as400.util.html.HTMLTreeElement;
import com.ibm.as400.util.html.URLParser;
import com.ibm.as400.util.html.DirFilter;
import com.ibm.as400.util.html.FileTreeElement;
import com.ibm.as400.util.servlet.ServletHyperlink;

/**
 *  An example of using the HTMLTree and FileTreeElement classes in a servlet.
 **/
public class TreeNav extends HttpServlet
{
   public void init(ServletConfig config)
      throws ServletException
   {
      super.init(config);

      // The Toolbox uses a set of default icons to represents expanded, collapsed, and documents
within the HTMLTree.
      // To enhance those icons, the Toolbox ships three gifs (expanded.gif, collapsed.gif,
bullet.gif) in the
      // jt400Servlet.jar file.  Browsers do not have the ability to find gifs in a jar or zip file,
so those images
      // need to be extracted from the jar file and placed in the appropriate webserver directory (by
default it is the
      // /html directory).  Then uncomment the following lines of code and specify the correct
location in the these
      // set methods.  The location can be absolute or relative.

      HTMLTreeElement.setExpandedGif("/images/expanded.gif");
      HTMLTreeElement.setCollapsedGif("/images/collapsed.gif");
      HTMLTreeElement.setDocGif("/images/bullet.gif");
   }
```

```
/**
 *  Process the GET request.
 *  @param req The request.
 *  @param res The response.
 **/
public void doGet (HttpServletRequest req, HttpServletResponse resp)
   throws ServletException, IOException
{
   HttpSession session = req.getSession(true);
   HTMLTree fileTree = (HTMLTree)session.getValue("filetree");

   // If this session does not already have a file tree, then
   // create the initial tree.
   if (fileTree == null)
      fileTree = createTree(req, resp, req.getRequestURI());

   // Set the Http servlet request on the HTMLTree.
   fileTree.setHttpServletRequest(req);

   resp.setContentType("text/html");

 PrintWriter out = resp.getWriter();
 out.println("<html>/n");
   out.println(new HTMLMeta("Expires", "Mon, 03 Jan 1990 13:00:00 GMT"));
 out.println("<body>/n");

   // Get the tag for the HTMLTree.
   out.println(fileTree.getTag());

   out.println("</body>/n");
 out.println("</html>/n");
 out.close();

   // Set the session tree value, so when entering this servlet for
   // the second time, the FileTree object will be reused.
 session.putValue("filetree", fileTree);
}

/**
 *  Process the POST request.
 *  @param req The request.
 *  @param res The response.
 **/
public void doPost (HttpServletRequest req, HttpServletResponse res)
   throws ServletException, IOException
{
   res.setContentType("text/html");
   ServletOutputStream out = res.getOutputStream();
}


/**
 *  This method will create the initial HTMLTree.
 **/
private HTMLTree createTree(HttpServletRequest req, HttpServletResponse resp, String uri)
{
   // Create an HTMLTree object.
   HTMLTree tree = new HTMLTree(req);
```

```
try
{
     // Create a URLParser object.
     URLParser urlParser = new URLParser(uri);

     AS400 sys = new AS400(CPUStatus.systemName_, "javactl", "jteam1");

     // Create a File object and set the root IFS directory.
   IFSJavaFile root = new IFSJavaFile(sys, "/QIBM");

     // Create a Filter and list all of the directories.
   DirFilter filter = new DirFilter();
     //File[] dirList = root.listFiles(filter);

     // Get the list of files that satisfy the directory filter.
     String[] list = root.list(filter);

     File[] dirList = new File[list.length];

     // We don't want to require webservers to use JDK1.2 because
     // most webserver JVM's are slower to upgrade to the latest JDK level.
     // The most efficient way to create these file objects is to use
     // the listFiles(filter) method in JDK1.2 which would be done
     // like the following, instead of using the list(filter) method
     // and then converting the returned string arrary into the appropriate
     // File array.
     // File[] dirList = root.listFiles(filter);

     for (int j=0; j<dirList.length; ++j)
     {
        if (root instanceof IFSJavaFile)
           dirList[j] = new IFSJavaFile((IFSJavaFile)root, list[j]);
        else
           dirList[j] = new File(list[j]);
     }


     for (int i=0; i<dirList.length; i++)
   {
        // Create a FileTreeElement for each directory in the list.
        FileTreeElement node = new FileTreeElement(dirList[i]);

        // Create a ServletHyperlink for the expand/collapse icons.
        ServletHyperlink sl = new ServletHyperlink(urlParser.getURI());
        //sl.setHttpServletResponse(resp);
        node.setIconUrl(sl);


        // Create a ServletHyperlink to the TreeList servlet, which will
        // display the contents of thie FileTreeElement (directory).
    ServletHyperlink tl = new ServletHyperlink("/servlet/TreeList");
    tl.setTarget("list");

        // If the ServletHyperlink doesn't have a name, then set it to the
        // name of the directory.
        if (tl.getText() == null)
           tl.setText(dirList[i].getName());
```

```
            // Set the TextUrl for the FileTreeElement.
            node.setTextUrl(tl);

            // Add the FileTreeElement to the HTMLTree.
         tree.addElement(node);
        }
     }
     catch (Exception e)
     {
         e.printStackTrace();
     }

     return tree;
    }


     public void destroy(ServletConfig config)
     {
        // do nothing
     }

     public String getServletInfo()
     {
         return "FileTree Navigation";
     }
}
```

例: 走査可能な統合ファイル・システム・ツリーを作成する (ファイル 3 の 1)

このコード例は、他の 2 つのファイル例と併せて、サーブレット内の HTMLTree および FileListElement を示します。例を構成する 3 つのファイルは、次のとおりです。

- FileTreeExample.java - このファイル。 HTML フレームを生成して、サーブレットを開始します。
- TreeNav.java - ツリーを構築して管理します。
- TreeList.java - TreeNav.java クラス内で行われた選択の内容を表示します。

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
////////////////////////////////////////////////////////////////////////////
//
// This source is an example of using the IBM Toolbox for Java HTML package
// classes, which allow you to easily build HTML and File Trees.
//
////////////////////////////////////////////////////////////////////////////

import java.io.PrintWriter;
import java.io.IOException;

import javax.servlet.*;
import javax.servlet.http.*;

import com.ibm.as400.util.html.HTMLMeta;


//
// An example of using frames to display an HTMLTree and FileListElement
// in a servlet.
//

public class FileTreeExample extends HttpServlet
{
   public void init(ServletConfig config)
      throws ServletException
   {
      super.init(config);
   }

   /**
    *  Process the GET request.
    *  @param req The request.
    *  @param res The response.
    **/
```

```
   public void doGet (HttpServletRequest req, HttpServletResponse resp)
      throws ServletException, IOException
   {
      resp.setContentType("text/html");

      // Set up two frames. The first, a navigation frame, will display
      // the HTMLTree, which will contain FileTreeElements and allow
      // navigation of the File system.  The second frame will display/list
      // the contents of a selected directory from the navigation frame.
      PrintWriter out = resp.getWriter();
      out.println("<html>/n");
      out.println(new HTMLMeta("Expires","Mon, 04 Jan 1990 13:00:00 GMT"));
      out.println("<frameset cols=/"25%,*/">");
      out.println("<frame frameborder=/"5/" src=/"/servlet/TreeNav/" name=/"nav/">");
      out.println("<frame frameborder=/"3/" src=/"/servlet/TreeList/" name=/"list/">");
      out.println("</frameset>");
      out.println("</html>/n");
      out.close();
   }

   /**
    *  Process the POST request.
    *  @param req The request.
    *  @param res The response.
    **/

   public void doPost (HttpServletRequest req, HttpServletResponse res)
      throws ServletException, IOException
   {
      res.setContentType("text/html");
      ServletOutputStream out = res.getOutputStream();
   }

   public void destroy(ServletConfig config)
   {
      // do nothing
   }

   public String getServletInfo()
   {
      return "FileTree Servlet";
   }
}
```

例: 走査可能な統合ファイル・システム・ツリーを作成する (ファイル 3 の 2)

このコード例は、他の 2 つのファイル例と併せて、 サーブレット内の HTMLTree および FileListElement を示します。 例を構成する 3 つのファイルは、次のとおりです。

- FileTreeExample.java - HTML フレームを生成して、サーブレットを開始します。
- TreeNav.java - ツリーを構築して管理します。
- TreeList.java - TreeNav.java クラス内で行われた選択の内容を表示します。

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
////////////////////////////////////////////////////////////////////////////
//
// This source is an example of using the IBM Toolbox for Java HTML
// package classes, which allow you to easily build HTML and File Trees.
//
////////////////////////////////////////////////////////////////////////////

import java.io.File;
import java.io.PrintWriter;
import java.io.IOException;

import javax.servlet.*;
import javax.servlet.http.*;

import com.ibm.as400.access.AS400;
import com.ibm.as400.access.IFSJavaFile;
import com.ibm.as400.util.html.HTMLMeta;
import com.ibm.as400.util.html.HTMLTree;
import com.ibm.as400.util.html.HTMLTreeElement;
import com.ibm.as400.util.html.URLParser;
import com.ibm.as400.util.html.DirFilter;
import com.ibm.as400.util.html.FileTreeElement;
import com.ibm.as400.util.servlet.ServletHyperlink;

//
// An example of using the HTMLTree and FileTreeElement classes
// in a servlet.
//

public class TreeNav extends HttpServlet
{
   private AS400 sys_;

   public void init(ServletConfig config)
       throws ServletException
   {
```

```
    super.init(config);

    // Create an AS400 object.
    sys_ = new AS400("mySystem", "myUserID", "myPassword");

    // The Toolbox uses a set of default icons to represents expanded, collapsed,
    // and documents within the HTMLTree. To enhance those icons, the Toolbox ships
    // three gifs (expanded.gif, collapsed.gif, bullet.gif) in the jt400Servlet.jar
    // file.  Browsers do not have the ability to find gifs in a jar or zip file, so
    // you need to extract those images from the jar file and place them in the
    // appropriate webserver directory (by default it is the /html directory). Then
    // change the following lines of code to specify the correct location in the
    // set methods.  The location can be absolute or relative.

    HTMLTreeElement.setExpandedGif("http://myServer/expanded.gif");
    HTMLTreeElement.setCollapsedGif("http://myServer/collapsed.gif");
    HTMLTreeElement.setDocGif("http://myServer/bullet.gif");
}

/**
 *  Process the GET request.
 *  @param req The request.
 *  @param res The response.
 **/

public void doGet (HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException
{
    // Use session data to remember the state of the tree.
    HttpSession session = req.getSession(true);
    HTMLTree fileTree = (HTMLTree)session.getValue("filetree");

    // If this session does not already have a file tree, then
    // create the initial tree.
    if (fileTree == null)
        fileTree = createTree(req, resp, req.getRequestURI());

    // Set the Http servlet request on the HTMLTree.
    fileTree.setHttpServletRequest(req);

    resp.setContentType("text/html");

    PrintWriter out = resp.getWriter();
    out.println("<html>/n");
    out.println(new HTMLMeta("Expires","Mon, 03 Jan 1990 13:00:00 GMT"));
    out.println("<body>/n");

    // Get the tag for the HTMLTree.
    out.println(fileTree.getTag());
```

```
   out.println("</body>/n");
   out.println("</html>/n");
   out.close();

   // Set the session tree value, so when entering this servlet for
   // the second time, the FileTree object will be reused.
   session.putValue("filetree", fileTree);
}

/**
 *  Process the POST request.
 *  @param req The request.
 *  @param res The response.
 **/

public void doPost (HttpServletRequest req, HttpServletResponse res)
   throws ServletException, IOException
{
   res.setContentType("text/html");
   ServletOutputStream out = res.getOutputStream();
}

/**
 *  This method will create the initial HTMLTree.
 **/

private HTMLTree createTree(HttpServletRequest req,
                            HttpServletResponse resp, String uri)
{
   // Create an HTMLTree object.
   HTMLTree tree = new HTMLTree(req);

     try
     {
       // Create a URLParser object.
       URLParser urlParser = new URLParser(uri);

       // Create a File object and set the root IFS directory.
       IFSJavaFile root = new IFSJavaFile(sys_, "/QIBM");

       // Create a Filter.
       DirFilter filter = new DirFilter();

       // Get the list of files that satisfy the directory filter.
       String[] list = root.list(filter);

       File[] dirList = new File[list.length];

       // We don't want to require webservers to use JDK1.2 because
       // most webserver JVM's are slower to upgrade to the latest
```

```
  // JDK level. The most efficient way to create these file objects
  // is to use the listFiles(filter) method in JDK1.2 which would
  // be done like the following, instead of using the list(filter)
  // method and then converting the returned string arrary into the
  // appropriate File array.
  // File[] dirList = root.listFiles(filter);

  for (int j=0; j<dirList.length; ++j)
  {
    if (root instanceof IFSJavaFile)
       dirList[j] = new IFSJavaFile((IFSJavaFile)root, list[j]);
    else
       dirList[j] = new File(list[j]);
  }

  for (int i=0; i<dirList.length; i++)
  {
    // Create a FileTreeElement for each directory in the list.
    FileTreeElement node = new FileTreeElement(dirList[i]);

    // Create a ServletHyperlink for the expand/collapse icons.
    ServletHyperlink sl = new ServletHyperlink(urlParser.getURI());
    sl.setHttpServletResponse(resp);
    node.setIconUrl(sl);

    // Create a ServletHyperlink to the TreeList servlet, which will
    // display the contents of thie FileTreeElement (directory).
    ServletHyperlink tl = new ServletHyperlink("/servlet/TreeList");
    tl.setTarget("list");

    // If the ServletHyperlink doesn't have a name, then set it to the
    // name of the directory.
    if (tl.getText() == null)
      tl.setText(dirList[i].getName());

    // Set the TextUrl for the FileTreeElement.
    node.setTextUrl(tl);

    // Add the FileTreeElement to the HTMLTree.
    tree.addElement(node);
  }

  sys_.disconnectAllServices();
}

catch (Exception e)
{
e.printStackTrace();
}
```

```
        return tree;
    }

    public void destroy(ServletConfig config)
    {
       // do nothing
    }

    public String getServletInfo()
    {
        return "FileTree Navigation";
    }
}
```

例: 走査可能な統合ファイル・システム・ツリーを作成する (ファイル 3 の 3)

このコード例は、他の 2 つのファイル例と併せて、 サーブレット内の HTMLTree および FileListElement を示します。 例を構成する 3 つのファイルは、次のとおりです。

- FileTreeExample.java - HTML フレームを生成して、サーブレットを開始します。
- TreeNav.java - ツリーを構築して管理します。
- TreeList.java - このファイルは、TreeNav.java クラス内で行われた選択の内容を表示します。

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
////////////////////////////////////////////////////////////////////////////
//
// This source is an example of using the IBM Toolbox for Java HTML
// package classes, which allow you to easily build HTML and File Lists.
//
////////////////////////////////////////////////////////////////////////////

import java.io.PrintWriter;
import java.io.IOException;
import java.io.File;

import com.ibm.as400.access.AS400;
import com.ibm.as400.access.Trace;
import com.ibm.as400.access.IFSJavaFile;
import com.ibm.as400.util.html.HTMLMeta;
import com.ibm.as400.util.html.HTMLHeading;
import com.ibm.as400.util.html.HTMLConstants;
import com.ibm.as400.util.html.FileListElement;
import com.ibm.as400.util.html.*;

import javax.servlet.*;
import javax.servlet.http.*;

/**
 *  An example of using the FileListElement class in a servlet.
 **/
public class TreeList extends HttpServlet
{

    private AS400 sys_;

    /**
     *  Process the GET request.
     *  @param req The request.
     *  @param res The response.
     **/
    public void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException
    {
        resp.setContentType("text/html");

        try
        {

        PrintWriter out = resp.getWriter();
            out.println("<html>/n");
```

```
            out.println(new HTMLMeta("Expires", "Mon, 02 Jan 1990 13:00:00 GMT"));
            out.println("<body>/n");

            // If the path parameter is not null, then the user has selected an element from
            // the FileTreeElement list in the navigation frame.
            if (req.getPathInfo() != null)
            {
                // Create a FileListElement passing in an AS400 system object and the Http servlet
request.
                // The request will contain the necessary path information to list out the contents of
                // the FileTreeElement (directory) selected.
                FileListElement fileList = new FileListElement(sys_, req);

                //  Alternately, create a FileListElement from a NetServer share name and share path.
                //
                //  FileListElement fileList = new FileListElement(sys_, req, "TreeShare",
"/QIBM/ProdData/HTTP/Public/jt400");


                // Display the FileListElement contents.
                out.println(fileList.list());
            }
            else     // Display this HTMLHeading if no FileTreeElement has been selected.
            {
                HTMLHeading heading = new HTMLHeading(1,"An HTML File List Example");
                heading.setAlign(HTMLConstants.CENTER);

                out.println(heading.getTag());
            }

            out.println("</body>/n");
            out.println("</html>/n");
            out.close();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }

    }

    /**
      *  Process the POST request.
      *  @param req The request.
      *  @param res The response.
    **/
    public void doPost (HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
    {
        res.setContentType("text/html");
        ServletOutputStream out = res.getOutputStream();

    }

    public void init(ServletConfig config)
    throws ServletException
    {
        super.init(config);

        // Create an AS400 object.
```

```
        sys_ = new AS400("mySystem", "myUID", "myPWD");
    }
}
```

例: HTMLTable クラスを使用する

以下の例は、HTMLTable クラスが動作する方法を示しています。

```
// Create a default HTMLTable object.
HTMLTable table = new HTMLTable();

// Set the table attributes.
table.setAlignment(HTMLTable.CENTER);
table.setBorderWidth(1);

// Create a default HTMLTableCaption object and set the caption text.
HTMLTableCaption caption = new HTMLTableCaption();
caption.setElement("Customer Account Balances - January 1, 2000");

// Set the caption.
table.setCaption(caption);

// Create the table headers and add to the table.
HTMLTableHeader account_header = new HTMLTableHeader(new HTMLText("ACCOUNT"));
HTMLTableHeader name_header = new HTMLTableHeader(new HTMLText("NAME"));
HTMLTableHeader balance_header = new HTMLTableHeader(new HTMLText("BALANCE"));

table.addColumnHeader(account_header);
table.addColumnHeader(name_header);
table.addColumnHeader(balance_header);

// Add rows to the table.  Each customer record represents a row in the table.
int numCols = 3;
for (int rowIndex=0; rowIndex< numCustomers; rowIndex++)
{
   HTMLTableRow row = new HTMLTableRow();
   row.setHorizontalAlignment(HTMLTableRow.CENTER);

   HTMLText account = new HTMLText(customers[rowIndex].getAccount());
   HTMLText name = new HTMLText(customers[rowIndex].getName());
   HTMLText balance = new HTMLText(customers[rowIndex].getBalance());

   row.addColumn(new HTMLTableCell(account));
   row.addColumn(new HTMLTableCell(name));
   row.addColumn(new HTMLTableCell(balance));

   // Add the row to the table.
   table.addRow(row);
}
```

```
System.out.println(table.getTag());
```

上記の Java コード例は、以下の HTML コードを生成します。

```
<table align="center" border="1">
<caption>Customer Account Balances - January 1, 2000</caption>
<tr>
<th>ACCOUNT</th>
<th>NAME</th>
<th>BALANCE</th>
</tr>
<tr align="center">
<td>0000001</td>
<td>Customer1</td>
<td>100.00</td>
</tr>
<tr align="center">
<td>0000002</td>
<td>Customer2</td>
<td>200.00</td>
</tr>
<tr align="center">
<td>0000003</td>
<td>Customer3</td>
<td>550.00</td>
</tr>

</table>
```

Web ブラウザーでは上記の HTML コードは以下の表のように表示されます。

Customer Account Balances - January 1, 2000

ACCOUNT	NAME	BALANCE
0000001	Customer1	100.00
0000002	Customer2	200.00
0000003	Customer3	550.00

例: プログラム呼び出しマークアップ言語 (PCML)

以下の例は OS/400 API の呼び出しにプログラム呼び出しマークアップ言語を使用しており、 それぞれは、PCML ソースの後に Java プログラムを示したページにリンクしています。

- データ検索の簡単な例: サーバーのユーザー・プロファイルに関する情報を検索するのに必要な PCML ソースおよび Java プログラムを示します。 呼び出されている API は、*ユーザー情報の検索* (QSYRUSRI) API です。
- 情報リストの検索: サーバーの許可ユーザーのリストを検索するのに必要な PCML ソースと Java プログラムを示します。 呼び出されている API は、*許可ユーザー・リストのオープン* (QGYOLAUS) API です。 この例では、サーバー・プログラムによって戻される構造体の配列にアクセスする方法を示します。
- 多次元データの検索: サーバーからネットワーク・ファイル・システム (NFS) がエクスポートするリストを検索するのに必要な PCML ソースと Java プログラムを示します。 呼び出されている API は、*NFS エクスポートの検索* (QZNFRTVE) API です。 この例では、構造体の配列内にある構造体の配列にアクセスする方法を示します。

注: 適切な権限はそれぞれの例で異なりますが、 特定のオブジェクト権限や特殊権限が含まれる場合もあります。 ここで示す例を実行するには、 以下の事柄を行うための権限を所有するユーザー・プロファイルを使ってサインオンする必要があります。

- 例の中の OS/400 API の呼び出し
- 要求されている情報へのアクセス

以下の特記事項は、IBM Toolbox for Java のすべての例に適用されます。

ここに含まれるすべてのプログラムは、現存するままの状態で提供され、いかなる保証条件も適用されません。 第三者の権利の不侵害、商品性、特定目的適合性に関する黙示の保証の適用も一切ありません。

例: 簡単なデータ検索例

この簡単な例には次の 2 つの部分があります。

- QSYRUSRI を呼び出す PCML ソース
- QSYRUSRI を呼び出す Java プログラムのソース

QSYRUSRI を呼び出す PCML ソース

```
<pcml version="1.0">

<!-- PCML source for calling "Retrieve user Information" (QSYRUSRI) API -->

  <!-- Format USRI0150 - Other formats are available -->
  <struct name="usri0100">
    <data name="bytesReturned"              type="int"    length="4"  usage="output"/>
    <data name="bytesAvailable"             type="int"    length="4"  usage="output"/>
    <data name="userProfile"                type="char"   length="10" usage="output"/>
    <data name="previousSignonDate"         type="char"   length="7"  usage="output"/>
    <data name="previousSignonTime"         type="char"   length="6"  usage="output"/>
    <data                                   type="byte"   length="1"  usage="output"/>
    <data name="badSignonAttempts"          type="int"    length="4"  usage="output"/>
    <data name="status"                     type="char"   length="10" usage="output"/>
    <data name="passwordChangeDate"         type="byte"   length="8"  usage="output"/>
    <data name="noPassword"                 type="char"   length="1"  usage="output"/>
    <data                                   type="byte"   length="1"  usage="output"/>
    <data name="passwordExpirationInterval" type="int"    length="4"  usage="output"/>
    <data name="datePasswordExpires"        type="byte"   length="8"  usage="output"/>
    <data name="daysUntilPasswordExpires"   type="int"    length="4"  usage="output"/>
    <data name="setPasswordToExpire"        type="char"   length="1"  usage="output"/>
    <data name="displaySignonInfo"          type="char"   length="10" usage="output"/>
      </struct>

  <!-- Program QSYRUSRI and its parameter list for retrieving USRI0100 format -->
  <program name="qsyrusri" path="/QSYS.lib/QSYRUSRI.pgm">
    <data name="receiver"             type="struct"   struct="usri0100"   usage="output"/>
    <data name="receiverLength"       type="int"      length="4"          usage="input" />
    <data name="format"               type="char"     length="8"          usage="input"
init="USRI0100"/>
    <data name="profileName"          type="char"     length="10"         usage="input"
init="*CURRENT"/>
    <data name="errorCode"            type="int"      length="4"          usage="input"
init="0"/>
  </program>

</pcml>
```

QSYRUSRI を呼び出す Java プログラムのソース

```
import com.ibm.as400.data.ProgramCallDocument;
import com.ibm.as400.data.PcmlException;
import com.ibm.as400.access.AS400;
import com.ibm.as400.access.AS400Message;
```

```
// Example program to call "Retrieve User Information" (QSYRUSRI) API
public class qsyrusri {

    public qsyrusri() {
    }

    public static void main(String[] argv)
    {
        AS400 as400System;         // com.ibm.as400.access.AS400
        ProgramCallDocument pcml;  // com.ibm.as400.data.ProgramCallDocument
        boolean rc = false;        // Return code from ProgramCallDocument.callProgram()
        String msgId, msgText;     // Messages returned from the server
        Object value;              // Return value from ProgramCallDocument.getValue()

        System.setErr(System.out);

        // Construct AS400 without parameters, user will be prompted
        as400System = new AS400();

        try
        {
            // Uncomment the following to get debugging information
            //com.ibm.as400.data.PcmlMessageLog.setTraceEnabled(true);

            System.out.println("Beginning PCML Example..");
            System.out.println("    Constructing ProgramCallDocument for QSYRUSRI API...");

            // Construct ProgramCallDocument
            // First parameter is system to connect to
            // Second parameter is pcml resource name. In this example,
            // serialized PCML file "qsyrusri.pcml.ser" or
            // PCML source file "qsyrusri.pcml" must be found in the classpath.
            pcml = new ProgramCallDocument(as400System, "qsyrusri");

            // Set input parameters. Several parameters have default values
            // specified in the PCML source. Do not need to set them using Java code.
            System.out.println("    Setting input parameters...");
            pcml.setValue("qsyrusri.receiverLength", new
Integer((pcml.getOutputsize("qsyrusri.receiver"))));

            // Request to call the API
            // User will be prompted to sign on to the system
            System.out.println("    Calling QSYRUSRI API requesting information for the sign-on
user.");
            rc = pcml.callProgram("qsyrusri");

            // If return code is false, we received messages from the server
            if(rc == false)
            {
                // Retrieve list of server messages
                AS400Message[] msgs = pcml.getMessageList("qsyrusri");

                // Iterate through messages and write them to standard output
                for (int m = 0; m < msgs.length; m++)
                {
```

```
                    msgId = msgs[m].getID();
                    msgText = msgs[m].getText();
                    System.out.println("    " + msgId + " - " + msgText);
                }
                System.out.println("** Call to QSYRUSRI failed. See messages above **");
                System.exit(0);
            }
            // Return code was true, call to QSYRUSRI succeeded
            // Write some of the results to standard output
            else
            {
                value = pcml.getValue("qsyrusri.receiver.bytesReturned");
                System.out.println("        Bytes returned:      " + value);
                value = pcml.getValue("qsyrusri.receiver.bytesAvailable");
                System.out.println("        Bytes available:     " + value);
                value = pcml.getValue("qsyrusri.receiver.userProfile");
                System.out.println("        Profile name:        " + value);
                value = pcml.getValue("qsyrusri.receiver.previousSignonDate");
                System.out.println("        Previous signon date:" + value);
                value = pcml.getValue("qsyrusri.receiver.previousSignonTime");
                System.out.println("        Previous signon time:" + value);
            }
        }
        catch (PcmlException e)
        {
            System.out.println(e.getLocalizedMessage());
            e.printStackTrace();
            System.out.println("*** Call to QSYRUSRI failed. ***");
            System.exit(0);
        }

        System.exit(0);
    } // End main()

}
```

例: 情報リストの検索

この例には次の 2 つの部分があります。

- QGYOLAUS を呼び出す PCML ソース
- QGYOLAUS を呼び出す Java プログラムのソース

QGYOLAUS を呼び出す PCML ソース

```
<pcml version="1.0">

<!-- PCML source for calling "Open List of Authorized Users" (QGYOLAUS) API -->

  <!-- Format AUTU0150 - Other formats are available -->
  <struct name="autu0150">
    <data name="name"         type="char" length="10" />
    <data name="userOrGroup"  type="char" length="1"  />
    <data name="groupMembers" type="char" length="1"  />
    <data name="description"  type="char" length="50" />
  </struct>

  <!-- List information structure (common for "Open List" type APIs) -->
  <struct name="listInfo">
    <data name="totalRcds"    type="int"  length="4" />
    <data name="rcdsReturned" type="int"  length="4" />
    <data name="rqsHandle"    type="byte" length="4" />
    <data name="rcdLength"    type="int"  length="4" />
    <data name="infoComplete" type="char" length="1" />
    <data name="dateCreated"  type="char" length="7" />
    <data name="timeCreated"  type="char" length="6" />
    <data name="listStatus"   type="char" length="1" />
    <data                     type="byte" length="1" />
    <data name="lengthOfInfo" type="int"  length="4" />
    <data name="firstRecord"  type="int"  length="4" />
    <data                     type="byte" length="40" />
  </struct>

  <!-- Program QGYOLAUS and its parameter list for retrieving AUTU0150 format -->
  <program name="qgyolaus" path="/QSYS.lib/QGY.lib/QGYOLAUS.pgm" parseorder="listInfo receiver">
    <data   name="receiver"       type="struct" struct="autu0150" usage="output"
              count="listInfo.rcdsReturned" outputsize="receiverLength" />
    <data   name="receiverLength" type="int"    length="4"        usage="input" init="16384" />
    <data   name="listInfo"       type="struct" struct="listInfo" usage="output" />
    <data   name="rcdsToReturn"   type="int"    length="4"        usage="input" init="264" />
    <data   name="format"         type="char"   length="10"       usage="input" init="AUTU0150" />
    <data   name="selection"      type="char"   length="10"       usage="input" init="*USER" />
    <data   name="member"         type="char"   length="10"       usage="input" init="*NONE" />
    <data   name="errorCode"      type="int"    length="4"        usage="input" init="0" />

  </program>

  <!-- Program QGYGTLE returned additional "records" from the list
       created by QGYOLAUS. -->
  <program name="qgygtle" path="/QSYS.lib/QGY.lib/QGYGTLE.pgm" parseorder="listInfo receiver">
    <data   name="receiver"       type="struct" struct="autu0150" usage="output"
            count="listInfo.rcdsReturned" outputsize="receiverLength" />
```

```
    <data   name="receiverLength" type="int"    length="4"        usage="input" init="16384" />
    <data   name="requestHandle"  type="byte"   length="4"        usage="input" />
    <data   name="listInfo"       type="struct" struct="listInfo" usage="output" />
    <data   name="rcdsToReturn"   type="int"    length="4"        usage="input" init="264" />
    <data   name="startingRcd"    type="int"    length="4"        usage="input" />
    <data   name="errorCode"      type="int"    length="4"        usage="input" init="0" />
  </program>

  <!-- Program QGYCLST closes the list, freeing resources on the server -->
  <program name="qgyclst" path="/QSYS.lib/QGY.lib/QGYCLST.pgm" >
    <data   name="requestHandle"  type="byte"   length="4"        usage="input" />
    <data   name="errorCode"      type="int"    length="4"        usage="input" init="0" />
  </program>
</pcml>
```

QGYOLAUS を呼び出す Java プログラムのソース

```
import com.ibm.as400.data.ProgramCallDocument;
import com.ibm.as400.data.PcmlException;
import com.ibm.as400.access.AS400;
import com.ibm.as400.access.AS400Message;

// Example program to call "Retrieve List of Authorized Users" (QGYOLAUS) API
public class qgyolaus
{

  public static void main(String[] argv)
  {
    AS400 as400System;         // com.ibm.as400.access.AS400
    ProgramCallDocument pcml;  // com.ibm.as400.data.ProgramCallDocument
    boolean rc = false;        // Return code from ProgramCallDocument.callProgram()
    String msgId, msgText;     // Messages returned from the server
    Object value;              // Return value from ProgramCallDocument.getValue()

    int[] indices = new int[1]; // Indices for access array value
    int nbrRcds,                // Number of records returned from QGYOLAUS and QGYGTLE
        nbrUsers;               // Total number of users retrieved
    String listStatus;          // Status of list on the server
    byte[] requestHandle = new byte[4];

    System.setErr(System.out);

    // Construct AS400 without parameters, user will be prompted
    as400System = new AS400();

    try
    {
      // Uncomment the following to get debugging information
      //com.ibm.as400.data.PcmlMessageLog.setTraceEnabled(true);

      System.out.println("Beginning PCML Example..");
      System.out.println("    Constructing ProgramCallDocument for QGYOLAUS API...");

      // Construct ProgramCallDocument
      // First parameter is system to connect to
      // Second parameter is pcml resource name. In this example,
```

```
   // serialized PCML file "qgyolaus.pcml.ser" or
   // PCML source file "qgyolaus.pcml" must be found in the classpath.
   pcml = new ProgramCallDocument(as400System, "qgyolaus");

   // All input parameters have default values specified in the PCML source.
   // Do not need to set them using Java code.

   // Request to call the API
   // User will be prompted to sign on to the system
   System.out.println("    Calling QGYOLAUS API requesting information for the sign-on user.");
   rc = pcml.callProgram("qgyolaus");

   // If return code is false, we received messages from the server
   if(rc == false)
   {
     // Retrieve list of server messages
     AS400Message[] msgs = pcml.getMessageList("qgyolaus");

     // Iterate through messages and write them to standard output
     for (int m = 0; m < msgs.length; m++)
     {
         msgId = msgs[m].getID();
         msgText = msgs[m].getText();
         System.out.println("    " + msgId + " - " + msgText);
     }
     System.out.println("** Call to QGYOLAUS failed. See messages above **");
   System.exit(0);
   }
   // Return code was true, call to QGYOLAUS succeeded
   // Write some of the results to standard output
else
   {
     boolean doneProcessingList = false;
     String programName = "qgyolaus";
     nbrUsers = 0;
     while (!doneProcessingList)
     {
       nbrRcds = pcml.getIntValue(programName + ".listInfo.rcdsReturned");
       requestHandle = (byte[]) pcml.getValue(programName + ".listInfo.rqsHandle");

       // Iterate through list of users
       for (indices[0] = 0; indices[0] < nbrRcds; indices[0]++)
       {
            value = pcml.getValue(programName + ".receiver.name", indices);
            System.out.println("User:  " + value);

            value = pcml.getValue(programName + ".receiver.description", indices);
            System.out.println("/t/t" + value);
       }

       nbrUsers += nbrRcds;

       // See if we retrieved all the users.
       // If not, subsequent calls to "Get List Entries" (QGYGTLE)
       // would need to be made to retrieve the remaining users in the list.
       listStatus = (String) pcml.getValue(programName + ".listInfo.listStatus");
       if ( listStatus.equals("2")   // List is marked as "Complete"
```

```
            || listStatus.equals("3") ) // Or list is marked "Error building"
          {
            doneProcessingList = true;
          }
          else
          {
            programName = "qgygtle";

            // Set input parameters for QGYGTLE
            pcml.setValue("qgygtle.requestHandle", requestHandle);
            pcml.setIntValue("qgygtle.startingRcd", nbrUsers + 1);

            // Call "Get List Entries" (QGYGTLE) to get more users from list
            rc = pcml.callProgram("qgygtle");

            // If return code is false, we received messages from the server
            if(rc == false)
            {
              // Retrieve list of server messages
              AS400Message[] msgs = pcml.getMessageList("qgygtle");

              // Iterate through messages and write them to standard output
              for (int m = 0; m < msgs.length; m++)
              {
                  msgId = msgs[m].getID();
                  msgText = msgs[m].getText();
                  System.out.println("    " + msgId + " - " + msgText);
              }
              System.out.println("** Call to QGYGTLE failed. See messages above **");
              System.exit(0);
            }
            // Return code was true, call to QGYGTLE succeeded

          }
        }
        System.out.println("Number of users returned:  " + nbrUsers);

        // Call the "Close List" (QGYCLST) API
        pcml.setValue("qgyclst.requestHandle", requestHandle);
        rc = pcml.callProgram("qgyclst");
      }
    }
    catch(PcmlException e)
    {
      System.out.println(e.getLocalizedMessage());
      e.printStackTrace();
      System.out.println("*** Call to QGYOLAUS failed. ***");
      System.exit(0);
    }

    System.exit(0);
  }
}
```

例: 多次元データの検索

この例には次の 2 つの部分があります。

- QZNFRTVE を呼び出す PCML ソース
- QZNFRTVE を呼び出す Java プログラムのソース

QZNFRTVE を呼び出す PCML ソース

```
<pcml version="1.0">

  <struct name="receiver">
    <data name="lengthOfEntry"            type="int"  length="4" />
    <data name="dispToObjectPathName"     type="int"  length="4" />
    <data name="lengthOfObjectPathName"   type="int"  length="4" />
    <data name="ccsidOfObjectPathName"    type="int"  length="4" />
    <data name="readOnlyFlag"             type="int"  length="4" />
    <data name="nosuidFlag"               type="int"  length="4" />
    <data name="dispToReadWriteHostNames" type="int"  length="4" />
    <data name="nbrOfReadWriteHostNames"  type="int"  length="4" />
    <data name="dispToRootHostNames"      type="int"  length="4" />
    <data name="nbrOfRootHostNames"       type="int"  length="4" />
    <data name="dispToAccessHostNames"    type="int"  length="4" />
    <data name="nbrOfAccessHostNames"     type="int"  length="4" />
    <data name="dispToHostOptions"        type="int"  length="4" />
    <data name="nbrOfHostOptions"         type="int"  length="4" />
    <data name="anonUserID"               type="int"  length="4" />
    <data name="anonUsrPrf"               type="char" length="10" />
    <data name="pathName"                 type="char" length="lengthOfObjectPathName"
          offset="dispToObjectPathName" offsetfrom="receiver" />

    <struct name="rwAccessList" count="nbrOfReadWriteHostNames"
            offset="dispToReadWriteHostNames" offsetfrom="receiver">
      <data name="lengthOfEntry"          type="int"  length="4" />
      <data name="lengthOfHostName"       type="int"  length="4" />
      <data name="hostName"               type="char" length="lengthOfHostName" />
      <data                               type="byte" length="0"
            offset="lengthOfEntry" />
    </struct>

    <struct name="rootAccessList" count="nbrOfRootHostNames"
            offset="dispToRootHostNames" offsetfrom="receiver">
      <data name="lengthOfEntry"          type="int"  length="4" />
      <data name="lengthOfHostName"       type="int"  length="4" />
      <data name="hostName"               type="char" length="lengthOfHostName" />
      <data                               type="byte" length="0"
            offset="lengthOfEntry" />
    </struct>

    <struct name="accessHostNames" count="nbrOfAccessHostNames"
            offset="dispToAccessHostNames" offsetfrom="receiver" >
      <data name="lengthOfEntry"          type="int"  length="4" />
      <data name="lengthOfHostName"       type="int"  length="4" />
      <data name="hostName"               type="char" length="lengthOfHostName" />
      <data                               type="byte" length="0"
            offset="lengthOfEntry" />
    </struct>
```

```
    <struct name="hostOptions" offset="dispToHostOptions" offsetfrom="receiver"
count="nbrOfHostOptions">
      <data name="lengthOfEntry"          type="int"  length="4" />
      <data name="dataFileCodepage"       type="int"  length="4" />
      <data name="pathNameCodepage"       type="int"  length="4" />
      <data name="writeModeFlag"          type="int"  length="4" />
      <data name="lengthOfHostName"       type="int"  length="4" />
      <data name="hostName"               type="char" length="lengthOfHostName" />
      <data                               type="byte" length="0"
            offset="lengthOfEntry" />
    </struct>

    <data type="byte" length="0" offset="lengthOfEntry" />
  </struct>

  <struct name="returnedRcdsFdbkInfo">
    <data name="bytesReturned"            type="int" length="4" />
    <data name="bytesAvailable"           type="int" length="4" />
    <data name="nbrOfNFSExportEntries"    type="int" length="4" />
    <data name="handle"                   type="int" length="4" />
  </struct>

  <program name="qznfrtve" path="/QSYS.lib/QZNFRTVE.pgm" parseorder="returnedRcdsFdbkInfo receiver" >
    <data name="receiver"             type="struct" struct="receiver" usage="output"
              count="returnedRcdsFdbkInfo.nbrOfNFSExportEntries" outputsize="receiverLength"/>
    <data name="receiverLength"       type="int"    length="4" usage="input" init="4096" />
    <data name="returnedRcdsFdbkInfo" type="struct" struct="returnedRcdsFdbkInfo" usage="output" />
    <data name="formatName"           type="char"   length="8" usage="input" init="EXPE0100" />
    <data name="objectPathName"       type="char"   length="lengthObjPathName" usage="input"
init="*FIRST" />
    <data name="lengthObjPathName"    type="int"    length="4" usage="input" init="6" />
    <data name="ccsidObjectPathName"  type="int"    length="4" usage="input" init="0" />
    <data name="desiredCCSID"         type="int"    length="4" usage="input" init="0" />
    <data name="handle"               type="int"    length="4" usage="input" init="0" />
    <data name="errorCode"            type="int"    length="4" usage="input" init="0" />
  </program>

</pcml>
```

QZNFRTVE を呼び出す Java プログラムのソース

```
import com.ibm.as400.data.ProgramCallDocument;
import com.ibm.as400.data.PcmlException;
import com.ibm.as400.access.AS400;
import com.ibm.as400.access.AS400Message;

// Example program to call "Retrieve NFS Exports" (QZNFRTVE) API
public class qznfrtve
{
  public static void main(String[] argv)
  {
    AS400 as400System;         // com.ibm.as400.access.AS400
    ProgramCallDocument pcml;  // com.ibm.as400.data.ProgramCallDocument
    boolean rc = false;        // Return code from ProgramCallDocument.callProgram()
    String msgId, msgText;     // Messages returned from the server
    Object value;              // Return value from ProgramCallDocument.getValue()

    System.setErr(System.out);
```

```
    // Construct AS400 without parameters, user will be prompted
    as400System = new AS400();

    int[] indices = new int[2]; // Indices for access array value
    int nbrExports;             // Number of exports returned
    int nbrOfReadWriteHostNames, nbrOfRWHostNames,
        nbrOfRootHostNames,      nbrOfAccessHostnames, nbrOfHostOpts;

    try
    {
      // Uncomment the following to get debugging information
      // com.ibm.as400.data.PcmlMessageLog.setTraceEnabled(true);

      System.out.println("Beginning PCML Example..");
      System.out.println("    Constructing ProgramCallDocument for QZNFRTVE API...");

      // Construct ProgramCallDocument
      // First parameter is system to connect to
      // Second parameter is pcml resource name. In this example,
      // serialized PCML file "qznfrtve.pcml.ser" or
      // PCML source file "qznfrtve.pcml" must be found in the classpath.
      pcml = new ProgramCallDocument(as400System, "qznfrtve");

      // Set input parameters. Several parameters have default values
      // specified in the PCML source. Do not need to set them using Java code.
      System.out.println("    Setting input parameters...");
      pcml.setValue("qznfrtve.receiverLength", new Integer( (
pcml.getOutputsize("qznfrtve.receiver"))));

      // Request to call the API
      // User will be prompted to sign on to the system
      System.out.println("    Calling QZNFRTVE API requesting NFS exports.");
      rc = pcml.callProgram("qznfrtve");

      if (rc == false)
      {
        // Retrieve list of server messages
        AS400Message[] msgs = pcml.getMessageList("qznfrtve");

        // Iterate through messages and write them to standard output
        for (int m = 0; m < msgs.length; m++)
        {
            msgId = msgs[m].getID();
            msgText = msgs[m].getText();
            System.out.println("    " + msgId + " - " + msgText);
        }
        System.out.println("** Call to QZNFRTVE failed. See messages above **");
        System.exit(0);
      }
      // Return code was true, call to QZNFRTVE succeeded
      // Write some of the results to standard output
      else
      {
        nbrExports = pcml.getIntValue("qznfrtve.returnedRcdsFdbkInfo.nbrOfNFSExportEntries");
        // Iterate through list of exports
        for (indices[0] = 0; indices[0] < nbrExports; indices[0]++)
        {
          value = pcml.getValue("qznfrtve.receiver.pathName", indices);
```

```
          System.out.println("Path name = " + value);

          // Iterate and write out Read Write Host Names for this export
          nbrOfReadWriteHostNames = pcml.getIntValue("qznfrtve.receiver.nbrOfReadWriteHostNames",
indices);
          for(indices[1] = 0; indices[1] < nbrOfReadWriteHostNames; indices[1]++)
          {
            value = pcml.getValue("qznfrtve.receiver.rwAccessList.hostName", indices);
            System.out.println("    Read/write access host name = " + value);
          }

          // Iterate and write out Root Host Names for this export
          nbrOfRootHostNames = pcml.getIntValue("qznfrtve.receiver.nbrOfRootHostNames", indices);
          for(indices[1] = 0; indices[1] < nbrOfRootHostNames; indices[1]++)
          {
            value = pcml.getValue("qznfrtve.receiver.rootAccessList.hostName", indices);
            System.out.println("    Root access host name = " + value);
          }

          // Iterate and write out Access Host Names for this export
          nbrOfAccessHostnames = pcml.getIntValue("qznfrtve.receiver.nbrOfAccessHostNames", indices);
          for(indices[1] = 0; indices[1] < nbrOfAccessHostnames; indices[1]++)
          {
            value = pcml.getValue("qznfrtve.receiver.accessHostNames.hostName", indices);
            System.out.println("    Access host name = " + value);
          }

          // Iterate and write out Host Options for this export
          nbrOfHostOpts = pcml.getIntValue("qznfrtve.receiver.nbrOfHostOptions", indices);
          for(indices[1] = 0; indices[1] < nbrOfHostOpts; indices[1]++)
          {
            System.out.println("    Host options:");
            value = pcml.getValue("qznfrtve.receiver.hostOptions.dataFileCodepage", indices);
            System.out.println("        Data file code page = " + value);
            value = pcml.getValue("qznfrtve.receiver.hostOptions.pathNameCodepage", indices);
            System.out.println("        Path name code page = " + value);
            value = pcml.getValue("qznfrtve.receiver.hostOptions.writeModeFlag", indices);
            System.out.println("        Write mode flag = " + value);
            value = pcml.getValue("qznfrtve.receiver.hostOptions.hostName", indices);
            System.out.println("        Host name = " + value);
          }
        } // end for loop iterating list of exports
      } // end call to QZNFRTVE succeeded
    }
    catch(PcmlException e)
    {
      System.out.println(e.getLocalizedMessage());
      e.printStackTrace();
      System.exit(-1);
    }

    System.exit(0);
  } // end main()
}
```

例: ReportWriter クラス

このセクションでは、ReportWriter クラスの資料に記載されているコーディング例をリストします。

JSPReportProcessor および PDFContext

- 例: PDFContext と共に JSPReportProcessor を使用する
- »例: JSPReportProcessor サンプル JSP ファイル«

XSLReportProcessor および PCLContext

- 例: PCLContext と共に XSLReportProcessor を使用する
- »例: XSLReportProcessor サンプル XML ファイル«
- »例: XSLReportProcessor サンプル XSL ファイル«

以下の特記事項は、IBM Toolbox for Java のすべての例に適用されます。

例: PDFContext と共に JSPReportProcessor を使用する

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
////////////////////////////////////////////////////////////////////////////
//
// The following example (JSPRunReport) uses the JSPReportProcessor and the
// PDFContext classes to obtain data from a specified URL and convert the data
// to the PDF format. The data is then streamed to a file as a PDF document.
//
// To view the contents of an example JSP source file that you can use with
// JSPRunReport, see JSPcust_table.jsp. You can also download a zip file
// that contains the JSP example file. The zip file also contains XML and XSL
// example files that you can use with the XSLReportProcessor example
// (PCLRunReport).
//
// Command syntax:
//     java JSPRunReport <jsp_Url> <output_filename>
//
////////////////////////////////////////////////////////////////////////////

import java.lang.*;
import java.awt.*;
import java.io.*;
import java.net.*;
import java.awt.print.*;
import java.awt.event.*;
import java.util.LinkedList;
import java.util.ListIterator;
import java.util.HashMap;

import com.ibm.xsl.composer.flo.*;
import com.ibm.xsl.composer.areas.*;
import com.ibm.xsl.composer.framework.*;
import com.ibm.xsl.composer.java2d.*;
import com.ibm.xsl.composer.prim.*;
import com.ibm.xsl.composer.properties.*;
import com.ibm.as400.util.reportwriter.processor.*;
import com.ibm.as400.util.reportwriter.pdfwriter.*;
import java.io.IOException;
import java.io.Serializable;
import org.xml.sax.SAXException;


public class JSPRunReport
```

```
{

    public static void main(String args[])
    {
        FileOutputStream fileout = null;

        /** specify the URL that contains the data you want
            to use in your report **/
        String JSPurl = args[0];
        URL jspurl = null;
        try {
              jspurl = new URL(JSPurl);
        }
        catch (MalformedURLException e)
        {}

        /** get output PDF file name  **/
        String filename = args[1];
        try {
        fileout = new FileOutputStream(filename);
        }
        catch (FileNotFoundException e)
        {}

        /** set up page format **/
        Paper paper = new Paper();
        paper.setSize(612,792);
        paper.setImageableArea(18, 18, 576, 756);
        PageFormat pf = new PageFormat();
        pf.setPaper(paper);

        /** create a PDFContext object and cast FileOutputStream
            as an OutputStream **/
        PDFContext pdfcontext = new PDFContext((OutputStream)fileout, pf);

        System.out.println( Ready to parse XSL document );

        /** create the JSPReportProcessor object and set the template
            to the specified JSP **/
        JSPReportProcessor jspprocessor = new JSPReportProcessor(pdfcontext);
        try {
        jspprocessor.setTemplate(jspurl);
        }

        catch (NullPointerException np){
            String mes = np.getMessage();
            System.out.println(mes);
```

```
            System.exit(0);
            }

        /** process the report **/
        try {
        jspprocessor.processReport();
        }
        catch (IOException e) {
            String mes = e.getMessage();
            System.out.println(mes);
            System.exit(0);
            }
        catch (SAXException se) {
            String mes = se.getMessage();
            System.out.println(mes);
            System.exit(0);
            }

        System.exit(0);
    }
}
```

»

例: JSPReportProcessor サンプル JSP ファイル

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
<?xml version="1.0"?>

<!--
  Copyright (c) 1999 The Apache Software Foundation.  All rights
  reserved.
-->

<%@ page session="false"%>
<%@ page language="java" contentType="text/html" %>
<%@ page import="java.lang.*" %>
<%@ page import="java.util.*" %>


<%-- <jsp:useBean id='cust_table' scope='page' class='table.JSPcust_table'/> --%>

<%!
   String[][] cust_data = new String [4][5];

   public void jspInit()
   {
     //cust_record_field [][] cust_data;
          // cust_record holds customer name, customer address, customer city, customer state,
          // customer zip



          String [] cust_record_1 = {"IBM","3602 4th St","Rochester","Mn","55901"};
      String [] cust_record_2 = {"HP","400 2nd","Springfield","Mo","33559"};
      String [] cust_record_3 = {"Wolzack","34 Hwy 52N","Lansing","0r","67895"};
      String [] cust_record_4 = {"Siems","343 60th","Salem","Tx","12345"};

          cust_data[0] = cust_record_1;
          cust_data[1] = cust_record_2;
          cust_data[2] = cust_record_3;
          cust_data[3] = cust_record_4;
   }
%>


<!-- First test of parse and compose. -->
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
    <fo:layout-master-set>
      <fo:simple-page-master master-name="theMaster" >
      <fo:region-body region-name="theRegion" margin-left=".2in"/>
      </fo:simple-page-master>
      <fo:page-sequence-master master-name="theMaster">
      <fo:single-page-master-reference master-name="thePage"/>
      </fo:page-sequence-master>
    </fo:layout-master-set>
```

```
    <fo:page-sequence master-name="theMaster">
     <fo:flow flow-name="theRegion">
       <fo:block>
          <fo:block text-align="center"> NORCAP </fo:block>
          <fo:block space-before=".2in" text-align="center"> PAN PACIFIC HOTEL IN SAN FRANCISCO
</fo:block>
          <fo:block text-align="center"> FRIDAY, DECEMBER 8-9, 2000 </fo:block>
       </fo:block>
       <fo:block space-before=".5in" font-size="8pt">
       <fo:table table-layout="fixed">
         <fo:table-column column-width="3in"/>
         <fo:table-column column-width="3in"/>
         <fo:table-column column-width="3in"/>
         <fo:table-column column-width="3in"/>
         <fo:table-column column-width="3in"/>
         <fo:table-body>
          <fo:table-row>
            <fo:table-cell column-number="1">
               <fo:block border-bottom-style="solid">NAME
               </fo:block>
            </fo:table-cell>
            <fo:table-cell column-number="2">
               <fo:block border-bottom-style="solid">ADDRESS
               </fo:block>
            </fo:table-cell>
            <fo:table-cell column-number="3">
               <fo:block border-bottom-style="solid">CITY
               </fo:block>
            </fo:table-cell>
            <fo:table-cell column-number="4">
               <fo:block border-bottom-style="solid">STATE
               </fo:block>
            </fo:table-cell>
            <fo:table-cell column-number="5">
               <fo:block border-bottom-style="solid">ZIP CODE
               </fo:block>
            </fo:table-cell>
          </fo:table-row>

          <%
            // add row to table
            for(int i = 0; i <= 3; i++)
            {
              String[] _array = cust_data[i];
          %>

          <fo:table-row>
            <fo:table-cell column-number="1">
              <fo:block space-before=".1in">
              <% if(_array[0].equals("IBM")) { %>
                 <fo:inline background-color="blue">
                 <% out.print(_array[0]); %>
                 </fo:inline>
              <% } else { %>
                 <% out.print(_array[0]); %>
              <% } %>
```

```
            </fo:block>
          </fo:table-cell>
          <fo:table-cell column-number="2">
            <fo:block space-before=".1in">
                <% out.print(_array[1]); %>
            </fo:block>
          </fo:table-cell>
          <fo:table-cell column-number="3">
            <fo:block space-before=".1in">
                <% out.print(_array[2]); %>
            </fo:block>
          </fo:table-cell>
          <fo:table-cell column-number="4">
            <fo:block space-before=".1in">
               <% out.print(_array[3]); %>
            </fo:block>
          </fo:table-cell>
          <fo:table-cell column-number="5">
           <fo:block space-before=".1in">
              <% out.print(_array[4]); %>
           </fo:block>
         </fo:table-cell>
        </fo:table-row>

      <%
      } // end row while
      %>

      </fo:table-body>
    </fo:table>
    </fo:block>
  </fo:flow>
 </fo:page-sequence>
</fo:root>
```
«

例: PCLContext と共に XSLReportProcessor を使用する

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
////////////////////////////////////////////////////////////////////////////
//
// The following example (PCLRunReport) uses the XSLPReportProcessor and the
// PCLContext classes to obtain XML data and convert the data to the PCL format.
// The data is then streamed to a printer OutputQueue.
//
// To view the contents of example XML and XSL source files that you can use
// with PCLRunReport, see realestate.xml and realestate.xsl. You can also
// download a zip file that contains the XML and XSL example files. The zip
// file also contains a JSP example file that you can use with the
// JSPReportProcessor example (JSPRunReport).
//
// Command syntax:
//     java PCLRunReport <xml_file> <xsl_file>
//
////////////////////////////////////////////////////////////////////////////

import java.lang.*;
import java.awt.*;
import java.io.*;
import java.awt.print.*;
import java.awt.event.*;
import java.util.LinkedList;
import java.util.ListIterator;
import java.util.HashMap;

import com.ibm.xsl.composer.flo.*;
import com.ibm.xsl.composer.areas.*;
import com.ibm.xsl.composer.framework.*;
import com.ibm.xsl.composer.java2d.*;
import com.ibm.xsl.composer.prim.*;
import com.ibm.xsl.composer.properties.*;
import com.ibm.as400.util.reportwriter.processor.*;
import com.ibm.as400.util.reportwriter.pclwriter.*;
import java.io.IOException;
import java.io.Serializable;
import org.xml.sax.SAXException;
import com.ibm.as400.access.*;

public class PCLRunReport
```

```
{

    public static void main(String args[])
    {
        SpooledFileOutputStream fileout = null;
        String xmldocumentName = args[0];
        String xsldocumentName = args[1];

        String sys = "<system>";      /* Insert ISeries system name       */
        String user = "<user>";       /* Insert ISeries user profile name */
        String pass = "<password>";   /* Insert ISeries password          */

        AS400 system = new AS400(sys, user, pass);

        /* Insert ISeries output queue */
        String outqname = "/QSYS.LIB/qusrsys.LIB/<outq>.OUTQ";
        OutputQueue outq = new OutputQueue(system, outqname);
        PrintParameterList parms = new PrintParameterList();
        parms.setParameter(PrintObject.ATTR_OUTPUT_QUEUE, outq.getPath());

        try{
          fileout = new SpooledFileOutputStream(system, parms, null, null);
        }
        catch (Exception e)
        {}

        /** set up page format **/
        Paper paper = new Paper();
        paper.setSize(612,792);
        paper.setImageableArea(18, 36, 576, 720);
        PageFormat pf = new PageFormat();
        pf.setPaper(paper);

        /** create a PCLContext object and case FileOutputStream
            as an OutputStream **/
        PCLContext pclcontext = new PCLContext((OutputStream)fileout, pf);

        System.out.println("Ready to parse XSL document");

        /** create the XSLReportProcessor object **/
        XSLReportProcessor xslprocessor = new XSLReportProcessor(pclcontext);
        try {
        xslprocessor.setXMLDataSource(xmldocumentName);
        }
        catch (SAXException se) {
            String mes = se.getMessage();
            System.out.println(mes);
            System.exit(0);
```

```
        }
    catch (IOException ioe) {
        String mes = ioe.getMessage();
        System.out.println(mes);
        System.exit(0);
        }
    catch (NullPointerException np){
        String mes = np.getMessage();
        System.out.println(mes);
        System.exit(0);
        }
    /** set the template to the specified XML data source **/
    try {
    xslprocessor.setTemplate(xsldocumentName);
    }
    catch (NullPointerException np){
        String mes = np.getMessage();
        System.out.println(mes);
        System.exit(0);
        }
    catch (IOException e) {
        String mes = e.getMessage();
        System.out.println(mes);
        System.exit(0);
        }
    catch (SAXException se) {
        String mes = se.getMessage();
        System.out.println(mes);
        System.exit(0);
        }

    /** process the report **/
    try {
    xslprocessor.processReport();
    }
    catch (IOException e) {
        String mes = e.getMessage();
        System.out.println(mes);
        System.exit(0);
        }
    catch (SAXException se) {
        String mes = se.getMessage();
        System.out.println(mes);
        System.exit(0);
        }

    System.exit(0);
}
```

```
}
```

»

例: XSLReportProcessor サンプル XML ファイル

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
<?xml version="1.0"?>

<RESIDENTIAL-LISTINGS VERSION="061698">

<RESIDENTIAL-LISTING ID="ID1287" VERSION="061698">
  <GENERAL>
        <TYPE>Apartment</TYPE>
        <PRICE>$110,000</PRICE>

        <STRUCTURE><NUM-BEDS>3</NUM-BEDS><NUM-BATHS>1</NUM-BATHS></STRUCTURE>
  <AGE UNITS="YEARS">15</AGE>
  <LOCATION COUNTRY="USA" STATE="MA" COUNTY="MIDDLESEX" SECURITY="Public">
          <ADDRESS>13 Some Avenue</ADDRESS>
    <CITY>Dorchester</CITY><ZIP>02121</ZIP>
        </LOCATION>
  <IMAGE FORMAT="JPG" WIDTH="300" HEIGHT="150" SRC="house1.jpg"/>
        <MLS>
    <MLS-CODE SECURITY="Restricted">
                        30224877
                </MLS-CODE>
    <MLS-SOURCE SECURITY="Public">
                        <NAME>Bob the Realtor</NAME>
                        <PHONE>1-617-555-1212</PHONE>
                        <FAX>1-617-555-1313</FAX>
                        <WEB>
                                <EMAIL>Bob@bigbucks.com</EMAIL>
                                <SITE>www.bigbucks.com</SITE>
                        </WEB>
                </MLS-SOURCE>
        </MLS>
        <DATES><LISTING-DATE>3/5/98</LISTING-DATE></DATES>
        <LAND-AREA UNITS="ACRES">0.01</LAND-AREA>

  </GENERAL>

  <FEATURES>
     <DISCLOSURES>
                In your dreams.
     </DISCLOSURES>
         <UTILITIES>
```

```
            Yes
        </UTILITIES>
        <EXTRAS>
            Pest control included.
        </EXTRAS>
        <CONSTRUCTION>
            Wallboard and glue
        </CONSTRUCTION>
        <ACCESS>
            Front door.
        </ACCESS>
</FEATURES>

<FINANCIAL>
        <ASSUMABLE>
            I assume so.
        </ASSUMABLE>
        <OWNER-CARRY>
            Too heavy.
        </OWNER-CARRY>
        <ASSESMENTS>
            $150,000
        </ASSESMENTS>
        <DUES>
            $100
        </DUES>
        <TAXES>
            $2,000
        </TAXES>
        <LENDER>
            Fly by nite mortgage co.
        </LENDER>
        <EARNEST>
            Burt
        </EARNEST>
        <DIRECTIONS>
            North, south, east, west
        </DIRECTIONS>
</FINANCIAL>

<REMARKS>
</REMARKS>

<CONTACTS>
      <COMPANY>
              <NAME>
                 Noplace Realty
```

```
                </NAME>
                <ADDRESS>
                   12 Main Street
                </ADDRESS>
                <CITY>
                   Lowell, MA
                </CITY>
                <ZIP>
                   34567
                </ZIP>
        </COMPANY>
        <AGENT>
                <NAME>
                   Mary Jones
                </NAME>
                <ADDRESS>
                </ADDRESS>
                <CITY>
                </CITY>
                <ZIP>
                </ZIP>
        </AGENT>
        <OWNER>
                <NAME>
                </NAME>
                <ADDRESS>
                </ADDRESS>
                <CITY>
                </CITY>
                <ZIP>
                </ZIP>
        </OWNER>
    <TENANT>
           Yes.
        </TENANT>
    <COMMISION>
           15%
        </COMMISION>
  </CONTACTS>

</RESIDENTIAL-LISTING>

<RESIDENTIAL-LISTING VERSION="061698" ID="ID1289">
  <GENERAL>

    <IMAGE FORMAT="JPG" WIDTH="300" HEIGHT="150" SRC="house2.jpg">
        </IMAGE>
```

```
<MLS>
        <MLS-CODE SECURITY="Restricted">
                                        30298877
        </MLS-CODE>
        <MLS-SOURCE SECURITY="Public">
                <NAME>
                        Mary the Realtor
                </NAME>
                <PHONE>
                        1-617-555-3333
                </PHONE>
                <FAX>
                        1-617-555-4444
                </FAX>
                <WEB>
                        <EMAIL>
                          Mary@somebucks.com
                        </EMAIL>
                        <SITE>
                          www.bigbucks.com
                        </SITE>
                </WEB>
        </MLS-SOURCE>
</MLS>

<TYPE>
        Home
</TYPE>

<PRICE>
        $200,000
</PRICE>

<AGE UNITS="MONTHS">
        3
</AGE>

<LOCATION COUNTRY="USA" STATE="CO" COUNTY="MIDDLESEX" SECURITY="Public">
        <ADDRESS>
                1 Main Street
        </ADDRESS>
        <CITY>
            Boulder
        </CITY>
        <ZIP>
                11111
```

```
            </ZIP>
        </LOCATION>

        <STRUCTURE>
          <NUM-BEDS>
            2
          </NUM-BEDS>
          <NUM-BATHS>
            2
          </NUM-BATHS>
        </STRUCTURE>

        <DATES>
            <LISTING-DATE>
                4/3/98
            </LISTING-DATE>
        </DATES>

        <LAND-AREA UNITS="ACRES">
            0.01
        </LAND-AREA>

</GENERAL>

<FEATURES>
   <DISCLOSURES>
            In your dreams.
   </DISCLOSURES>
       <UTILITIES>
          Yes
       </UTILITIES>
       <EXTRAS>
          Pest control included.
       </EXTRAS>
       <CONSTRUCTION>
          Wallboard and glue
       </CONSTRUCTION>
       <ACCESS>
          Front door.
       </ACCESS>
</FEATURES>

<FINANCIAL>
       <ASSUMABLE>
          I assume so.
       </ASSUMABLE>
       <OWNER-CARRY>
```

```
            Too heavy.
         </OWNER-CARRY>
         <ASSESMENTS>
            $150,000
         </ASSESMENTS>
         <DUES>
            $100
         </DUES>
         <TAXES>
            $2,000
         </TAXES>
         <LENDER>
            Fly by nite mortgage co.
         </LENDER>
         <EARNEST>
            Burt
         </EARNEST>
         <DIRECTIONS>
            North, south, east, west
         </DIRECTIONS>
</FINANCIAL>

<REMARKS>
</REMARKS>

<CONTACTS>
      <COMPANY>
               <NAME>
                  Noplace Realty
               </NAME>
               <ADDRESS>
                  12 Main Street
               </ADDRESS>
               <CITY>
                  Lowell, MA
               </CITY>
               <ZIP>
                  34567
               </ZIP>
      </COMPANY>
      <AGENT>
               <NAME>
                  Mary Jones
               </NAME>
               <ADDRESS>
               </ADDRESS>
               <CITY>
```

```
                </CITY>
                <ZIP>
                </ZIP>
        </AGENT>
        <OWNER>
                <NAME>
                </NAME>
                <ADDRESS>
                </ADDRESS>
                <CITY>
                </CITY>
                <ZIP>
                </ZIP>
        </OWNER>
    <TENANT>
           Yes.
        </TENANT>
    <COMMISION>
           15%
        </COMMISION>
  </CONTACTS>

</RESIDENTIAL-LISTING>
<RESIDENTIAL-LISTING VERSION="061698" ID="ID1290">
  <GENERAL>

    <IMAGE FORMAT="JPG" WIDTH="300" HEIGHT="150" SRC="house3.jpg">
        </IMAGE>

        <MLS>
                <MLS-CODE SECURITY="Restricted">
                        20079877
                </MLS-CODE>
                <MLS-SOURCE SECURITY="Public">
                        <NAME>
                                Bob the Realtor
                        </NAME>
                        <PHONE>
                                1-617-555-1212
                        </PHONE>
                        <FAX>
                                1-617-555-1313
                        </FAX>
                        <WEB>
                                <EMAIL>
                                  Bob@bigbucks.com
                                </EMAIL>
```

```
                        <SITE>
                          www.bigbucks.com
                        </SITE>
                </WEB>
        </MLS-SOURCE>
</MLS>

<TYPE>
        Apartment
</TYPE>

<PRICE>
        $65,000
</PRICE>

<AGE UNITS="YEARS">
        30
</AGE>

<LOCATION COUNTRY="USA" STATE="MA" COUNTY="MIDDLESEX" SECURITY="Public">
        <ADDRESS>
                25 Which Ave.
        </ADDRESS>
        <CITY>
            Cambridge
        </CITY>
        <ZIP>
                02139
        </ZIP>
</LOCATION>

<STRUCTURE>
  <NUM-BEDS>
    3
  </NUM-BEDS>
  <NUM-BATHS>
    1
  </NUM-BATHS>
</STRUCTURE>

<DATES>
        <LISTING-DATE>
            3/5/97
        </LISTING-DATE>
</DATES>

<LAND-AREA UNITS="ACRES">
```

```
                0.05
        </LAND-AREA>

</GENERAL>

<FEATURES>
   <DISCLOSURES>
                In your dreams.
   </DISCLOSURES>
        <UTILITIES>
           Yes
        </UTILITIES>
        <EXTRAS>
           Pest control included.
        </EXTRAS>
        <CONSTRUCTION>
           Wallboard and glue
        </CONSTRUCTION>
        <ACCESS>
           Front door.
        </ACCESS>
</FEATURES>

<FINANCIAL>
        <ASSUMABLE>
           I assume so.
        </ASSUMABLE>
        <OWNER-CARRY>
           Too heavy.
        </OWNER-CARRY>
        <ASSESMENTS>
           $150,000
        </ASSESMENTS>
        <DUES>
           $100
        </DUES>
        <TAXES>
           $2,000
        </TAXES>
        <LENDER>
           Fly by nite mortgage co.
        </LENDER>
        <EARNEST>
           Burt
        </EARNEST>
        <DIRECTIONS>
           North, south, east, west
```

```
        </DIRECTIONS>
</FINANCIAL>

<REMARKS>
</REMARKS>

<CONTACTS>
      <COMPANY>
              <NAME>
                 Noplace Realty
              </NAME>
              <ADDRESS>
                 12 Main Street
              </ADDRESS>
              <CITY>
                 Lowell, MA
              </CITY>
              <ZIP>
                 34567
              </ZIP>
      </COMPANY>
      <AGENT>
              <NAME>
                 Mary Jones
              </NAME>
              <ADDRESS>
              </ADDRESS>
              <CITY>
              </CITY>
              <ZIP>
              </ZIP>
      </AGENT>
      <OWNER>
              <NAME>
              </NAME>
              <ADDRESS>
              </ADDRESS>
              <CITY>
              </CITY>
              <ZIP>
              </ZIP>
      </OWNER>
  <TENANT>
         Yes.
      </TENANT>
  <COMMISION>
         15%
```

```
        </COMMISION>
  </CONTACTS>

</RESIDENTIAL-LISTING>
<RESIDENTIAL-LISTING VERSION="061698"  ID="ID1291">
  <GENERAL>

    <IMAGE FORMAT="JPG" WIDTH="300" HEIGHT="150" SRC="house4.jpg">
        </IMAGE>

        <MLS>
                <MLS-CODE SECURITY="Restricted">
                        29389877
                </MLS-CODE>
                <MLS-SOURCE SECURITY="Public">
                        <NAME>
                                Mary the Realtor
                        </NAME>
                        <PHONE>
                                1-617-555-3333
                        </PHONE>
                        <FAX>
                                1-617-555-4444
                        </FAX>
                        <WEB>
                                <EMAIL>
                                  Mary@somebucks.com
                                </EMAIL>
                                <SITE>
                                  www.bigbucks.com
                                </SITE>
                        </WEB>
                </MLS-SOURCE>
        </MLS>

        <TYPE>
        Home
        </TYPE>

        <PRICE>
                $449,000
        </PRICE>

        <AGE UNITS="YEARS">
                7
        </AGE>
```

```
        <LOCATION COUNTRY="USA" STATE="MA" COUNTY="MIDDLESEX" SECURITY="Public">
                <ADDRESS>
                        100 Any Road
                </ADDRESS>
                <CITY>
                    Lexington
                </CITY>
                <ZIP>
                        02421
                </ZIP>
        </LOCATION>

        <STRUCTURE>
          <NUM-BEDS>
            7
          </NUM-BEDS>
          <NUM-BATHS>
            3
          </NUM-BATHS>
        </STRUCTURE>

        <DATES>
                <LISTING-DATE>
                    6/8/98
                </LISTING-DATE>
        </DATES>

        <LAND-AREA UNITS="ACRES">
                2.0
        </LAND-AREA>

</GENERAL>

<FEATURES>
   <DISCLOSURES>
            In your dreams.
   </DISCLOSURES>
       <UTILITIES>
          Yes
       </UTILITIES>
       <EXTRAS>
          Pest control included.
       </EXTRAS>
       <CONSTRUCTION>
          Wallboard and glue
       </CONSTRUCTION>
       <ACCESS>
```

```
            Front door.
        </ACCESS>
</FEATURES>

<FINANCIAL>
        <ASSUMABLE>
            I assume so.
        </ASSUMABLE>
        <OWNER-CARRY>
            Too heavy.
        </OWNER-CARRY>
        <ASSESMENTS>
            $300,000
        </ASSESMENTS>
        <DUES>
            $100
        </DUES>
        <TAXES>
            $2,000
        </TAXES>
        <LENDER>
            Fly by nite mortgage co.
        </LENDER>
        <EARNEST>
            Burt
        </EARNEST>
        <DIRECTIONS>
            North, south, east, west
        </DIRECTIONS>
</FINANCIAL>

<REMARKS>
</REMARKS>

<CONTACTS>
      <COMPANY>
              <NAME>
                 Noplace Realty
              </NAME>
              <ADDRESS>
                 12 Main Street
              </ADDRESS>
              <CITY>
                 Lowell, MA
              </CITY>
              <ZIP>
                 34567
```

```
                </ZIP>
        </COMPANY>
        <AGENT>
                <NAME>
                   Mary Jones
                </NAME>
                <ADDRESS>
                </ADDRESS>
                <CITY>
                </CITY>
                <ZIP>
                </ZIP>
        </AGENT>
        <OWNER>
                <NAME>
                </NAME>
                <ADDRESS>
                </ADDRESS>
                <CITY>
                </CITY>
                <ZIP>
                </ZIP>
        </OWNER>
    <TENANT>
           Yes.
        </TENANT>
    <COMMISION>
           15%
        </COMMISION>
  </CONTACTS>

</RESIDENTIAL-LISTING>

</RESIDENTIAL-LISTINGS>
```

»

例: XSLReportProcessor サンプル XSL ファイル

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
<?xml version="1.0"?>

<!-- Sample of styling an imagined real estate document. -->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
       xmlns:fo="http://www.w3.org/1999/XSL/Format" >

  <xsl:template match="RESIDENTIAL-LISTINGS">
    <fo:root>
    <fo:layout-master-set>
        <fo:simple-page-master master-name="theMaster">
         <fo:region-body region-name="theRegion"/>
        </fo:simple-page-master>
        <fo:page-sequence-master master-name="theMaster">
           <fo:single-page-master-reference master-name="thePage" />
      </fo:page-sequence-master>
        </fo:layout-master-set>
         <fo:page-sequence master-name="theMaster">
         <fo:flow flow-name="theRegion">
           <xsl:apply-templates/>
        </fo:flow>
        </fo:page-sequence>
    </fo:root>
  </xsl:template>
  <xsl:template match="RESIDENTIAL-LISTING">


      <fo:block font-family="Times New Roman" font-weight="normal" font-size="24pt"
       background-color="silver" padding-before="5px" padding-after="5px"
       padding-start="5px" padding-end="5px" border-before-style="solid"
       border-before-color="blue" border-after-style="solid" border-after-color="blue"
       border-start-style="solid" border-start-color="blue" border-end-style="solid"
       border-end-color="blue">

        <fo:character character="y" background-color="blue" border-before-style="solid"
       border-before-color="yellow" border-after-style="solid" border-after-color="yellow"
       border-start-style="solid" border-start-color="yellow" border-end-style="solid"
       border-end-color="yellow" />
        </fo:block>


  </xsl:template>
</xsl:stylesheet>
```

«

例: リソース・クラス

このセクションでは、リソース・クラスの資料に記載されているコーディング例をリストします。

Resource および ChangeableResource

- RUser からの属性値の取り出し。RUser は Resource の具象サブクラスです。
- 例: RJob の属性値の設定。RJob は ChangeableResource の具象サブクラスです。
- 例: 汎用コードを使用してリソースにアクセスする

ResourceList

- 例: ResourceList の内容を取得および印刷する
- 例: 汎用コードを使用して ResourceList にアクセスする
- 例: サーブレットでリソース・リストを表示する

プレゼンテーション

- 例: プレゼンテーションを使用する

以下の特記事項は、IBM Toolbox for Java のすべての例に適用されます。

例: Resource からの属性値の取り出し

Resource の 1 つの具象サブクラスは com.ibm.as400.resource.RUser で、 これは iSeries ユーザーを表します。 RUser では、多数の属性 ID がサポートされ、 それぞれの属性 ID を使用して属性値を取得することができます。

この例では、RUser から属性値を取り出します。

```
// Create an RUser object to refer to a specific user.
AS400 system = new AS400("MYSYSTEM", "MYUSERID", "MYPASSWORD");
RUser user = new RUser(system, "AUSERID");

// Get the text description attribute value.
String textDescription = (String)user.getAttributeValue(RUser.TEXT_DESCRIPTION);
```

例: ChangeableResource の属性値の設定

ChangeableResource の 1 つの具象サブクラスは com.ibm.as400.resource.RJob で、これは iSeries ジョブを表します。 RJob では、多数の属性 ID がサポートされ、それぞれの属性 ID を使用して属性値にアクセスすることができます。 以下の例では、RJob の 2 つの属性値を設定します。

```
// Create an RJob object to refer to a specific job.
AS400 system = new AS400("MYSYSTEM", "MYUSERID", "MYPASSWORD");
RJob job = new RJob(system, "AJOBNAME", "AUSERID", "AJOBNUMBER");

// Set the date format attribute value.
job.setAttributeValue(RJob.DATE_FORMAT, RJob.DATE_FORMAT_JULIAN);

// Set the country or region ID attribute value.
job.setAttributeValue(RJob.COUNTRY_ID, RJob.USER_PROFILE);

// Commit both attribute changes.
job.commitAttributeChanges();
```

例: 汎用コードを使用してリソースにアクセスする

Resource または ChangeableResource サブクラスを扱うための汎用コードを作成することができます。 そのようなコードを使用すると、再使用可能性および保守容易性が向上し、 さらに将来、変更なしで Resource または ChangeableResource サブクラスを扱うことができます。

すべての属性は、関連する属性メタデータ・オブジェクト (com.ibm.as400.resource.ResourceMetaData) を持ちます。 このオブジェクトは、属性のさまざまなプロパティーを記述するものです。 これらのプロパティーには、属性が読み取り専用であるかどうかや、 デフォルト値および使用可能な値が含まれます。

以下は、リソースによってサポートされるすべての属性の値を印刷する汎用コードの例です。

```
void printAllAttributeValues(Resource resource) throws ResourceException
{
    // Get the attribute meta data.
    ResourceMetaData[] attributeMetaData = resource.getAttributeMetaData();


    // Loop through all attributes and print the values.
    for(int i = 0; i < attributeMetaData.length; ++i)
    {
        Object attributeID = attributeMetaData[i].getID();
        Object value = resource.getAttributeValue(attributeID);
        System.out.println("Attribute " + attributeID + " = " + value);
    }
}
```

以下は、ChangeableResource のすべての属性をデフォルト値にリセットする汎用コードの例です。

```
void resetAttributeValues(ChangeableResource resource) throws ResourceException
{
    // Get the attribute meta data.
    ResourceMetaData[] attributeMetaData = resource.getAttributeMetaData();


    // Loop through all attributes.
    for(int i = 0; i < attributeMetaData.length; ++i)
    {
        // If the attribute is changeable (not read only), then
        // reset its value to the default.
        if (! attributeMetaData[i].isReadOnly())
        {
```

```
            Object attributeID = attributeMetaData[i].getID();
            Object defaultValue = attributeMetaData[i].getDefaultValue();
            resource.setAttributeValue(attributeID, defaultValue);
        }
    }

    // Commit all of the attribute changes.
    resource.commitAttributeChanges();
}
```

例: リソース・リスト

以下の例では、リソース・リストを扱うさまざまな方法を示します。

- 例: ResourceList の内容を取得および印刷する
- 例: 汎用コードを使用して ResourceList にアクセスする
- 例: サーブレットでリソース・リストを表示する

例: ResourceList の内容を取得および印刷する

ResourceList の具象サブクラスの一例として、 iSeries ジョブのリストを表す com.ibm.as400.resource.RJobList があります。 RJobList では、多数の選択 ID とソート ID がサポートされます。 これらはそれぞれ、リストをフィルターに掛けるかまたはソートするために使用できます。 この例では、RJobList の内容を印刷します。

```
// Create an RJobList object to represent a list of jobs.
AS400 system = new AS400("MYSYSTEM", "MYUSERID", "MYPASSWORD");
RJobList jobList = new RJobList(system);

// Filter the list to include only interactive jobs.
jobList.setSelectionValue(RJobList.JOB_TYPE, RJob.JOB_TYPE_INTERACTIVE);

// Sort the list by user name, then job name.
Object[] sortValue = new Object[] { RJob.USER_NAME, RJob.JOB_NAME };
jobList.setSortValue(sortValue);

// Open the list and wait for it to complete.
jobList.open();
jobList.waitForComplete();

// Read and print the contents of the list.
long length = jobList.getListLength();
for(long i = 0; i < length; ++i)
{
    System.out.println(jobList.resourceAt(i));
}

// Close the list.
jobList.close();
```

例: 汎用コードを使用してリソースを処理する

具象サブクラスを直接に使用する以外に、 ResourceList サブクラスを扱うための汎用コードを作成することができます。 そのようなコードを使用すると、再使用可能性および保守容易性が向上し、 さらに将来、変更なしで ResourceList サブクラスを扱うことができます。

例: ResourceList の内容を印刷する

以下は、ResourceList の内容の一部を印刷する汎用コードの例です。

```
void printContents(ResourceList resourceList, long numberOfItems) throws ResourceException
{
    // Open the list and wait for the requested number of items
    // to become available.
```

```
        resourceList.open();
        resourceList.waitForResource(numberOfItems);

        for(long i = 0; i < numberOfItems; ++i)
        {
            System.out.println(resourceList.resourceAt(i));
        }
    }
```

例: ResourceMetaData を使用して、リソースによってサポートされるすべての属性にアクセスする

すべての属性は、関連する属性メタデータ・オブジェクト (com.ibm.as400.resource.ResourceMetaData) を持ちます。 このオブジェクトは、属性のさまざまなプロパティーを記述するものです。 これらのプロパティーには、属性が読み取り専用であるかどうかや、 デフォルト値および使用可能な値が含まれます。

以下は、リソースによってサポートされるすべての属性の値を印刷する汎用コードの例です。

```
void printAllAttributeValues(Resource resource) throws ResourceException
{
    // Get the attribute meta data.
    ResourceMetaData[] attributeMetaData = resource.getAttributeMetaData();


    // Loop through all attributes and print the values.
    for(int i = 0; i < attributeMetaData.length; ++i)
    {
        Object attributeID = attributeMetaData[i].getID();
        Object value = resource.getAttributeValue(attributeID);
        System.out.println("Attribute " + attributeID + " = " + value);
    }
}
```

例: ResourceMetaData を使用して ChangeableResource のすべての属性をリセットする

以下は、ChangeableResource のすべての属性をデフォルト値にリセットする汎用コードの例です。

```
void resetAttributeValues(ChangeableResource resource) throws ResourceException
{
    // Get the attribute meta data.
    ResourceMetaData[] attributeMetaData = resource.getAttributeMetaData();


    // Loop through all attributes.
    for(int i = 0; i < attributeMetaData.length; ++i)
    {
        // If the attribute is changeable (not read only), then
        // reset its value to the default.
        if (! attributeMetaData[i].isReadOnly())
        {
            Object attributeID = attributeMetaData[i].getID();
            Object defaultValue = attributeMetaData[i].getDefaultValue();
            resource.setAttributeValue(attributeID, defaultValue);
        }
    }

    // Commit all of the attribute changes.
```

```
    resource.commitAttributeChanges();
}
```

例: サーブレットでリソース・リストを表示する

サーブレットでリソース・リストを表示するには、 HTMLFormConverter または HTMLTableConverter クラスと一緒に ResourceListRowData クラスを使用します。

- HTMLFormConverter は、リソース・リストの内容を一連のフォームとして表示します。 それぞれのフォームには、リソース・リスト内の 1 つのリソースの属性値が入ります。
- HTMLTableConverter は、リソース・リストの内容をテーブルとして表示します。 それぞれの行には、リソース・リスト内の 1 つのリソースに関する情報が入ります。

ResourceListRowData オブジェクトの列は列属性 ID の配列として指定され、 それぞれの行は 1 つのリソース・オブジェクトを表します。

```
// Create the resource list.  This example creates
// a list of all messages in the current user's message
// queue.
AS400 system = new AS400("MYSYSTEM", "MYUSERID", "MYPASSWORD");
RMessageQueue messageQueue = new RMessageQueue(system, RMessageQueue.CURRENT);

// Create the ResourceListRowData object.  In this example,
// there are four columns in the table.  The first column
// contains the icons and names for each message in the
// message queue.  The remaining columns contain the text,
// severity, and type for each message.
ResourceListRowData rowdata = new ResourceListRowData(messageQueue,
    new Object[] { null, RQueuedMessage.MESSAGE_TEXT, RQueuedMessage.MESSAGE_SEVERITY,
                   RQueuedMessage.MESSAGE_TYPE } );

// Create HTMLTable and HTMLTableConverter objects to
// use for generating and customizing the HTML tables.
HTMLTable table = new HTMLTable();
table.setCellSpacing(6);
table.setBorderWidth(8);

HTMLTableConverter converter = new HTMLTableConverter();
converter.setTable(table);
converter.setUseMetaData(true);

// Generate the HTML table.
String[] html = converter.convert(rowdata);
System.out.println(html[0]);
```

例: RFML

このセクションでは、RFML の資料に記載されているコーディング例をリストします。

- 例: Toolbox for Java の Record クラスの使用と比較した RFML の使用
- 例: RFML ソース・ファイル

以下の特記事項は、IBM Toolbox for Java のすべての例に適用されます。

例: プロファイル・トークン信任証を使用して OS/400 スレッド ID を交換する

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

以下のコード例では、プロファイル・トークンを使用して、OS/400 スレッド ID を交換し、 特定のユーザーに代わって作業を実行する方法を示します。

```
// Prepare to work with the local AS/400 system.
AS400 system = new AS400("localhost", "*CURRENT", "*CURRENT");

// Create a single-use ProfileTokenCredential with a 60 second timeout.
// A valid user ID and password must be substituted.
ProfileTokenCredential pt = new ProfileTokenCredential();
pt.setSystem(system);
pt.setTimeoutInterval(60);
pt.setTokenType(ProfileTokenCredential.TYPE_SINGLE_USE);

pt.setToken("USERID", "PASSWORD");

// Swap the OS/400 thread identity, retrieving a credential to
// swap back to the original identity later.
AS400Credential cr = pt.swap(true);

// Perform work under the swapped identity at this point.

// Swap back to the original OS/400 thread identity.
cr.swap();

// Clean up the credentials.
cr.destroy();
pt.destroy();
```

サーブレット・クラスの例

以下の例では、サーブレット・クラスのいくつかの使用法を示します。

- 例: ListRowData を使用する
- 例: RecordListRowData を使用する
- 例: SQLResultSetRowData を使用する
- 例: HTMLFormConverter を使用する
- 例: ListMetaData を使用する
- 例: SQLResultSetMetaData を使用する
- 例: サーブレットでリソース・リストを表示する

この例にあるように、 サーブレット・クラスと HTML クラスを一緒に使用することもできます。

以下の特記事項は、IBM Toolbox for Java のすべての例に適用されます。

例: ListRowData を使用する

この例は 3 つの部分に分かれています。

- ListRowData クラスの動作を示す Java ソース
- HTMLTableConverter を使用して Java ソースから生成される HTML ソース
- 生成された HTML をブラウザーが表示する方法

ListRowData クラスの動作を示す Java ソース

```
// Access an existing non-empty data queue
KeyedDataQueue dq = new KeyedDataQueue(systemObject_, "/QSYS.LIB/MYLIB.LIB/MYDQ.DTAQ");

// Create a metadata object.
ListMetaData metaData = new ListMetaData(2);

// Set first column to be the customer ID.
metaData.setColumnName(0, "Customer ID");
metaData.setColumnLabel(0, "Customer ID");
metaData.setColumnType(0, RowMetaDataType.STRING_DATA_TYPE);

// Set second column to be the order to be processed.
metaData.setColumnName(1, "Order Number");
metaData.setColumnLabel(1, "Order Number");
metaData.setColumnType(1, RowMetaDataType.STRING_DATA_TYPE);

// Create a ListRowData object.
ListRowData rowData = new ListRowData();
rowData.setMetaData(metaData);

// Get the entries off the data queue.
KeyedDataQueueEntry data = dq.read(key, 0, "EQ");
while (data != null)
{
   // Add queue entry to row data object.
   Object[] row = new Object[2];
   row[0] = new String(key);
   row[1] = new String(data.getData());
   rowData.addRow(row);

   // Get another entry from the queue.
   data = dq.read(key, 0, "EQ");
}

// Create an HTML converter object and convert the rowData to HTML.
HTMLTableConverter conv = new HTMLTableConverter();
conv.setUseMetaData(true);
HTMLTable[] html = conv.convertToTables(rowData);

// Display the output from the converter.
System.out.println(html[0]);
```

HTMLTableConverter を使用して Java ソースから生成される HTML ソース

上記の Java ソースの例における HTMLTableConverter クラスの使用により、 以下の HTML コードが生成されます。

```
<table>
<tr>
<th>Customer ID</th>
<th>Order Number</th>
</tr>
<tr>
<td>777-53-4444</td>
<td>12345-XYZ</td>
</tr>
<tr>
<td>777-53-4444</td>
<td>56789-ABC</td>
</tr>
</table>
```

生成された HTML をブラウザーが表示する方法

以下の表は、この HTML ソース・コードがどのようにブラウザーで表示されるかを示します。

Customer ID	Order Number
777-53-4444	12345-XYZ
777-53-4444	56789-ABC

例: RecordListRowData を使用する

この例は 3 つの部分に分かれています。

- RecordListRowData クラスの動作を示す Java ソース
- HTMLTableConverter を使用して Java ソースから生成された HTML ソース
- 生成された HTML をブラウザーが表示する方法

RecordListRowData クラスの動作を示す Java ソース

```
        // Create a server object.
        AS400 mySystem = new AS400 ("mySystem.myComp.com", "UserId", "Password");

        // Get the path name for the file.
        QSYSObjectPathName file = new QSYSObjectPathName(myLibrary, myFile, "%first%", "mbr");
        String ifspath = file.getPath();

        // Create a file object that represents the file.
        SequentialFile sf = new SequentialFile(mySystem, ifspath);

        // Retrieve the record format from the file.
        AS400FileRecordDescription recordDescription = new AS400FileRecordDescription(mySystem,
ifspath);
        RecordFormat recordFormat = recordDescription.retrieveRecordFormat()[0];

        // Set the record format for the file.
        sf.setRecordFormat(recordFormat);

        // Get the records in the file.
        Record[] records = sf.readAll();

        // Create a RecordListRowData object and add the records.
        RecordListRowData rowData = new RecordListRowData(recordFormat);

        for (int i=0; i < records.length; i++)
        {
           rowData.addRow(records[i]);
        }

        // Create an HTML converter object and convert the rowData to HTML.
        HTMLTableConverter conv = new HTMLTableConverter();
        conv.setMaximumTableSize(3);
        HTMLTable[] html = conv.convertToTables(rowData);

        // Display the first HTML table generated by the converter.
        System.out.println(html[0]);
```

HTMLTableConverter を使用して Java ソースから生成される HTML ソース

上記の Java ソースの例における HTMLTableConverter クラスの使用により、 以下の HTML コードが生成されます。

```
<table>
<tr>
<th>CUSNUM</th>
<th>LSTNAM</th>
<th>INIT</th>
```

```
<th>STREET</th>
<th>CITY</th>
<th>STATE</th>
<th>ZIPCOD</th>
<th>CDTLMT</th>
<th>CHGCOD</th>
<th>BALDUE</th>
<th>CDTDUE</th>
</tr>
<tr>
<td>938472</td>
<td>Henning </td>
<td>G K</td>
<td>4859 Elm Ave </td>
<td>Dallas</td>
<td>TX</td>
<td align="right">75217</td>
<td align="right">5000</td>
<td align="right">3</td>
<td align="right">37.00</td>
<td align="right">0.00</td>
</tr>
<tr>
<td>839283</td>
<td>Jones   </td>
<td>B D</td>
<td>21B NW 135 St</td>
<td>Clay  </td>
<td>NY</td>
<td align="right">13041</td>
<td align="right">400</td>
<td align="right">1</td>
<td align="right">100.00</td>
<td align="right">0.00</td>
</tr>
<tr>
<td>392859</td>
<td>Vine    </td>
<td>S S</td>
<td>PO Box 79    </td>
<td>Broton</td>
<td>VT</td>
<td align="right">5046</td>
<td align="right">700</td>
<td align="right">1</td>
<td align="right">439.00</td>
<td align="right">0.00</td>
</tr>
</table>
```

生成された HTML をブラウザーが表示する方法

以下の表は、この HTML ソース・コードがどのようにブラウザーで表示されるかを示します。

CUSNUM	LSTNAM	INIT	STREET	CITY	STATE	ZIPCOD	CDTLMT	CHGCOD	BALDUE	CDTDUE
938472	Henning	G K	4859 Elm Ave	Dallas	TX	75217	5000	3	37.00	0.00
839283	Jones	B D	21B NW 135 St	Clay	NY	13041	400	1	100.00	0.00
392859	Vine	S S	PO Box 79	Broton	VT	5046	700	1	439.00	0.00

例: SQLResultSetRowData を使用する

この例は 3 つの部分に分かれています。

- SQLResultSetRowData クラスの動作を示す Java ソース
- HTMLTableConverter を使用して Java ソースから生成された HTML ソース
- 生成された HTML をブラウザーが表示する方法

SQLResultSetRowData クラスの動作を示す Java ソース

```
        // Create a server object.
        AS400 mySystem = new AS400 ("mySystem.myComp.com", "UserId", "Password");

        // Register and get a connection to the database.
        DriverManager.registerDriver(new com.ibm.as400.access.AS400JDBCDriver());
        Connection connection = DriverManager.getConnection("jdbc:as400://" +
mySystem.getSystemName());

        // Execute an SQL statement and get the result set.
        Statement statement = connection.createStatement();
        statement.execute("select * from qiws.qcustcdt");
        ResultSet resultSet = statement.getResultSet();

        // Create the SQLResultSetRowData object and initialize to the result set.
        SQLResultSetRowData rowData = new SQLResultSetRowData(resultSet);

        // Create an HTML table object to be used by the converter.
        HTMLTable table = new HTMLTable();

        // Set descriptive column headers.
        String[] headers = {"Customer Number", "Last Name", "Initials",
                                        "Street Address", "City", "State", "Zip Code",
                                        "Credit Limit", "Charge Code", "Balance Due",
                                        "Credit Due"};
        table.setHeader(headers);

        // Set several formatting options within the table.
        table.setBorderWidth(2);
        table.setCellSpacing(1);
        table.setCellPadding(1);

        // Create an HTML converter object and convert the rowData to HTML.
        HTMLTableConverter conv = new HTMLTableConverter();
        conv.setTable(table);
        HTMLTable[] html = conv.convertToTables(rowData);

        // Display the HTML table generated by the converter.
        System.out.println(html[0]);
```

HTMLTableConverter を使用して Java ソースから生成される HTML ソース

上記の Java ソースの例における HTMLTableConverter クラスの使用により、 以下の HTML コードが生成されます。

```
<table border="2" cellpadding="1" cellspacing="1">
<tr>
<th>Customer Number</th>
<th>Last Name</th>
<th>Initials</th>
<th>Street Address</th>
<th>City</th>
<th>State</th>
<th>Zip Code</th>
<th>Credit Limit</th>
<th>Charge Code</th>
<th>Balance Due</th>
<th>Credit Due</th>
</tr>
<tr>
<td>938472</td>
<td>Henning </td>
<td>G K</td>
<td>4859 Elm Ave </td>
<td>Dallas</td>
<td>TX</td>
<td align="right">75217</td>
<td align="right">5000</td>
<td align="right">3</td>
<td align="right">37.00</td>
<td align="right">0.00</td>
</tr>
<tr>
<td>839283</td>
<td>Jones   </td>
<td
>B D</td>
<td>21B NW 135 St</td>
<td>Clay  </td>
<td>NY</td>
<td align="right">13041</td>
<td align="right">400</td>
<td align="right">1</td>
<td align="right">100.00</td>
<td align="right">0.00</td>
</tr>
<tr>
<td>392859</td>
<td>Vine    </td>
<td>S S</td>
<td>PO Box 79    </td>
<td>Broton</td>
<td>VT</td>
<td align="right">5046</td>
```

```
<td align="right">700</td>
<td align="right">1</td>
<td align="right">439.00</td>
<td align="right">0.00</td>
</tr>
<tr>
<td>938485</td>
<td>Johnson </td>
<td>J A</td>
<td>3 Alpine Way </td>
<td>Helen </td>
<td>GA</td>
<td align="right">30545</td>
<td align="right">9999</td>
<td align="right">2</td>
<td align="right">3987.50</td>
<td align="right">33.50</td>
</tr>
<tr>
<td>397267</td>
<td>Tyron  </td>
<td>W E</td>
<td>13 Myrtle Dr </td>
<td>Hector</td>
<td>NY</td>
<td align="right">14841</td>
<td align="right">1000</td>
<td align="right">1</td>
<td align="right">0.00</td>
<td align="right">0.00</td>
</tr>
<tr>
<td>389572</td>
<td>Stevens </td>
<td>K L</td>
<td>208 Snow Pass</td>
<td>Denver</td>
<td>CO</td>
<td align="right">80226</td>
<td align="right">400</td>
<td align="right">1</td>
<td align="right">58.75</td>
<td align="right">1.50</td>
</tr>
<tr>
<td>846283</td>
<td>Alison  </td>
<td>J S</td>
<td>787 Lake Dr  </td>
<td>Isle  </td>
<td>MN</td>
<td align="right">56342</td>
<td align="right">5000</td>
<td align="right">3</td>
<td align="right">10.00</td>
```

```
<td align="right">0.00</td>
</tr>
<tr>
<td>475938</td>
<td>Doe     </td>
<td>J W</td>
<td>59 Archer Rd </td>
<td>Sutter</td>
<td>CA</td>
<td align="right">95685</td>
<td align="right">700</td>
<td align="right">2</td>
<td align="right">250.00</td>
<td align="right">100.00</td>
</tr>
<tr>
<td>693829</td>
<td>Thomas  </td>
<td>A N</td>
<td>3 Dove Circle</td>
<td>Casper</td>
<td>WY</td>
<td align="right">82609</td>
<td align="right">9999</td>
<td align="right">2</td>
<td align="right">0.00</td>
<td align="right">0.00</td>
</tr>
<tr>
<td>593029</td>
<td>Williams</td>
<td>E D</td>
<td>485 SE 2 Ave </td>
<td>Dallas</td>
<td>TX</td>
<td align="right">75218</td>
<td align="right">200</td>
<td align="right">1</td>
<td align="right">25.00</td>
<td align="right">0.00</td>
</tr>
<tr>
<td>192837</td>
<td>Lee     </td>
<td>F L</td>
<td>5963 Oak St  </td>
<td>Hector</td>
<td>NY</td>
<td align="right">14841</td>
<td align="right">700</td>
<td align="right">2</td>
<td align="right">489.50</td>
<td align="right">0.50</td>
</tr>
<tr>
```

```
<td>583990</td>
<td>Abraham </td>
<td>M T</td>
<td>392 Mill St  </td>
<td>Isle  </td>
<td>MN</td>
<td align="right">56342</td>
<td align="right">9999</td>
<td align="right">3</td>
<td align="right">500.00</td>
<td align="right">0.00</td>
</tr>
</table>
```

生成された HTML をブラウザーが表示する方法

以下の表は、この HTML ソース・コードがどのようにブラウザーで表示されるかを示します。

Customer Number	Last Name	Initials	Street Address	City	State	Zip Code	Credit Limit	Charge Code	Balance Due	Credit Due
938472	Henning	G K	4859 Elm Ave	Dallas	TX	75217	5000	3	37.00	0.00
839283	Jones	B D	21B NW 135 St	Clay	NY	13041	400	1	100.00	0.00
392859	Vine	S S	PO Box 79	Broton	VT	5046	700	1	439.00	0.00
938485	Johnson	J A	3 Alpine Way	Helen	GA	30545	9999	2	3987.50	33.50
397267	Tyron	W E	13 Myrtle Dr	Hector	NY	14841	1000	1	0.00	0.00
389572	Stevens	K L	208 Snow Pass	Denver	CO	80226	400	1	58.75	1.50
846283	Alison	J S	787 Lake Dr	Isle	MN	56342	5000	3	10.00	0.00
475938	Doe	J W	59 Archer Rd	Sutter	CA	95685	700	2	250.00	100.00
693829	Thomas	A N	3 Dove Circle	Casper	WY	82609	9999	2	0.00	0.00
593029	Williams	E D	485 SE 2 Ave	Dallas	TX	75218	200	1	25.00	0.00
192837	Lee	F L	5963 Oak St	Hector	NY	14841	700	2	489.50	0.50
583990	Abraham	M T	392 Mill St	Isle	MN	56342	9999	3	500.00	0.00

例: HTMLFormConverter を使用する

サーブレット・サポート付きの Web サーバーを稼働している状態で以下の例をコンパイルおよび実行することによって、 HTMLFormConverter の動作方法を確認してください。

```
import java.awt.Color;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Enumeration;
import java.util.Hashtable;
import javax.servlet.*;
import javax.servlet.http.*;

import com.ibm.as400.util.html.GridLayoutFormPanel;
import com.ibm.as400.util.html.HTMLConstants;
import com.ibm.as400.util.html.HTMLForm;
import com.ibm.as400.util.html.HTMLTable;
import com.ibm.as400.util.html.HTMLTableCaption;
import com.ibm.as400.util.html.HTMLText;
import com.ibm.as400.util.html.LabelFormElement;
import com.ibm.as400.util.html.LineLayoutFormPanel;
import com.ibm.as400.util.html.SubmitFormInput;
import com.ibm.as400.util.html.TextFormInput;

import com.ibm.as400.util.servlet.HTMLFormConverter;
import com.ibm.as400.util.servlet.SQLResultSetRowData;

import com.ibm.as400.access.AS400;
import com.ibm.as400.access.AS400JDBCDriver;

/**
 * An example of using the HTMLFormConverter class in a servlet.
 */
public class HTMLFormConverterExample extends HttpServlet

{
   private String userId_ = "myUserId";
   private String password_ = "myPwd";
   private AS400  system_;

   private Connection databaseConnection_;



   // Perform cleanup before returning to the main HTML form.
   public void cleanup()
   {
      try
      {
         // Close the database connection.
         if (databaseConnection_ != null)
         {
```

```
            databaseConnection_.close();
            databaseConnection_ = null;
         }
      }
      catch (Exception e)
      {
         e.printStackTrace ();
      }
   }


   // Convert the row data to formatted HTML.
   private HTMLTable[] convertRowData(SQLResultSetRowData rowData)
   {
      try
      {
         // Create the converter, which will generate HTML from
         // the result set that comes back from the database query.
         HTMLFormConverter converter = new HTMLFormConverter();

         // Set the form attributes.
         converter.setBorderWidth(3);
         converter.setCellPadding(2);
         converter.setCellSpacing(4);

         // Convert the row data to HTML.
         HTMLTable[] htmlTable = converter.convertToForms(rowData);
         return htmlTable;
      }
      catch (Exception e)
      {
         e.printStackTrace ();
         return null;
      }

   }


   // Return the response to the client.
   public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
   throws IOException, ServletException
   {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      out.println(showHtmlMain());
      out.close();
   }


   // Handle the data posted to the form.
   public void doPost (HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException
   {
      SQLResultSetRowData rowData = new SQLResultSetRowData();
      HTMLTable[] htmlTable = null;
```

```
// Get the current session object or create one if needed.
HttpSession session = request.getSession(true);

ServletOutputStream out = response.getOutputStream();

response.setContentType("text/html");

Hashtable parameters = getRequestParameters (request);

// Retrieve the row data and HTML table values for this session.
rowData = (SQLResultSetRowData) session.getValue("sessionRowData");
htmlTable = (HTMLTable[]) session.getValue("sessionHtmlTable");

// if this is the first time through, show first record
if (parameters.containsKey("getRecords"))
{
   rowData = getAllRecords(parameters, out);

   if (rowData != null)
   {
      // Set the row data value for this session.
      session.putValue("sessionRowData", rowData);

      // Position to the first record.
      rowData.first();

      // Convert the row data to formatted HTML.
      htmlTable = convertRowData(rowData);

      if (htmlTable != null)
      {
         rowData.first();
         session.putValue("sessionHtmlTable", htmlTable);
         out.println(showHtmlForRecord(htmlTable, 0));
      }
   }
}
// if the "Return To Main" button was pressed, go back to the main HTML form
else if (parameters.containsKey("returnToMain"))
{
   session.invalidate();
   cleanup();
   out.println(showHtmlMain());
}
// if the "First" button was pressed, show the first record
else if (parameters.containsKey("getFirstRecord"))
{
   rowData.first();
   out.println(showHtmlForRecord(htmlTable, 0));
}
// if the "Previous" button was pressed, show the previous record
else if (parameters.containsKey("getPreviousRecord"))
{
   if (!rowData.previous())
   {
      rowData.first();
```

```
      }
      out.println(showHtmlForRecord(htmlTable, rowData.getCurrentPosition()));
   }
   // if the "Next" button was pressed, show the next record
   else if (parameters.containsKey("getNextRecord"))
   {
      if (!rowData.next())
      {
         rowData.last();
      }
      out.println(showHtmlForRecord(htmlTable, rowData.getCurrentPosition()));
   }
   // if the "Last" button was pressed, show the last record
   else if (parameters.containsKey("getLastRecord"))
   {
      rowData.last();
      out.println(showHtmlForRecord(htmlTable, rowData.getCurrentPosition()));
   }
   // if none of the above, there must have been an error
   else
   {
      out.println(showHtmlForError("Internal error occurred.  Unexpected parameters."));
   }

   // Save the row data value for this session so the current position
   // is updated in the object associated with this session.
   session.putValue("sessionRowData", rowData);

   // Close the output stream
   out.close();
}


// Get all the records from the file input by the user.
private SQLResultSetRowData getAllRecords(Hashtable parameters, ServletOutputStream out)
throws IOException
{
   SQLResultSetRowData records = null;

   try
   {
      // Get the system, library and file name from the parameter list.
      String sys = ((String) parameters.get("System")).toUpperCase();
      String lib = ((String) parameters.get("Library")).toUpperCase();
      String file = ((String) parameters.get("File")).toUpperCase();
      if ((sys == null || sys.equals("")) ||
          (lib == null || lib.equals("")) ||
          (file == null || file.equals("")))
      {
         out.println(showHtmlForError("Invalid system, file or library name."));
      }
      else
      {
         // Get the connection to the server.
         getDatabaseConnection (sys, out);
         if (databaseConnection_ != null)
         {
```

```
                Statement sqlStatement = databaseConnection_.createStatement();

                // Query the database to get the result set.
                String query = "SELECT * FROM " + lib + "." + file;
                ResultSet rs = sqlStatement.executeQuery (query);

                boolean rsHasRows = rs.next();  // position cursor to first row

                // Show error message if the file contains no record;
                // otherwise, set row data to result set data.
                if (!rsHasRows)
                {
                   out.println(showHtmlForError("No records in the file."));
                }
                else
                {
                   records = new SQLResultSetRowData (rs);
                }

                // Don't close the Statement before we're done using the ResultSet
                // or bad things may happen.
                sqlStatement.close();
             }
          }
       }
       catch (Exception e)
       {
          e.printStackTrace ();
          out.println(showHtmlForError(e.toString()));
       }

       return records;
    }


    // Establish a database connection.
    private void getDatabaseConnection (String sysName, ServletOutputStream out)
    throws IOException
    {
       if (databaseConnection_ == null)
       {
          try
          {
             databaseConnection_ = DriverManager.getConnection("jdbc:as400://" + sysName, userId_,
password_ );
          }
          catch (Exception e)
          {
             e.printStackTrace ();
             out.println(showHtmlForError(e.toString()));
          }
       }
    }


    // Gets the parameters from an HTTP servlet request.
    private static Hashtable getRequestParameters (HttpServletRequest request)
```

```
{
   Hashtable parameters = new Hashtable ();
   Enumeration enum = request.getParameterNames();
   while (enum.hasMoreElements())
   {
      String key = (String) enum.nextElement();
      String value = request.getParameter (key);
      parameters.put (key, value);
   }
   return parameters;
}


// Get the servlet information.
public String getServletInfo()
{
   return "HTMLFormConverterExample";
}


// Perform initialization steps.
public void init(ServletConfig config)
{
   try
   {
      super.init(config);

      // Register the JDBC driver
      try
      {
         DriverManager.registerDriver(new AS400JDBCDriver());
      }
      catch (Exception e)
      {
         System.out.println("JDBC Driver not found");
      }
   }
   catch (Exception e)
   {
      e.printStackTrace();
   }
}


// Set the page header info.
private String showHeader(String title)
{
   StringBuffer page = new StringBuffer();
   page.append("<html><head><title>" + title + "</title>");
   page.append("</head><body bgcolor=/"blanchedalmond/">");
   return page.toString ();
}


// Show the HTML page with the appropriate error information.
private String showHtmlForError(String message)
{
```

```
    String title = "Error";
    StringBuffer page = new StringBuffer();
    page.append (showHeader (title));
    try
    {
       // Create the HTML Form object
       HTMLForm errorForm = new HTMLForm("HTMLFormConverterExample");

       // Set up so that doPost() gets called when the form is submitted.
       errorForm.setMethod(HTMLForm.METHOD_POST);

       // Create a single-column panel to which the HTML elements will be added.
       GridLayoutFormPanel grid = new GridLayoutFormPanel();

       // Create the text element for the error and add it to the panel.
       HTMLText text = new HTMLText(message);
       text.setBold(true);
       text.setColor(Color.red);
       grid.addElement(text);

       // Create the button to return to main and add it to the panel.
       grid.addElement(new SubmitFormInput("returnToMain", "Return to Main"));

       // Add the panel to the HTML form.
       errorForm.addElement(grid);

       page.append(errorForm.toString());
    }
    catch (Exception e)
    {
       e.printStackTrace ();
    }
    page.append("</body></html>");
    return page.toString();
}


// Show the HTML form for an individual record.
private String showHtmlForRecord(HTMLTable[] htmlTable, int position)
{
    String title = "HTMLFormConverter Example";
    StringBuffer page = new StringBuffer();
    page.append (showHeader (title));

    page.append("<h1>" + title + "</h1>");

    try
    {
       // Create the HTML Form object
       HTMLForm recForm = new HTMLForm("HTMLFormConverterExample");

       // Set up so that doPost() gets called when the form is submitted.
       recForm.setMethod(HTMLForm.METHOD_POST);

       // Set up a single-column panel layout, within which to arrange
       // the generated HTML elements.
       GridLayoutFormPanel grid = new GridLayoutFormPanel();
```

```
      // Create and add a table caption that keeps track of the current record.
      HTMLText recNumText = new HTMLText("Record number: " + (position + 1));
      recNumText.setBold(true);
      grid.addElement(recNumText);

      // Set up a two-column panel layout, within which to arrange
      // the table and text to comment on the converter output.
      GridLayoutFormPanel tableGrid = new GridLayoutFormPanel(2);
      tableGrid.addElement(htmlTable[position]);
      HTMLText comment = new HTMLText("   <---- Output from the HTMLFormConverter class");
      comment.setBold(true);
      comment.setColor(Color.blue);
      tableGrid.addElement(comment);

      // Add the table line to the panel.
      grid.addElement(tableGrid);

      // Set up a single-row panel layout, within which to arrange
      // the buttons for moving through the record list.
      LineLayoutFormPanel buttonLine = new LineLayoutFormPanel();
      buttonLine.addElement(new SubmitFormInput("getFirstRecord", "First"));
      buttonLine.addElement(new SubmitFormInput("getPreviousRecord", "Previous"));
      buttonLine.addElement(new SubmitFormInput("getNextRecord", "Next"));
      buttonLine.addElement(new SubmitFormInput("getLastRecord", "Last"));

      // Set up another single-row panel layout for the Return To Main button.
      LineLayoutFormPanel returnToMainLine = new LineLayoutFormPanel();
      returnToMainLine.addElement(new SubmitFormInput("returnToMain", "Return to Main"));

      // Add the lines containing the buttons to the grid panel.
      grid.addElement(buttonLine);
      grid.addElement(returnToMainLine);

      // Add the panel to the form.
      recForm.addElement(grid);

      // Add the form to the HTML page.
      page.append(recForm.toString());
   }
   catch (Exception e)
   {
      e.printStackTrace ();
   }

   page.append("</body></html>");
   return page.toString();
}


// Show the main HTML form (request input for system, file, and library name).
private String showHtmlMain()
{
   String title = "HTMLFormConverter Example";
   StringBuffer page = new StringBuffer();
   page.append (showHeader (title));
```

```
page.append("<h1>" + title + "</h1>");


// Create the HTML Form object
HTMLForm mainForm = new HTMLForm("HTMLFormConverterExample");

try
{
   // Set up so that doPost() gets called when the form is submitted.
   mainForm.setMethod(HTMLForm.METHOD_POST);

   // Add a brief description to the form.
   HTMLText desc = new HTMLText("<P>This example uses the HTMLFormConverter class " +
                                "to convert data retrieved from a server " +
                                "file.  The converter produces an array of HTML " +
                                "tables.  Each entry in the array is a record from " +
                                "the file.  " +
                                "Records are displayed one at a time, " +
                                "giving you buttons to move forward or backward " +
                                "through the list of records.</P>");
   mainForm.addElement(desc);

   // Add instructions to the form.
   HTMLText instr = new HTMLText("<P>Please input the name of the server, " +
                                 "and the file and library name for the file you " +
                                 "wish to access.  Then push the Show Records " +
                                 "button to continue.</P>");
   mainForm.addElement(instr);


   // Create a grid layout panel and add the system, file and library input fields.
   GridLayoutFormPanel panel = new GridLayoutFormPanel(2);

   LabelFormElement sysPrompt = new LabelFormElement("Server: ");
   TextFormInput system = new TextFormInput("System");
   system.setSize(10);

   LabelFormElement filePrompt = new LabelFormElement("File name: ");
   TextFormInput file = new TextFormInput("File");
   file.setSize(10);

   LabelFormElement libPrompt = new LabelFormElement("Library name: ");
   TextFormInput library = new TextFormInput("Library");
   library.setSize(10);

   panel.addElement(sysPrompt);
   panel.addElement(system);
   panel.addElement(filePrompt);
   panel.addElement(file);
   panel.addElement(libPrompt);
   panel.addElement(library);

   // Add the panel to the form.
   mainForm.addElement(panel);

   // Create the submit button and add it to the form.
   mainForm.addElement(new SubmitFormInput("getRecords", "Show Records"));
```

```
      }
      catch (Exception e)
      {
         e.printStackTrace ();
      }

      page.append(mainForm.toString());
      page.append("</body></html>");

      return page.toString();
   }

}
```

上記の例で生成された HTML は以下のようになります。

```
<table border="0">
<tr>
<td><b>Record number: 1</b></td>
</tr>
<tr>
<td><table border="0">
<tr>
<td><table border="3" cellpadding="2" cellspacing="4">
<tr>
<th>CUSNUM</th>
<td>839283</td>
</tr>
<tr>
<th>LSTNAM</th>
<td>Jones   </td>
</tr>
<tr>
<th>INIT</th>
<td>B D</td>
</tr>
<tr>
<th>STREET</th>
<td>21B NW 135 St</td>
</tr>
<tr>
<th>CITY</th>
<td>Clay  </td>
</tr>
<tr>
<th>STATE</th>
<td>NY</td>
</tr>
<tr>
<th>ZIPCOD</th>
<td>13041</td>
</tr>
<tr>
<th>CDTLMT</th>
<td>400</td>
</tr>
<tr>
```

```
<th>CHGCOD</th>
<td>1</td>
</tr>
<tr>
<th>BALDUE</th>
<td>100.00</td>
</tr>
<tr>
<th>CDTDUE</th>
<td>0.00</td>
</tr>
</table>
</td>
<td><font color="#0000ff"> <b><!-- Output from the HTMLFormConverter class-->
</b></font></td>
</tr>
</table>
</td>
</tr>
<tr>
<form>
<td><input type="submit" name="getFirstRecord" value="First" />
<input type="submit" name="getPreviousRecord" value="Previous" />
<input type="submit" name="getNextRecord" value="Next" />
<input type="submit" name="getLastRecord" value="Last" />
<br />
</td>
</tr>
<tr>
<td><input type="submit" name="returnToMain" value="Return to Main" />
<br />
</td>
</tr>
</table>
</form>
```

HTML とサーブレット・クラスの Lights On の例

この例では、HTML およびサーブレット・クラスの動作方法が示されています。 これは、一般的な概要です。 この例を表示するには、 Web サーバーおよびブラウザーが稼働されている状態でコンパイルしてから実行してください。

```
import java.io.IOException;
import java.io.CharArrayWriter;
import java.io.PrintWriter;
import java.sql.*;
import java.util.Enumeration;
import java.util.Hashtable;
import javax.servlet.*;
import javax.servlet.http.*;

import com.ibm.as400.util.html.*;
import com.ibm.as400.util.servlet.*;
import com.ibm.as400.access.*;

/*
An example of using Toolbox classes in a servlet.

Schemas of SQL databases on the server:

File . . . . . .   LICENSES
Library  . . .     LIGHTSON

 Field            Type          Length   Nulls
 LICENSE          CHARACTER         10   NOT NULL
 USER_ID          CHARACTER         10   NOT NULL WITH DEFAULT
 E_MAIL           CHARACTER         20   NOT NULL
 WHEN_ADDED       DATE                   NOT NULL WITH DEFAULT
 TIME_STAMP       TIMESTAMP              NOT NULL WITH DEFAULT


File . . . . . .   REPORTS
Library  . . .     LIGHTSON

 Field            Type          Length   Nulls
 LICENSE          CHARACTER         10   NOT NULL
 REPORTER         CHARACTER         10   NOT NULL WITH DEFAULT
 DATE_ADDED       DATE                   NOT NULL WITH DEFAULT
 TIME_ADDED       TIME                   NOT NULL WITH DEFAULT
 TIME_STAMP       TIMESTAMP              NOT NULL WITH DEFAULT
 LOCATION         CHARACTER         10   NOT NULL
 COLOR            CHARACTER         10   NOT NULL
 CATEGORY         CHARACTER         10   NOT NULL
*/

public class LightsOn extends javax.servlet.http.HttpServlet

{
  private AS400  system_;
```

```
private String password_;  // password for the server and for the SQL database
private java.sql.Connection databaseConnection_;



public void destroy (ServletConfig config)
{
  try {
    if (databaseConnection_ != null) {
      databaseConnection_.close();
    }
  }
  catch (Exception e) { e.printStackTrace (); }
}

public void doGet (HttpServletRequest request,
                   HttpServletResponse response)
  throws IOException, ServletException
{
  HttpSession session = request.getSession();

  response.setContentType("text/html");
  PrintWriter out = response.getWriter();

  out.println(showHtmlMain());

  out.close();
}

public void doPost (HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException
{
  HttpSession session = request.getSession(true);
  ServletOutputStream out = response.getOutputStream();
  response.setContentType("text/html");

  Hashtable parameters = getRequestParameters (request);

  if (parameters.containsKey("askingToReport"))
    out.println (showHtmlForReporting ());
  else if (parameters.containsKey("askingToRegister"))
    out.println (showHtmlForRegistering ());
  else if (parameters.containsKey("askingToUnregister"))
    out.println(showHtmlForUnregistering());
  else if (parameters.containsKey("askingToListRegistered"))
    out.println (showHtmlForListingAllRegistered ());
  else if (parameters.containsKey("askingToListReported"))
    out.println (showHtmlForListingAllReported ());
  else if (parameters.containsKey("returningToMain"))
    out.println (showHtmlMain ());


  else { // None of the above, so assume the user has filled out a form
         // and is submitting information.  Grab the incoming info
```

```
        // and do the requested action.

    if (parameters.containsKey("submittingReport")) {
      String acknowledgement = reportLightsOn (parameters, out);
      out.println (showAcknowledgement(acknowledgement));
    }

    else if (parameters.containsKey("submittingRegistration")) {
      String acknowledgement = registerLicense (parameters, out);
      out.println (showAcknowledgement(acknowledgement));
    }

    else if (parameters.containsKey("submittingUnregistration")) {
      String acknowledgement = unregisterLicense (parameters, out);
      out.println (showAcknowledgement(acknowledgement));
    }

    else {
      out.println (showAcknowledgement("Error (internal): " +
                        "Neither Report, Register, " +
                        "Unregister, ListRegistered, or ListReported."));
    }
  }

  out.close(); // Close the output stream.

}

// Gets the parameters from an HTTP servlet request, and packages them
// into a hashtable for convenience.
private static Hashtable getRequestParameters (HttpServletRequest request)
{
  Hashtable parameters = new Hashtable ();
  Enumeration enum = request.getParameterNames();
  while (enum.hasMoreElements()) {
    String key = (String) enum.nextElement();
    String value = request.getParameter (key);
    parameters.put (key, value);
  }
  return parameters;
}


// Removes blanks and hyphens from a String, and sets it to all uppercase.
private static String normalize (String oldString)
{
  if (oldString == null || oldString.length() == 0) return null;
  StringBuffer newString = new StringBuffer ();
  for (int i=0; i<oldString.length(); i++) {
    if (oldString.charAt(i) != ' ' && oldString.charAt(i) != '-')
      newString.append (oldString.charAt(i));
  }
  return newString.toString().toUpperCase();
}
```

```
// Composes a list of single-quoted strings.
private static String quoteList (String[] inList)
{
  StringBuffer outList = new StringBuffer();
  for (int i=0; i<inList.length; i++)
  {
    outList.append ("'" + inList[i] + "'");
    if (i<inList.length-1)
      outList.append (",");
  }
  return outList.toString();
}


public String getServletInfo ()
{
  return "Lights-On Servlet";
}

private AS400 getSystem ()
{
  try
  {
    if (system_ == null)
    {
      system_ = new AS400();

      // Note: It would be better to get these values
      // from a properties file.
      String sysName  = "MYSYSTEM";    // TBD
      String userId   = "MYUSERID";    // TBD
      String password = "MYPASSWD";    // TBD

      system_.setSystemName(sysName);
      system_.setUserId(userId);
      system_.setPassword(password);
      password_ = password;

      system_.connectService(AS400.DATABASE);
      system_.connectService(AS400.FILE);
      system_.addPasswordCacheEntry(sysName, userId, password_);
    }
  }
  catch (Exception e) { e.printStackTrace (); system_ = null; }

  return system_;
}


public void init (ServletConfig config)
{
  boolean rc;
```

```
  try {
    super.init(config);

    // Register the JDBC driver.
    try {
      java.sql.DriverManager.registerDriver(new com.ibm.as400.access.AS400JDBCDriver());
    }
    catch (Exception e)
    {
      System.out.println("JDBC Driver not found");
    }


  }
  catch (Exception e) { e.printStackTrace(); }
}

private void getDatabaseConnection ()
{
  if (databaseConnection_ == null) {
    try {
      databaseConnection_ = java.sql.DriverManager.getConnection(
                "jdbc:as400://" + getSystem().getSystemName() + "/" +
                "LIGHTSON", getSystem().getUserId(), password_ );

    }
    catch (Exception e) { e.printStackTrace (); }
  }
}


private String registerLicense (Hashtable parameters, ServletOutputStream out)
{
  String licenseNum = normalize ((String)parameters.get("licenseNum"));
  String eMailAddress = (String)parameters.get("eMailAddress");
  StringBuffer acknowledgement = new StringBuffer();

  if (licenseNum == null || licenseNum.length() == 0)
    acknowledgement.append ("Error: License number not specified./n");

  if (eMailAddress == null || eMailAddress.length() == 0)
    acknowledgement.append ("Error: Notification e-mail address not specified./n");

  if (acknowledgement.length() == 0)
  {
    try
    {
      // Insert the new license number and e-mail address into the database.
      getDatabaseConnection ();
      Statement sqlStatement = databaseConnection_.createStatement();

      // Issue the request.
      String cmd = "INSERT INTO LICENSES " +
        "(LICENSE, E_MAIL) VALUES (" +
```

```
        quoteList (new String[] {licenseNum, eMailAddress} ) +
        ")";
      sqlStatement.executeUpdate(cmd);
      sqlStatement.close();

      // Acknowledge the request.
      acknowledgement.append ("License number " + licenseNum + " has been registered.");
      acknowledgement.append ("Notification e-mail address is: " + eMailAddress);
    }
    catch (Exception e) { e.printStackTrace (); }
  }
  return acknowledgement.toString();
}


private String unregisterLicense (Hashtable parameters, ServletOutputStream out)
{
  String licenseNum = normalize ((String)parameters.get("licenseNum"));
  StringBuffer acknowledgement = new StringBuffer();

  if (licenseNum == null || licenseNum.length() == 0)
    acknowledgement.append ("Error: License number not specified./n");

  if (acknowledgement.length() == 0)
  {
    try
    {
      // Remove the specified license number and e-mail address from database.
      getDatabaseConnection ();
      Statement sqlStatement = databaseConnection_.createStatement();

      // Delete the row(s) from the LICENSES database.
      String cmd = "DELETE FROM LICENSES WHERE LICENSE = '" + licenseNum + "'";
      sqlStatement.executeUpdate(cmd);
      sqlStatement.close();

      // Acknowledge the request.
      acknowledgement.append ("License number " + licenseNum + " has been unregistered.");
    }
    catch (Exception e) { e.printStackTrace (); }
  }
  return acknowledgement.toString();
}


private String reportLightsOn (Hashtable parameters, ServletOutputStream out)
{
  String licenseNum = normalize ((String)parameters.get("licenseNum"));
  String location = (String)parameters.get("location");
  String color = (String)parameters.get("color");
  String category = (String)parameters.get("category");
  StringBuffer acknowledgement = new StringBuffer();
  if (licenseNum == null || licenseNum.length() == 0)
    acknowledgement.append ("Error: License number not specified.");
```

```
  if (acknowledgement.length() == 0)
  {
    try
    {
      // Report "lights on" for a specified vehicle.
      getDatabaseConnection ();
      Statement sqlStatement = databaseConnection_.createStatement();

      // Add an entry to the REPORTS database.
      String cmd = "INSERT INTO REPORTS " +
        "(LICENSE, LOCATION, COLOR, CATEGORY) VALUES (" +
        quoteList (new String[] {licenseNum, location, color, category} ) +
        ")";
      sqlStatement.executeUpdate(cmd);
      sqlStatement.close();

      // Acknowledge the request.
      acknowledgement.append ("License number " + licenseNum +
                              " has been reported.  Thanks!");
    }
    catch (Exception e) { e.printStackTrace (); }
  }
  return acknowledgement.toString();
}


private String showHeader (String title)
{
  StringBuffer page = new StringBuffer();
  page.append("<html><head><title>" + title + "</title>");
  page.append("</head><body bgcolor=/"blanchedalmond/">");
  return page.toString ();
}


private String showAcknowledgement (String acknowledgement)
{
  String title = "Acknowledgement";
  StringBuffer page = new StringBuffer();
  page.append (showHeader (title));

  try {
    HTMLForm form = new HTMLForm("LightsOn");
    GridLayoutFormPanel grid = new GridLayoutFormPanel();
    HTMLText text = new HTMLText(acknowledgement);
    if (acknowledgement.startsWith("Error")) text.setBold(true);
    grid.addElement(text);
    grid.addElement(new SubmitFormInput("returningToMain", "Home"));
    form.addElement(grid);
    page.append(form.toString());
  }
  catch (Exception e) { e.printStackTrace (); }
  page.append("</body></html>");
```

```
  return page.toString();
}


private String showHtmlMain ()
{
  String title = "Lights-On tool";
  StringBuffer page = new StringBuffer();
  page.append (showHeader (title));

  page.append("<h1>" + title + "</h1>");


  // Create the HTML Form object.
  HTMLForm mainForm = new HTMLForm("LightsOn");
  GridLayoutFormPanel grid = new GridLayoutFormPanel();

  try {
    // Set up so that doPost() gets called when the form is submitted.
    mainForm.setMethod(HTMLForm.METHOD_POST);

    // Create some buttons.
    grid.addElement(new SubmitFormInput("askingToReport",
                                        "Report a vehicle with lights on"));
    grid.addElement(new SubmitFormInput("askingToRegister",
                                        "Register my license number"));
    grid.addElement(new SubmitFormInput("askingToUnregister",
                                        "Unregister my license number"));
    grid.addElement(new SubmitFormInput("askingToListRegistered",
                                        "List all registered licenses"));
    grid.addElement(new SubmitFormInput("askingToListReported",
                                        "List all vehicles with lights on"));

    mainForm.addElement(grid);
  }
  catch (Exception e) { e.printStackTrace (); }

  page.append(mainForm.toString());
  page.append("</body></html>");

  return page.toString();
}


private String showHtmlForReporting ()
{
  String title = "Report a vehicle with lights on";
  StringBuffer page = new StringBuffer();
  page.append (showHeader (title));

  page.append("<h1>" + title + "</h1>");

  // Create the HTML Form object.
  HTMLForm reportForm = new HTMLForm("LightsOn");
```

```
GridLayoutFormPanel grid = new GridLayoutFormPanel(2);

try {
  // Set up so that doPost() gets called when the form is submitted.
  reportForm.setMethod(HTMLForm.METHOD_POST);

  TextFormInput licenseNum = new TextFormInput("licenseNum");
  licenseNum.setSize(10);
  licenseNum.setMaxLength(10);

  // Add elements to the line form
  grid.addElement(new LabelFormElement("Vehicle license number:"));
  grid.addElement(licenseNum);

  // Create a radio button group and add the radio buttons.
  RadioFormInputGroup colorGroup = new RadioFormInputGroup("color");

  colorGroup.add("color", "white", "white", true);
  colorGroup.add("color", "black", "black", false);
  colorGroup.add("color", "gray", "gray", false);
  colorGroup.add("color", "red", "red", false);
  colorGroup.add("color", "yellow", "yellow", false);
  colorGroup.add("color", "green", "green", false);
  colorGroup.add("color", "blue", "blue", false);
  colorGroup.add("color", "brown", "brown", false);

  // Create a selection list for category of vehicle.
  SelectFormElement category = new SelectFormElement("category");
  category.addOption("sedan", "sedan", true);
  category.addOption("convertible", "convertibl"); // 10-char field in DB
  category.addOption("truck", "truck");
  category.addOption("van", "van");
  category.addOption("SUV", "SUV");
  category.addOption("motorcycle", "motorcycle");
  category.addOption("other", "other");

  // Create a selection list for vehicle location (building number).
  SelectFormElement location = new SelectFormElement("location");
  location.addOption("001", "001", true);
  location.addOption("002", "002");
  location.addOption("003", "003");
  location.addOption("005", "005");
  location.addOption("006", "006");
  location.addOption("015", "015");

  grid.addElement(new LabelFormElement("Color:"));
  grid.addElement(colorGroup);

  grid.addElement(new LabelFormElement("Vehicle type:"));
  grid.addElement(category);

  grid.addElement(new LabelFormElement("Building:"));
  grid.addElement(location);
```

```
    grid.addElement(new SubmitFormInput("submittingReport", "Submit report"));
    grid.addElement(new SubmitFormInput("returningToMain", "Home"));

    reportForm.addElement(grid);
  }
  catch (Exception e) { e.printStackTrace (); }

  page.append(reportForm.toString());

  page.append("</body></html>");

  return page.toString();
}


private String showHtmlForRegistering ()
{
  String title = "Register my license number";
  StringBuffer page = new StringBuffer();
  page.append (showHeader (title));

  page.append("<h1>" + title + "</h1>");

  // Create the HTML Form object.
  HTMLForm registrationForm = new HTMLForm("LightsOn");

  // Set up a two-column panel layout, within which to arrange
  // the generated HTML elements.
  GridLayoutFormPanel grid = new GridLayoutFormPanel(2);

  try {
    // Set up so that doPost() gets called when the form is submitted.
    registrationForm.setMethod(HTMLForm.METHOD_POST);

    TextFormInput licenseNum = new TextFormInput("licenseNum");
    licenseNum.setSize(10);
    licenseNum.setMaxLength(10);

    TextFormInput eMailAddress = new TextFormInput("eMailAddress");
    eMailAddress.setMaxLength(20);

    grid.addElement(new LabelFormElement("License number:"));
    grid.addElement(licenseNum);

    grid.addElement(new LabelFormElement("E-mail notification address:"));
    grid.addElement(eMailAddress);

    grid.addElement(new SubmitFormInput("submittingRegistration", "Register"));
    grid.addElement(new SubmitFormInput("returningToMain", "Home"));

    registrationForm.addElement(grid);
  }
  catch (Exception e) { e.printStackTrace (); }
```

```
  page.append(registrationForm.toString());

  page.append("</body></html>");

  return page.toString();
}


private String showHtmlForUnregistering ()
{
  String title = "Unregister my license number";
  StringBuffer page = new StringBuffer();
  page.append (showHeader (title));

  page.append("<h1>" + title + "</h1>");

  // Create the HTML Form object.
  HTMLForm unregistrationForm = new HTMLForm("LightsOn");
  GridLayoutFormPanel grid = new GridLayoutFormPanel(2);

  try {
    // Set up so that doPost() gets called when the form is submitted.
    unregistrationForm.setMethod(HTMLForm.METHOD_POST);

    // Create the LineLayoutFormPanel object.
    TextFormInput licenseNum = new TextFormInput("licenseNum");
    licenseNum.setSize(10);
    licenseNum.setMaxLength(10);

    grid.addElement(new LabelFormElement("Vehicle license number:"));
    grid.addElement(licenseNum);

    grid.addElement(new SubmitFormInput("submittingUnregistration", "Unregister"));
    grid.addElement(new SubmitFormInput("returningToMain", "Home"));

    unregistrationForm.addElement(grid);
  }
  catch (Exception e) {
    e.printStackTrace ();
    CharArrayWriter cWriter = new CharArrayWriter();
    PrintWriter pWriter = new PrintWriter (cWriter, true);
    e.printStackTrace (pWriter);
    page.append (cWriter.toString());
  }

  page.append(unregistrationForm.toString());

  page.append("</body></html>");

  return page.toString();
}


private String showHtmlForListingAllRegistered ()
```

```
{
  String title = "All registered licenses";
  StringBuffer page = new StringBuffer();
  page.append (showHeader (title));

  try
  {
    // Create the HTML Form object.
    HTMLForm mainForm = new HTMLForm("LightsOn");

    // Set up a single-column panel layout, within which to arrange
    // the generated HTML elements.
    GridLayoutFormPanel grid = new GridLayoutFormPanel();

    // Specify the layout for the generated table.
    HTMLTable table = new HTMLTable();
    table.setAlignment(HTMLConstants.LEFT);
    table.setBorderWidth(3);

    // Create and add the table caption and header.
    HTMLTableCaption caption = new HTMLTableCaption();
    caption.setAlignment(HTMLConstants.TOP);
    caption.setElement(title);
    table.setCaption(caption);
    table.setHeader(new String[] { "License", "Date added" } );

    // Create the converter, which will generate table HTML from
    // the result set that comes back from the database query.
    HTMLTableConverter converter = new HTMLTableConverter();
    converter.setTable(table);

    getDatabaseConnection ();
    Statement sqlStatement = databaseConnection_.createStatement();

    // First pre-query the database to verify that it's not empty.
    String query = "SELECT COUNT(*) FROM LICENSES";
    ResultSet rs = sqlStatement.executeQuery (query);
    rs.next();  // position cursor to first row
    int rowCount = rs.getInt(1);

    if (rowCount == 0) {
      page.append ("<font size=4 color=red>No vehicles have been reported.</font>");
    }
    else {
      query = "SELECT LICENSE,WHEN_ADDED FROM LICENSES";
      rs = sqlStatement.executeQuery (query);
      SQLResultSetRowData rowData = new SQLResultSetRowData (rs);
      HTMLTable[] generatedHtml = converter.convertToTables(rowData);
      grid.addElement(generatedHtml[0]);
    }
    sqlStatement.close();
    // Note: Mustn't close statement before we're done using result set.

    grid.addElement(new SubmitFormInput("returningToMain", "Home"));
```

```
    mainForm.addElement(grid);
    page.append(mainForm.toString());
  }
  catch (Exception e) { e.printStackTrace (); }
  page.append("</body></html>");
  return page.toString();
}


private String showHtmlForListingAllReported ()
{
  String title = "All vehicles with lights on";
  StringBuffer page = new StringBuffer();
  page.append (showHeader (title));

  try
  {
    // Create the HTML Form object.
    HTMLForm form = new HTMLForm("LightsOn");

    // Set up a single-column panel layout, within which to arrange
    // the generated HTML elements.
    GridLayoutFormPanel grid = new GridLayoutFormPanel();

    // Specify the layout for the generated table.
    HTMLTable table = new HTMLTable();
    table.setAlignment(HTMLConstants.LEFT);
    table.setBorderWidth(3);

    // Create and add the table caption and header.
    HTMLTableCaption caption = new HTMLTableCaption();
    caption.setAlignment(HTMLConstants.TOP);
    caption.setElement(title);
    table.setCaption(caption);
    table.setHeader(new String[] { "License", "Color", "Category", "Date", "Time" } );

    // Create the converter, which will generate table HTML from
    // the result set that comes back from the database query.
    HTMLTableConverter converter = new HTMLTableConverter();
    converter.setTable(table);

    getDatabaseConnection ();
    Statement sqlStatement = databaseConnection_.createStatement();

    // First pre-query the database to verify that it's not empty.
    String query = "SELECT COUNT(*) FROM REPORTS";
    ResultSet rs = sqlStatement.executeQuery (query);
    rs.next();  // position cursor to first row
    int rowCount = rs.getInt(1);

    if (rowCount == 0) {
      page.append ("<font size=4 color=red>No vehicles have been reported.</font>");
    }
```

```
      else {
        query = "SELECT LICENSE,COLOR,CATEGORY,DATE_ADDED,TIME_ADDED FROM REPORTS";
        rs = sqlStatement.executeQuery (query);
        SQLResultSetRowData rowData = new SQLResultSetRowData (rs);
        HTMLTable[] generatedHtml = converter.convertToTables(rowData);
        grid.addElement(generatedHtml[0]);
      }
      sqlStatement.close();
      // Note: Mustn't close statement before we're done using result set.

      grid.addElement(new SubmitFormInput("returningToMain", "Home"));
      form.addElement(grid);
      page.append(form.toString());
    }
    catch (Exception e) { e.printStackTrace (); }
    page.append("</body></html>");
    return page.toString();
  }

}
```

初めて Toolbox for Java プログラムを書く

この簡単な練習を始めるには、ワークステーションに Java がインストール済みである必要があります。 Java アプリケーションを実行するための要件を検討して、 どのバージョンをインストールするか決めることができます。

Java をクライアントにインストールした後、以下の作業を行ってください。

1. ワークステーションに jt400.jar をコピーします。
2. JAR ファイルの絶対パスを CLASSPATH に追加することにより、jt400.jar をワークステーションの CLASSPATH に追加します。 たとえば、jt400.jar ファイルがワークステーション (Windows が実行している) の c:\lib ディレクトリーにある場合、 以下のファイルを CLASSPATH ステートメントの末尾に追加します。

   ```
   ;c:/lib/jt400.jar
   ```

3. テキスト・エディターを開き、 最初の簡単なプログラミング例を入力します。

   注: 注釈を参照するテキスト (たとえば、「Note 1」、「Note 2」など) は確実に除外してください。 新規文書を CmdCall.java という名前で保管します。

4. コマンド・セッションをワークステーションで開始し、以下のコマンドを使用して簡単なプログラミング例をコンパイルします。

   ```
   javac CmdCall.java
   ```

5. コマンド・セッションで、以下のコマンドを入力して簡単なプログラミング例を実行します。

   ```
   java CmdCall
   ```

別個のウィンドウで詳細説明を参照するためには、ブラウザーの JavaScript サポートが使用可能になっている必要があります。 ブラウザーの JavaScript が使用可能になっていない場合あるいは JavaScript をサポートしない場合には、 注釈 (Note) のテキスト・リンクをクリックするか、 ページの最後までスクロールして説明を参照してください。

[簡単なプログラミング例]

例: CommandCall を使用する

以下のプログラム例を参考にしてプログラムを作成してください。 例には、コード内のかぎとなる行についての詳細説明が含まれています。 以下の方法で、詳細説明を表示することができます。

- 詳細説明を表示するには、ポップアップ・ウィンドウの  イメージをクリックします。
- テキスト・リンクをクリックして、例の最後にある詳細説明を参照します。

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// Example using the IBM Toolbox for Java's access class, CommandCall.
//
// This source is an example of IBM Toolbox for Java "Job List".
//
///////////////////////////////////////////////////////////////////////////
//
// The access classes of the Toolbox are in the com.ibm.as400.access.package.
// Import this package to use the Toolbox classes.
//
///////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;

public class CmdCall
{
   public static void main(String[] args)
   {
      // Like other Java classes the Toolbox classes throw
      // exceptions when something goes wrong.  These must be
      // caught by programs that use the Toolbox.
      tryNote 1
      {
         AS400 system  = new AS400();

         CommandCall cc = new CommandCall(system);Note 2
```

```
            cc.run("CRTLIB MYLIB");Note 3

            AS400Message[] ml = cc.getMessageList();Note 4

            for (int i=0; i<ml.length; i++)
            {
               System.out.println(ml[i].getText());Note 5
            }
         }
         catch (Exception e)
         {
            e.printStackTrace();
         }

         System.exit(0);
      }
}
```

1. Toolbox for Java は、 "AS400" オブジェクトを使用してターゲット・サーバーを識別します。 パラメーターを指定せずに AS400 オブジェクトを構成する場合、Toolbox for Java は、 システム名、ユーザー ID、およびパスワードの入力を求めるプロンプトを出します。 AS400 クラスにはさらに、システム名、ユーザー ID、およびパスワードを取るコンストラクターも含まれます。
2. Toolbox for Java の CommandCall オブジェクトを使用してコマンドをサーバーに送信します。 CommandCall オブジェクトを作成する時に、AS400 オブジェクトをそれに渡し、 コマンドのターゲットがどのサーバーであるかがわかるようにします。
3. コマンド呼び出しオブジェクト上で run() メソッドを使用してコマンドを実行します。
4. コマンドの実行の結果は、OS/400 メッセージのリストになります。 Toolbox は、これらのメッセージを AS400Message オブジェクトとして表します。 コマンドが完了すると、結果のメッセージを CommandCall オブジェクトから受け取ります。
5. メッセージ・テキストを印刷します。 さらに、メッセージ ID、メッセージ重大度、その他の情報も入手可能です。 このプログラムはメッセージ・テキストだけを印刷します。

[簡単なプログラミング例]

別個のウィンドウで詳細説明を参照するためには、ブラウザーの JavaScript サポートが使用可能になっている必要があります。 ブラウザーの JavaScript が使用可能になっていない場合あるいは JavaScript をサポートしない場合には、 注釈 (Note) のテキスト・リンクをクリックするか、 ページの最後までスクロールして説明を参照してください。

[次の部 | 簡単なプログラミング例]

例: メッセージ待ち行列を使用する (3 部の 1)

以下のプログラム例を参考にしてプログラムを作成してください。 例には、コード内のかぎとなる行についての詳細説明が含まれています。 以下の方法で、詳細説明を表示することができます。

- 詳細説明を表示するには、ポップアップ・ウィンドウの  イメージをクリックします。
- テキスト・リンクをクリックして、例の最後にある詳細説明を参照します。

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// Example using the Message Queue function of the IBM Toolbox for Java
//
// This source is an example of IBM Toolbox for Java "Message Queue".
//
///////////////////////////////////////////////////////////////////////////

package examples; Note 1


import java.io.*;
import java.util.*;

import com.ibm.as400.access.*; Note 2

public class displayMessages extends Object
{

   public static void main(String[] parameters) Note 3
   {
      displayMessages me = new displayMessages();
      me.Main(parameters); Note 4

      System.exit(0); Note 5
   }
```

```
    void displayMessage()
    {
    }


    void Main(String[] parms)
    {
       try Note 6
       {

                // IBM Toolbox for Java code goes here
       }
       catch (Exception e)
       {
          e.printStackTrace(); Note 7
       }
    }
}
```

1. このクラスは、'例' パッケージ内にあります。 Java は Java クラス・ファイル間の名前の競合を避けるためにパッケージを使用します。
2. この行は、access パッケージ内のすべての IBM Toolbox for Java クラスを このプログラムが使用できるようにします。 access パッケージ内のクラスには、共通接頭部として com.ibm.as400 が付いています。 インポート・ステートメントを使用することにより、 プログラムは完全修飾名ではなく名前だけでクラスを参照することができます。 たとえば、com.ibm.as400.AS400 ではなく、AS400 を使用することにより、 AS400 クラスを参照することができます。
3. このクラスには、アプリケーションとして実行できるようにする main メソッドがあります。 プログラムを起動するには、java examples.displayMessages を実行します。 プログラムの実行時には、大文字小文字を一致させる必要があります。 IBM Toolbox for Java クラスが使用されるので、 jt400.zip を CLASSPATH 環境変数で指定しておかなければなりません。
4. 注 3 で言及されている main メソッドは静的です。 静的メソッドの制限の 1 つは、 静的メソッドはそのクラス内の他の静的メソッドしか呼び出せないことです。 この制限を避けるためには、 多数の java プログラムがオブジェクトを作成した後、 Main と呼ばれるメソッドで初期化処理を実行します。 Main() メソッドは、displayMessages オブジェクト内の他のメソッドを呼び出すことができます。
5. IBM Toolbox for Java アクティビティーを実行するアプリケーションのために、 IBM Toolbox for Java はスレッドを作成します。 終了時にプログラムが System.exit(0) を発行しないと、 プログラムは正常に終了しないことがあります。 たとえば、このプログラムを Windows 95 DOS プロンプトから実行したとすると、 上記の行がなければ、プログラムの終了時にコマンド・プロンプトに戻りません。 ユーザーが Ctrl-C を入力して、コマンド・プロンプトを表示する必要があります。
6. IBM Toolbox for Java コードは、ユーザーのプログラムがキャッチしなければならない例外を出します。

7. このプログラムはエラーを処理する際に、例外のテキストを表示します。 IBM Toolbox for Java によって出された例外は変換され、 例外のテキストがワークステーションの言語と同じものになるようにされます。

[次の部 | 簡単なプログラミング例]

別個のウィンドウで詳細説明を参照するためには、ブラウザーの JavaScript サポートが使用可能になっている必要があります。 ブラウザーの JavaScript が使用可能になっていない場合あるいは JavaScript をサポートしない場合には、 注釈 (Note) のテキスト・リンクをクリックするか、 ページの最後までスクロールして説明を参照してください。

[前の部 ¦ 次の部 ¦ 簡単なプログラミング例]

例: メッセージ待ち行列を使用する (3 部の 2)

以下のプログラム例を参考にしてプログラムを作成してください。 例には、コード内のかぎとなる行についての詳細説明が含まれています。 以下の方法で、詳細説明を表示することができます。

- 詳細説明を表示するには、ポップアップ・ウィンドウの  イメージをクリックします。
- テキスト・リンクをクリックして、例の最後にある詳細説明を参照します。

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// Example using the Message Queue function of the IBM Toolbox for Java
//
// This source is an example of IBM Toolbox for Java "Message Queue".
//
///////////////////////////////////////////////////////////////////////////

package examples;


import java.io.*;
import java.util.*;
import com.ibm.as400.access.*;

public class displayMessages extends Object
{

   public static void main(String[] parameters)
   {
      displayMessages me = new displayMessages();

      me.Main(parameters);

      System.exit(0);
   }


   void displayMessage()
```

```
    {
    }


    void Main(String[] parms)
    {
       try
       {

          AS400 system = new AS400(); Note 1


          if (parms.length > 0)
                        system.setSystemName(parms[0]); Note 2

       }
       catch (Exception e)
       {
          e.printStackTrace();
       }
    }
}
```

1. プログラムは AS400 オブジェクトを使用して、 どのサーバーが接続するかを指定します。 サーバーからのリソースを必要とするすべてのプログラムは、AS400 オブジェクトを持っている必要があります。 ただし、1 つの例外として、JDBC の場合は異なります。 ご使用のプログラムが JDBC を使用する場合、 IBM Toolbox for Java JDBC ドライバーがそのプログラム用の AS400 オブジェクトを作成します。
2. このプログラムは、最初のコマンド行パラメーターがサーバーの名前であると想定します。 パラメーターがプログラムに渡されると、 AS400 オブジェクトの setSystemName メソッドを使用してシステム名が設定されます。 また、AS400 オブジェクトは サーバー・サインオン情報を必要とします。
   - プログラムがワークステーション上で実行している場合、 IBM Toolbox for Java プログラムはユーザーにユーザー ID とパスワードを要求するプロンプトを出します。 注 : システム名がコマンド行パラメーターとして指定されていない場合、 AS400 オブジェクトもシステム名を要求するプロンプトを出します。
   - プログラムが iSeries の JVM 上で実行している場合、 Java プログラムを実行しているユーザーのユーザー ID とパスワードが使用されます。 この場合、ユーザーはシステム名を指定しませんが、 システム名は、プログラムが実行しているシステムの名前にデフォルト指定されます。

[前の部 | 次の部 | 簡単なプログラミング例]

別個のウィンドウで詳細説明を参照するためには、ブラウザーの JavaScript サポートが使用可能になっている必要があります。 ブラウザーの JavaScript が使用可能になっていない場合あるいは JavaScript をサポートしない場合には、 注釈 (Note) のテキスト・リンクをクリックするか、 ページの最後までスクロールして説明を参照してください。

例: メッセージ待ち行列を使用する (3 部の 3)

以下のプログラム例を参考にしてプログラムを作成してください。 例には、コード内のかぎとなる行についての詳細説明が含まれています。 以下の方法で、詳細説明を表示することができます。

- 詳細説明を表示するには、ポップアップ・ウィンドウの  イメージをクリックします。
- テキスト・リンクをクリックして、例の最後にある詳細説明を参照します。

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// Example using the Message Queue function of the IBM Toolbox for Java
//
// This source is an example of IBM Toolbox for Java "Message Queue".
//
///////////////////////////////////////////////////////////////////////////

package examples;


import java.io.*;
import java.util.*;
import com.ibm.as400.access.*;

public class displayMessages extends Object
{

   public static void main(String[] parameters)
   {
      displayMessages me = new displayMessages();

      me.Main(parameters);

      System.exit(0);
   }


   void displayMessage()
   {
   }


   void Main(String[] parms)
```

```
    {
        try
        {
            AS400 system = new AS400();

            if (parms.length > 0)
               system.setSystemName(parms[0]);

            MessageQueue queue = new MessageQueue(system, MessageQueue.CURRENT); Note 1


                    Enumeration e = queue.getMessages(); Note 2

                    while (e.hasMoreElements())
                    {

                        QueuedMessage message = (QueuedMessage) e.nextElement(); Note 3
                        System.out.println(message.getText()); Note 4
                     }
                }
                catch (Exception e)
                {
                   e.printStackTrace();
                }
            }
}
```

1. このプログラムの目的は、サーバー・メッセージ待ち行列上のメッセージを表示することです。 IBM Toolbox for Java の MessageQueue オブジェクトは、この作業のために使用されます。 メッセージ待ち行列オブジェクトが構成されるときのパラメーターは、 AS400 オブジェクトとメッセージ待ち行列名です。 AS400 オブジェクトは、どのサーバーにリソースが含まれているかを示し、 メッセージ待ち行列名は、サーバー上のどのメッセージ待ち行列であるかを示します。 この場合、メッセージ待ち行列オブジェクトに、サインオンしたユーザーの待ち行列にアクセスするように通知する定数を使用します。
2. メッセージ待ち行列オブジェクトは、サーバーからのメッセージのリストを取得します。 サーバーへの接続は、この時点で行われます。
3. リストからメッセージを除去します。 メッセージは IBM Toolbox for Java プログラムの QueuedMessage オブジェクト内にあります。
4. メッセージのテキストを印刷します。

[前の部 | 簡単なプログラミング例]

別個のウィンドウで詳細説明を参照するためには、ブラウザーの JavaScript サポートが使用可能になっている必要があります。 ブラウザーの JavaScript が使用可能になっていない場合あるいは JavaScript をサポートしない場合には、 注釈 (Note) のテキスト・リンクをクリックするか、 ページの最後までスクロールして説明を参照してください。

[次の部 | 簡単なプログラミング例]

例: レコード・レベルのアクセスを使用する (2 部の 1)

以下のプログラム例を参考にしてプログラムを作成してください。 例には、コード内のかぎとなる行についての詳細説明が含まれています。 以下の方法で、詳細説明を表示することができます。

- 詳細説明を表示するには、ポップアップ・ウィンドウの  イメージをクリックします。
- テキスト・リンクをクリックして、例の最後にある詳細説明を参照します。

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
//////////////////////////////////////////////////////////////////////////////
//
// Record level access example.  This program will prompt the user
// for the name of the server and the file to display.  The file must exist
// and should contain records.  Each record in the file will be displayed
// to System.out.
//
// Calling syntax: java RLSequentialAccessExample
//
// This source is an example of IBM Toolbox for Java "RecordLevelAccess"
//
//////////////////////////////////////////////////////////////////////////////

import java.io.*;
import java.util.*;
import com.ibm.as400.access.*;

public class RLSequentialAccessExample
{
   public static void main(String[] parameters)
   {
      BufferedReader inputStream = new BufferedReader(new InputStreamReader(System.in),1);

      String systemName = "";
      String library = "";
      String file = "";
      String member = "";

      System.out.println();
      try
      {
        System.out.print("System name: ");
        systemName = inputStream.readLine();

        System.out.print("Library in which the file exists: ");
        library = inputStream.readLine();

        System.out.print("File name: ");
        file = inputStream.readLine();

        System.out.print("Member name (press enter for first member): ");
        member = inputStream.readLine();
        if (member.equals(""))
```

```
        {
          member = "*FIRST";
        }

        System.out.println();
      }
      catch (Exception e)
      {
        System.out.println("Error obtaining user input.");
        e.printStackTrace();
        System.exit(0);
      }


      AS400 system = new AS400(systemName); Note 1
      try
      {
        system.connectService(AS400.RECORDACCESS);
      }
      catch(Exception e)
      {
        System.out.println("Unable to connect for record level access.");
        System.out.println("Check the programmer's guide setup file for special instructions regarding
record level access");
        e.printStackTrace();
        System.exit(0);
      }


      QSYSObjectPathName filePathName = new QSYSObjectPathName(library, file, member, "MBR"); Note 2


      SequentialFile theFile = new SequentialFile(system, filePathName.getPath()); Note 3


      AS400FileRecordDescription recordDescription = new AS400FileRecordDescription(system,
filePathName.getPath());
      try
      {
        RecordFormat[] format = recordDescription.retrieveRecordFormat(); Note 4


        theFile.setRecordFormat(format[0]); Note 5


        theFile.open(AS400File.READ_ONLY, 100, AS400File.COMMIT_LOCK_LEVEL_NONE); Note 6

        System.out.println("Displaying file " + library.toUpperCase() + "/" + file.toUpperCase() + "("
+ theFile.getMemberName().trim() + "):");

        Record record = theFile.readNext(); Note 7
        while (record != null)
        {
          System.out.println(record);
          record = theFile.readNext();
        }
```

```
        System.out.println();


        theFile.close(); Note 8


        system.disconnectService(AS400.RECORDACCESS); Note 9
      }
      catch (Exception e)
      {
        System.out.println("Error occurred attempting to display the file.");
        e.printStackTrace();

        try
        {
          // Close the file
          theFile.close();
        }
        catch(Exception x)
        {
        }

        system.disconnectService(AS400.RECORDACCESS);
        System.exit(0);
      }

      // Make sure that the application ends; see readme for details
      System.exit(0);
    }
  }
```

1. このコード行は、AS400 オブジェクトを作成し、 レコード・レベルのアクセス・サービスに接続します。
2. この行は、 表示されるオブジェクトの統合ファイル・システム・パス名の形式を取得する QSYSObjectPathName オブジェクトを作成します。
3. このステートメントは、接続されたサーバー上の既存の順次ファイルを表す オブジェクトを作成します。 この順次ファイルが表示されるファイルです。
4. これらの行は、ファイルのレコード様式を検索します。
5. この行は、ファイルのレコード様式を設定します。
6. この行は、選択したファイルを読み取りのためにオープンします。 可能な場合には、一度に 100 個のレコードを読み取ります。
7. このコード行は、各レコードを順番に読み取ります。
8. この行は、ファイルをクローズします。
9. この行は、レコード・レベルのアクセス・サービスから切断します。

[次の部 | 簡単なプログラミング例]

別個のウィンドウで詳細説明を参照するためには、ブラウザーの JavaScript サポートが使用可能になっている必要があります。 ブラウザーの JavaScript が使用可能になっていない場合あるいは JavaScript をサポートしない場合には、 注釈 (Note) のテキスト・リンクをクリックするか、 ページの最後までスクロールして説明を参照してください。

[前の部 | 簡単なプログラミング例]

例: レコード・レベルのアクセスを使用する (2 部の 2)

以下のプログラム例を参考にしてプログラムを作成してください。 例には、コード内のかぎとなる行についての詳細説明が含まれています。 以下の方法で、詳細説明を表示することができます。

- 詳細説明を表示するには、ポップアップ・ウィンドウの  イメージをクリックします。
- テキスト・リンクをクリックして、例の最後にある詳細説明を参照します。

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
////////////////////////////////////////////////////////////////////////////
//
// Record level access example.
//
// Calling syntax: java RLACreateExample
//
////////////////////////////////////////////////////////////////////////////

import java.io.*;
import java.util.*;
import com.ibm.as400.access.*;

public class RLACreateExample
{
  public static void main(String[] args)
  {
    AS400 system = new AS400(args[0]);
    String filePathName = "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE/MBR1.MBR"; Note 1

    try
    {
      SequentialFile theFile = new SequentialFile(system, filePathName);

      // Begin Note Two
      CharacterFieldDescription lastNameField = new CharacterFieldDescription(new AS400Text(20),
"LNAME");
      CharacterFieldDescription firstNameField = new CharacterFieldDescription(new AS400Text(20),
"FNAME");
      BinaryFieldDescription yearsOld = new BinaryFieldDescription(new AS400Bin4(), "AGE");

      RecordFormat fileFormat = new RecordFormat("RF");
      fileFormat.addFieldDescription(lastNameField);
      fileFormat.addFieldDescription(firstNameField);
      fileFormat.addFieldDescription(yearsOld);

      theFile.create(fileFormat, "A file of names and ages"); Note 2
      // End Note Two
```

```
      theFile.open(AS400File.READ_WRITE, 1, AS400File.COMMIT_LOCK_LEVEL_NONE);

      // Begin Note Three
      Record newData = fileFormat.getNewRecord();
      newData.setField("LNAME", "Doe");
      newData.setField("FNAME", "John");
      newData.setField("AGE", new Integer(63));

      theFile.write(newData); Note 3
      // End Note Three

      theFile.close();
    }
    catch(Exception e)
    {
      System.out.println("An error has occurred: ");
      e.printStackTrace();
    }

    system.disconnectService(AS400.RECORDACCESS);

    System.exit(0);
  }
}
```

1. 前の行の (args[0]) および MYFILE.FILE は、 例の残りの部分を実行するための前提条件となるコード部分です。 このプログラムは、ライブラリー MYLIB がサーバー上に存在し、 ユーザーがそれにアクセスしていることを前提としています。
2. "Begin Note Two" および "End Note Two" というラベルの付いた Java コメント内のテキストは、 レコード様式を既存ファイルから取得する代わりに、それを自分で作成する方法を示しています。 このブロックの最後の行は、サーバー上にファイルを作成します。
3. "Begin Note Three" および "End Note Three" というラベルの付いた Java コメント内のテキストは、 レコードを作成してそれをファイルに書き込む方法を示しています。

[前の部 | 簡単なプログラミング例]

別個のウィンドウで詳細説明を参照するためには、ブラウザーの JavaScript サポートが使用可能になっている必要があります。 ブラウザーの JavaScript が使用可能になっていない場合あるいは JavaScript をサポートしない場合には、 注釈 (Note) のテキスト・リンクをクリックするか、 ページの最後までスクロールして説明を参照してください。

[次の部 | 簡単なプログラミング例]

例: JDBC クラスを使用して、テーブルの作成と記入を行う (2 部の 1)

以下のプログラム例を参考にしてプログラムを作成してください。 例には、コード内のかぎとなる行についての詳細説明が含まれています。 以下の方法で、詳細説明を表示することができます。

- 詳細説明を表示するには、ポップアップ・ウィンドウの  イメージをクリックします。
- テキスト・リンクをクリックして、例の最後にある詳細説明を参照します。

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// JDBCPopulate example.  This program uses the IBM Toolbox for Java JDBC driver
// to create and populate a table.
//
// Command syntax:
//    JDBCPopulate system collectionName tableName
//
// For example,
//    JDBCPopulate MySystem MyLibrary MyTable
//
// This source is an example of IBM Toolbox for Java JDBC driver.
//
///////////////////////////////////////////////////////////////////////////

import java.sql.*;

public class JDBCPopulate
{

    private static final String words[]
        = { "One",      "Two",      "Three",    "Four",     "Five",
            "Six",      "Seven",    "Eight",    "Nine",     "Ten",
            "Eleven",   "Twelve",   "Thirteen", "Fourteen", "Fifteen",
            "Sixteen",  "Seventeen","Eighteen", "Nineteen", "Twenty" };



    public static void main (String[] parameters)
    {

        if (parameters.length != 3) {
            System.out.println("");
            System.out.println("Usage:");
            System.out.println("");
            System.out.println("   JDBCPopulate system collectionName tableName");
            System.out.println("");
```

```
        System.out.println("");
        System.out.println("For example:");
        System.out.println("");
        System.out.println("");
        System.out.println("   JDBCPopulate MySystem MyLibrary MyTable");
        System.out.println("");
        return;
    }

    String system           = parameters[0];
    String collectionName   = parameters[1];
    String tableName        = parameters[2];

    Connection connection   = null;

    try {


        DriverManager.registerDriver(new com.ibm.as400.access.AS400JDBCDriver()); Note 1


        connection = DriverManager.getConnection ("jdbc:as400://"
            + system + "/" + collectionName); Note 2


        try {
            Statement dropTable = connection.createStatement ();
            dropTable.executeUpdate ("DROP TABLE " + tableName); Note 3
        }
        catch (SQLException e) {
        }


        Statement createTable = connection.createStatement ();
        createTable.executeUpdate ("CREATE TABLE " + tableName
            + " (I INTEGER, WORD VARCHAR(20), SQUARE INTEGER, "
            + " SQUAREROOT DOUBLE)"); Note 4


        PreparedStatement insert = connection.prepareStatement ("INSERT INTO "
            + tableName + " (I, WORD, SQUARE, SQUAREROOT) "
            + " VALUES (?, ?, ?, ?)"); Note 5


        for (int i = 1; i <= words.length; ++i) {
            insert.setInt (1, i);
            insert.setString (2, words[i-1]);
            insert.setInt (3, i*i);
            insert.setDouble (4, Math.sqrt(i));
            insert.executeUpdate (); Note 6
        }

        System.out.println ("Table " + collectionName + "." + tableName
            + " has been populated.");
```

```
        }

        catch (Exception e) {
            System.out.println ();
            System.out.println ("ERROR: " + e.getMessage());
        }

        finally {

            try {
                if (connection != null)
                    connection.close (); Note 7
            }
            catch (SQLException e) {
                // Ignore.
            }
        }

        System.exit (0);
    }



}
```

1. この行は、IBM Toolbox for Java JDBC ドライバーをロードします。 JDBC ドライバーは、作業しているデータベースと JDBC との間を仲介するために必要となります。
2. このステートメントはデータベースに接続します。 ユーザー ID とパスワードの入力を求めるプロンプトが出されます。 デフォルトのスキーマが提供されるので、SQL ステートメント内のテーブル名を修飾する必要はありません。
3. これらの行は、テーブルを削除します (すでに存在している場合)。
4. これらの行はテーブルを作成します。
5. この行は、行をテーブルに挿入するステートメントを準備します。 このステートメントは数回実行されるので、 PreparedStatement およびパラメーター・マーカーを使用する必要があります。
6. このコード・ブロックは、ユーザーに代わってテーブルへの行の挿入を行います。 ループが実行されるたびに、行がテーブルに挿入されます。
7. テーブルが作成されて、行が挿入されると、 このステートメントがデータベースへの接続を閉じます。

[次の部 | 簡単なプログラミング例]

別個のウィンドウで詳細説明を参照するためには、ブラウザーの JavaScript サポートが使用可能になっている必要があります。 ブラウザーの JavaScript が使用可能になっていない場合あるいは JavaScript をサポートしない場合には、 注釈 (Note) のテキスト・リンクをクリックするか、 ページの最後までスクロールして説明を参照してください。

[前の部 | 簡単なプログラミング例]

例: JDBC クラスを使用して、テーブルの作成と記入を行う (2 部の 2)

以下のプログラム例を参考にしてプログラムを作成してください。 例には、コード内のかぎとなる行についての詳細説明が含まれています。 以下の方法で、詳細説明を表示することができます。

- 詳細説明を表示するには、ポップアップ・ウィンドウの  イメージをクリックします。
- テキスト・リンクをクリックして、例の最後にある詳細説明を参照します。

注: 法律上の重要な情報に関しては、 コードの特記事項情報をお読みください。

```
////////////////////////////////////////////////////////////////////////////
//
// JDBCQuery example.  This program uses the IBM Toolbox for Java JDBC driver to
// query a table and output its contents.
//
// Command syntax:
//    JDBCQuery system collectionName tableName
//
// For example,
//    JDBCQuery MySystem qiws qcustcdt
//
// This source is an example of IBM Toolbox for Java JDBC driver.
//
////////////////////////////////////////////////////////////////////////////

import java.sql.*;

public class JDBCQuery
{



    // Format a string so that it has the specified width.
    private static String format (String s, int width)
    {
        String formattedString;

        // The string is shorter than specified width,
        // so we need to pad with blanks.
        if (s.length() < width) {
            StringBuffer buffer = new StringBuffer (s);
            for (int i = s.length(); i < width; ++i)
                buffer.append (" ");
            formattedString = buffer.toString();
        }

        // Otherwise, we need to truncate the string.
```

```
    else
        formattedString = s.substring (0, width);

    return formattedString;
}



public static void main (String[] parameters)
{
    // Check the input parameters.
    if (parameters.length != 3) {
        System.out.println("");
        System.out.println("Usage:");
        System.out.println("");
        System.out.println("   JDBCQuery system collectionName tableName");
        System.out.println("");
        System.out.println("");
        System.out.println("For example:");
        System.out.println("");
        System.out.println("");
        System.out.println("   JDBCQuery mySystem qiws qcustcdt");
        System.out.println("");
        return;
    }

    String system           = parameters[0];
    String collectionName   = parameters[1];
    String tableName        = parameters[2];

    Connection connection   = null;

    try {


        DriverManager.registerDriver(new com.ibm.as400.access.AS400JDBCDriver()); Note 1

        // Get a connection to the database.  Since we do not
        // provide a user id or password, a prompt will appear.
        connection = DriverManager.getConnection ("jdbc:as400://" + system);
        DatabaseMetaData dmd = connection.getMetaData (); Note 2

        // Execute the query.
        Statement select = connection.createStatement ();
        ResultSet rs = select.executeQuery ("SELECT * FROM "
            + collectionName + dmd.getCatalogSeparator() + tableName); Note 3

        // Get information about the result set.  Set the column
        // width to whichever is longer: the length of the label
        // or the length of the data.
        ResultSetMetaData rsmd = rs.getMetaData ();
        int columnCount = rsmd.getColumnCount (); Note 4
        String[] columnLabels = new String[columnCount];
        int[] columnWidths = new int[columnCount];
```

```
    for (int i = 1; i <= columnCount; ++i) {
        columnLabels[i-1] = rsmd.getColumnLabel (i);
        columnWidths[i-1] = Math.max (columnLabels[i-1].length(),
            rsmd.getColumnDisplaySize (i)); Note 5
    }

    // Output the column headings.
    for (int i = 1; i <= columnCount; ++i) {
        System.out.print (format (rsmd.getColumnLabel(i), columnWidths[i-1]));
        System.out.print (" ");
    }
    System.out.println ();

    // Output a dashed line.
    StringBuffer dashedLine;
    for (int i = 1; i <= columnCount; ++i) {
        for (int j = 1; j <= columnWidths[i-1]; ++j)
            System.out.print ("-");
        System.out.print (" ");
    }
    System.out.println ();

    // Iterate throught the rows in the result set and output
    // the columns for each row.
    while (rs.next ()) {
        for (int i = 1; i <= columnCount; ++i) {
            String value = rs.getString (i);
            if (rs.wasNull ())
                value = "<null>"; Note 6
            System.out.print (format (value, columnWidths[i-1]));
            System.out.print (" ");
        }
        System.out.println ();
    }

}

catch (Exception e) {
    System.out.println ();
    System.out.println ("ERROR: " + e.getMessage());
}

finally {

    // Clean up.
    try {
        if (connection != null)
            connection.close ();
    }
    catch (SQLException e) {
        // Ignore.
    }
}

System.exit (0);
```

```
    }


}
```

1. この行は、IBM Toolbox for Java JDBC ドライバーをロードします。 JDBC ドライバーは、作業しているデータベースと JDBC との間を仲介します。
2. この行は、データベースの多くの特性を記述するオブジェクトである、 接続のメタ・データを検索します。
3. このステートメントは、指定されたテーブルに対する QUERY を実行します。
4. これらの行は、テーブルに関する情報を検索します。
5. これらの行は、ラベルの長さとデータの長さのうちの長いほうに列の幅を設定します。
6. このコード・ブロックは、 テーブル内のすべての行を繰り返し、各行の各列の内容を表示します。

[前の部 | 簡単なプログラミング例]

別個のウィンドウで詳細説明を参照するためには、ブラウザーの JavaScript サポートが使用可能になっている必要があります。 ブラウザーの JavaScript が使用可能になっていない場合あるいは JavaScript をサポートしない場合には、 注釈 (Note) のテキスト・リンクをクリックするか、 ページの最後までスクロールして説明を参照してください。

[簡単なプログラミング例]

例: サーバー・ジョブのリストを GUI で表示する

以下のプログラム例を参考にしてプログラムを作成してください。 例には、コード内のかぎとなる行についての詳細説明が含まれています。 以下の方法で、詳細説明を表示することができます。

- 詳細説明を表示するには、ポップアップ・ウィンドウの  イメージをクリックします。
- テキスト・リンクをクリックして、例の最後にある詳細説明を参照します。

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// Example using the IBM Toolbox for Java's vaccess
// class, VJobList.
//
// This source is an example of IBM Toolbox for Java "Job List".
//
///////////////////////////////////////////////////////////////////////////

package examples; Note 1

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*; Note 2

import javax.swing.*; Note 3
import java.awt.*;
import java.awt.event.*;

public class GUIExample
{

  public static void main(String[] parameters) Note 4
  {
     GUIExample example = new GUIExample(parameters);

  }

  public GUIExample(String[] parameters)
```

```
{
   try Note 5
   {
        // Create an AS400 object.
   //The system name was passed as the first command line argument.
        AS400 system = new AS400 (parameters[0]); Note 6

        VJobList jobList = new VJobList (system); Note 7

   // Create a frame.
   JFrame frame = new JFrame ("Job List Example"); Note 8

   // Create an error dialog adapter.  This will display any errors to the user.
   ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (frame); Note 9

   // Create an explorer pane to present the job list.
   AS400ExplorerPane explorerPane = new AS400ExplorerPane (jobList); Note 10

   explorerPane.addErrorListener (errorHandler); Note 11

   // Use load to load the information from the system.
   explorerPane.load(); Note 12

   // When the frame closes, exit the program.
   frame.addWindowListener (new WindowAdapter () Note 13
   {
         public void windowClosing (WindowEvent event)
         {
            System.exit(0);
         }
   } );

   // Layout the frame with the explorer pane.
   frame.getContentPane().setLayout(new BorderLayout() );
   frame.getContentPane().add("Center", explorerPane); Note 14

   frame.pack();
   frame.show(); Note 15
   }

   catch (Exception e)
   {
        e.printStackTrace(); Note 16
```

```
    System.exit(0); Note 17
    }
    }
}
```

1. このクラスは、例パッケージ内にあります。 Java は Java クラス・ファイル間の名前の競合を避けるためにパッケージを使用します。
2. この行は、vaccess パッケージ内のすべての IBM Toolbox for Java クラスをこのプログラムが使用できるようにします。 vaccess パッケージ内のクラスには、共通接頭部として com.ibm.as400.vaccess が付いています。 import ステートメントを使用することにより、 プログラムはパッケージに名前を付けたものではなく名前を呼び出します。 たとえば、com.ibm.as400.AS400ExplorerPane ではなく、AS400ExplorerPane を使用することにより、 AS400ExplorerPane クラスを参照することができます。
3. この行は、 Swing パッケージ内のすべての Java Foundation Classes (JFC) をこのプログラムで使用できるようにします。 IBM Toolbox for Java の Vaccess (GUI) クラスを使用する Java プログラムには、 Sun Microsystems, Inc. の JDK 1.1.2 と Java Swing 1.0.3 が必要です。 Swing は Sun 社の JFC 1.1 と共に入手できます。
4. このクラスには、メイン・メソッドがあるので、アプリケーションとして実行できます。 プログラムを起動するには、"java examples.GUIExample serverName" を実行します。 (serverName はご使用のサーバーの名前です。) これが作動するためには、jt400.zip または jt400.jar がクラスパス内になければなりません。
5. IBM Toolbox for Java コードは、 ユーザーのプログラムがキャッチしなければならない例外を出します。
6. AS400 クラスは IBM Toolbox for Java によって使用されます。 このクラスはサインオン情報の管理、 ソケット接続の作成と保守、およびデータの送受信を行います。 この例では、プログラムは AS400 オブジェクトにサーバー名を渡します。
7. VJobList クラスは、IBM Toolbox for Java が、 Vaccess (GUI) コンポーネント内で表示できるサーバー・ジョブのリストを表すために使用されます。 リストが常駐するサーバーを指定するには、AS400 オブジェクトを使用することに注意してください。
8. この行は、ジョブ・リストを表示するために使用される フレームまたは最上位ウィンドウを構成します。
9. ErrorDialogAdapter は、 アプリケーションでエラー・イベントが生じたときにはいつでもダイアログ・ウィンドウを自動的に表示するために作成される、 IBM Toolbox for Java グラフィカル・ユーザー・インターフェース (GUI) コンポーネントです。
10. この行は、サーバー・リソース内のオブジェクトの階層を表すグラフィカル・ユーザー・インターフェース (GUI) である AS400ExplorerPane を作成します。 AS400ExplorerPane では、左側に VJobList をルートとするツリーが示され、 右側にリソースの詳細が示されます。 これはペインをデフォルトの状態に初期化するだけに過ぎず、 VJobList の内容をペインにロードするわけではありません。
11. この行は、VJobList グラフィカル・ユーザー・インターフェース (GUI) コンポーネントでのリスナーとして、 ステップ 9 で作成したエラー・ハンドラーを追加します。

12. この行は、JobList の内容を ExplorerPane にロードします。 このメソッドは、サーバーと通信して、そこから情報をロードするために、明示的に呼び出す必要があります。 これは、サーバーとの通信が生じるときに、アプリケーション制御を提供します。 これによって、以下の事柄を行えます。
    - ペインをフレームに追加する前に、コンテンツをロードすることができます。 すべての情報がロードされるまでは、この例のようにフレームは表示しません。
    - ペインをフレームに追加し、そのフレームを表示した後で、内容をロードすることもできます。 「待機カーソル」のあるフレームが表示され、 ロードが完了した情報からフレームの中に入れられていきます。
13. この行は、フレームをクローズしたときにアプリケーションが終了するように、 ウィンドウ・リスナーを追加します。
14. この行は、制御フレームの中央に ジョブ・リストのグラフィカル・ユーザー・インターフェース GUI コンポーネントを追加します。
15. この行は、ウィンドウをユーザーに見えるようにするための表示メソッドを呼び出します。
16. IBM Toolbox for Java 例外は、 テキストがワークステーションの言語で表示されるように変換されます。 たとえば、このプログラムはエラーを処理する際に、例外のテキストを表示します。
17. IBM Toolbox for Java は、IBM Toolbox for Java アクティビティーを実行するための スレッドを作成します。 終了時にプログラムが System.exit(0) を発行しないと、 プログラムは正常に終了しないことがあります。 たとえば、Windows 95 DOS プロンプトからプログラムを実行している場合、 上記の行がなければ、プログラムの終了時にコマンド・プロンプトに戻りません。

[簡単なプログラミング例]

例: プログラミングに関するヒント

このセクションでは、接続の管理についてのトピックを扱った資料に記載されているコーディング例をリストします。

接続の管理

- 例: CommandCall オブジェクトを使用して iSeries への接続を 1 つ作成する
- 例: CommandCall オブジェクトを使用して iSeries への接続を 2 つ作成する
- 例: 同じ AS400 を使用して、 CommandCall オブジェクトと IFSFileInputStream オブジェクトを作成する
- 例: AS400ConnectionPool を使用して iSeries サーバーに事前接続する
- 例: AS400ConnectionPool を使用して iSeries サーバー上の特定の装置に事前接続してから、接続を再利用する

接続の開始および終了

- 例: Java プログラムを iSeries サーバーに事前接続する方法
- 例: Java プログラムを iSeries サーバーから切断する方法
- 例: disconnectService() と run() を使用して、 Java プログラムを iSeries サーバーから切断してから、再接続する方法
- 例: Java プログラムを iSeries サーバーから切断し、再接続できないようにする方法

例外

- 例: 例外を使用する

エラー・イベント

- 例: エラー・イベントを処理する
- 例: エラー・リスナーを定義する
- 例: カスタマイズ・ハンドラーを使用してエラー・イベントを処理する

トレース

- 例: トレースを使用する
- 例: setTraceOn() を使用する
- 例: コンポーネント・トレースを使用する

最適化

- 例: AS400 オブジェクトを 2 つ作成する
- 例: AS400 オブジェクトを使用して 2 番目のサーバーを表す

インストールおよび更新

- 例: AS400Toolbox Installer クラスを使用する

以下の特記事項は、IBM Toolbox for Java のすべての例に適用されます。

例: ToolboxME for iSeries

このセクションでは、 ToolboxME for iSeries の資料に記載されているコーディング例をリストします。

- 例: ToolboxME for iSeries の例を作成する - JdbcDemo.java
- 例: ToolboxME for iSeries、 MIDP、および JDBC を使用する
- 例: ToolboxME for iSeries、 MIDP、および Toolbox for Java を使用する

以下の特記事項は、IBM Toolbox for Java のすべての例に適用されます。

»

例: ToolboxME for iSeries、MIDP、および JDBC を使用する

以下のソースは、ToolboxME for iSeries アプリケーションが Mobile Information Device Profile (MIDP) および JDBC を使用してデータベースにアクセスし、 オフラインの情報を保管できる 1 つの方法を例示しています。

この例は、不動産業者が現在売り出している土地を表示し入札できるようにする方法を示します。 業者は、Tier0 デバイスを使用し、 iSeries サーバー・データベースに保管されている土地の情報にアクセスします。

作業プログラムとして構築されると、以下のコード例はその目的で作成されたデータベースに接続します。

ソース・コードの作業バージョンを作成し、 必要なデータベースを作成して移植するためのソースを入手するには、 例をダウンロードする必要があります。 また、プログラム例を作成して実行する際の指示を検討することもできます。

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
////////////////////////////////////////////////////////////////////////////
//
// ToolboxME for iSeries example. This program is an example MIDlet that shows how
// you might code a JdbcMe application for the MIDP profile. Refer to the
// startApp, pauseApp, destroyApp and commandAction methods to see how it handles
// each requested transition.
//
////////////////////////////////////////////////////////////////////////////

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.sql.*;
import javax.microedition.rms.*;

import com.ibm.as400.micro.*;


public class JdbcMidpBid extends MIDlet implements CommandListener
{
    private static int BID_PROPERTY = 0;
    private Display   display;

    private TextField urlText = new TextField("urltext",
"jdbc:as400://mySystem;user=myUid;password=myPwd;", 65, TextField.ANY);
    private TextField jdbcmeText = new TextField("jdbcmetext", "meserver=myMEServer", 40,
TextField.ANY);
    private TextField jdbcmeTraceText = new TextField("jdbcmetracetext", "0", 10, TextField.ANY);
    private final static String GETBIDS = "No bids are available, select here to download bids";
    private List      main = new List("JdbcMe Bid Demo", Choice.IMPLICIT);
    private List      listings = null;
    private Form      aboutBox;
    private Form      bidForm;
    private Form      settingsForm;
    private int       bidRow = 0;
    private String    bidTarget = null;
    private String    bidTargetKey = null;
    private TextField bidText = new TextField("bidtext", "", 10, TextField.NUMERIC);
    private Form      errorForm = null;

    private Command exitCommand   = new Command("Exit", Command.SCREEN, 0);
    private Command backCommand   = new Command("Back", Command.SCREEN, 0);
    private Command cancelCommand = new Command("Cancel", Command.SCREEN, 0);
```

```
private Command goCommand     = new Command("Go", Command.SCREEN, 1);
private Displayable   onErrorGoBackTo = null;

/*
 * Construct a new JdbcMidpBid.
 */
public JdbcMidpBid()
{
    display = Display.getDisplay(this);
}

/**
 * Show the main screen
 */
public void startApp()
{
    main.append("Show Bids", null);
    main.append("Get New Bids", null);
    main.append("Settings", null);
    main.append("About", null);
    main.addCommand(exitCommand);
    main.setCommandListener(this);

    display.setCurrent(main);
}

public void commandAction(Command c, Displayable s)
{
    // All exitCommand processing is the same.
    if (c == exitCommand)
    {
        destroyApp(false);
        notifyDestroyed();
        return;
    }
    if (s instanceof List)
    {
        List     current = (List)s;

        // An action occurred on the main page
        if (current == main)
        {
            int   idx = current.getSelectedIndex();
            switch (idx)
            {
            case 0:     // Show current bids
                showBids();
                break;
            case 1:     // Get New Bids
                getNewBids();
                break;
            case 2:     // Settings
                doSettings();
                break;
            case 3:     // About
                aboutBox();
                break;
            default :
                break;
            }
```

```
            return;
        } // current == main

        // An action occurred on the listings page
        if (current == listings)
        {
            if (c == backCommand)
            {
                display.setCurrent(main);
                return;
            }
            if (c == List.SELECT_COMMAND)
            {
                int idx = listings.getSelectedIndex();
                String stext = listings.getString(idx);
                if (stext.equals(GETBIDS))
                {
                    getNewBids();
                    return;
                }
                int   commaIdx = stext.indexOf(',');
                bidTargetKey = stext.substring(0, commaIdx);
                bidTarget = stext.substring(commaIdx+1) + "/n";
                // Also keep track of which offline result set row
                // This is. It happens to be the same as the index
                // in the list.
                bidRow = idx;

                bidOnProperty();
            }
        } // current == listings
        return;
    } // instanceof List
    if (s instanceof Form)
    {
        Form     current = (Form)s;
        if (current == errorForm)
        {
            if (c == backCommand)
                display.setCurrent(onErrorGoBackTo);

            return;
        } // errorForm
        if (current == settingsForm)
        {
            if (c == backCommand)
            {
                // Done with settings.
                display.setCurrent(main);
                settingsForm = null;
                return;
            }
        } // settingsForm
        if (current == aboutBox)
        {
            if (c == backCommand)
            {
                // Done with about box.
                display.setCurrent(main);
```

```
                aboutBox = null;
                return;
            }
        }
        if (current == bidForm)
        {
            if (c == cancelCommand)
            {
                display.setCurrent(listings);
                bidForm = null;
                return;
            }
            if (c == goCommand)
            {
                submitBid();
                if (display.getCurrent() != bidForm)
                {
                    // If we're no longer positioned at the
                    // bidForm, we will get rid of it.
                    bidForm = null;
                }
                return;
            }
            return;
        } // current == bidForm
    } // instanceof Form
}

public void aboutBox()
{
    aboutBox = new Form("aboutbox");
    aboutBox.setTitle("About");
    aboutBox.append(new StringItem("", "Midp RealEstate example for JdbcMe "));
    aboutBox.addCommand(backCommand);
    aboutBox.setCommandListener(this);
    display.setCurrent(aboutBox);
}

/**
 *  The settings form.
 */
public void doSettings()
{
    settingsForm = new Form("settingsform");
    settingsForm.setTitle("Settings");
    settingsForm.append(new StringItem("", "DB URL"));
    settingsForm.append(urlText);
    settingsForm.append(new StringItem("", "JdbcMe server"));
    settingsForm.append(jdbcmeText);
    settingsForm.append(new StringItem("", "Trace"));

    settingsForm.addCommand(backCommand);
    settingsForm.setCommandListener(this);
    display.setCurrent(settingsForm);
}


/**
 * Show the bid screen for the bid target
 * that we selected.
```

```
     */
    public void bidOnProperty()
    {
        StringItem item = new StringItem("", bidTarget);

        bidText = new TextField("bidtext", "", 10, TextField.NUMERIC);
        bidText.setString("");

        bidForm = new Form("bidform");
        bidForm.setTitle("Submit a bid for:");
        BID_PROPERTY = 0;
        bidForm.append(item);
        bidForm.append(new StringItem("", "Your bid:"));
        bidForm.append(bidText);
        bidForm.addCommand(cancelCommand);
        bidForm.addCommand(goCommand);
        bidForm.setCommandListener(this);
        display.setCurrent(bidForm);
    }

    /**
     * Update the listings card with the
     * current list of bids that we're interested in.
     */
    public void getNewBids()
    {
        // Reset the old listing
        listings = null;
        listings = new List("JdbcMe Bids", Choice.IMPLICIT);
        java.sql.Connection     conn = null;
        Statement               stmt = null;
        try
        {
            conn = DriverManager.getConnection(urlText.getString() + ";" + jdbcmeText.getString());

            stmt = conn.createStatement();

            // Since we don't want the prepared statement to persist,
            // a normal statement is really better in this environemnt.
            String sql = "select mls, address, currentbid from qjdbcme.realestate where currentbid <>
0";

            boolean results =((JdbcMeStatement)stmt).executeToOfflineData(sql,"JdbcMidpBidListings",0,
0);
            if (results)
            {
                setupListingsFromOfflineData();
            }
            else
            {
                listings.append("No bids found", null);
                listings.addCommand(backCommand);
                listings.setCommandListener(this);
            }
        }
        catch (Exception e)
        {
            // Currently no valid listings retrieved, so lets
            // reset it to empty.
```

```
        listings = new List("JdbcMe Bids", Choice.IMPLICIT);
        listings.append(GETBIDS, null);
        listings.addCommand(backCommand);
        listings.setCommandListener(this);

        // Return to main after showing the error.
        showError(main, e);
        return;
    }
    finally
    {
        if (conn != null)
        {
            try
            {
                conn.close();
            }
            catch (Exception e)
            {
            }
        }
        conn = null;
        stmt = null;
    }
    showBids();
}

public void setupListingsFromOfflineData()
{
    // Skip the first four rows in the record store
    // (eyecatcher, version, num columns, sql column
    // types)
    // and each subsequent row in the record store is
    // a single column. Our query returns 3 columns which
    // we'll return concatenated as a single string.
    ResultSet         rs = null;
    listings.addCommand(backCommand);
    listings.setCommandListener(this);
    try
    {
        int                     i = 5;
        int                     max = 0;
        StringBuffer   buf  = new StringBuffer(20);

        // Creator and dbtype unused in MIDP
        rs = new JdbcMeOfflineResultSet("JdbcMidpBidListings", 0, 0);
        if (rs == null)
        {
            // New listings...
            listings = new List("JdbcMe Bids", Choice.IMPLICIT);
            listings.append(GETBIDS, null);
            listings.addCommand(backCommand);
            listings.setCommandListener(this);
            return;
        }

        i = 0;
        String      s = null;
        while (rs.next())
        {
```

```
            ++i;

            s = rs.getString(1);
            buf.append(s);

            buf.append(",");
            s = rs.getString(2);
            buf.append(s);

            buf.append(", $");
            s = rs.getString(3);
            buf.append(s);

            listings.append(buf.toString(), null);
            buf.setLength(0);
        }

        if (i == 0)
        {
            listings.append("No bids found", null);
            return;
        }
    }
    catch (Exception e)
    {
        // Currently no valid listings retrieved, so lets
        // reset it to empty.
        listings = new List("JdbcMe Bids", Choice.IMPLICIT);
        listings.append(GETBIDS, null);
        listings.addCommand(backCommand);
        listings.setCommandListener(this);

        // Return to main after showing the error.
        showError(main, e);
        return;
    }
    finally
    {
        if (rs != null)
        {
            try
            {
                rs.close();
            }
            catch (Exception e)
            {
            }
            rs = null;
        }
        System.gc();
    }
}

/**
 * Update the listings card with the
 * current list of bids that we're interested in.
 */
public void submitBid()
{
```

```
java.sql.Connection     conn = null;
Statement               stmt = null;
try
{
    conn = DriverManager.getConnection(urlText.getString() + ";" + jdbcmeText.getString());

    stmt = conn.createStatement();

    // Since we don't want the prepared statement to persist,
    // a normal statement is really better in this environemnt.
    StringBuffer   buf = new StringBuffer(100);
    buf.append("Update QJdbcMe.RealEstate Set CurrentBid = ");
    buf.append(bidText.getString());
    buf.append(" Where MLS = '");
    buf.append(bidTargetKey);
    buf.append("' and CurrentBid < ");
    buf.append(bidText.getString());
    String      sql = buf.toString();

    int   updated = stmt.executeUpdate(sql);
    if (updated == 1)
    {
        // BID Accepted.
        String oldS = listings.getString(bidRow);
        int   commaIdx = bidTarget.indexOf(',');
        String bidAddr = bidTarget.substring(0, commaIdx);

        String newS = bidTargetKey + "," + bidAddr + ", $" + bidText.getString();

        ResultSet         rs = null;
        try
        {
            // Creator and dbtype unused in MIDP
            rs = new JdbcMeOfflineResultSet("JdbcMidpBidListings", 0, 0);
            rs.absolute(bidRow+1);
            rs.updateString(3, bidText.getString());
            rs.close();
        }
        catch (Exception e)
        {
            if (rs != null)
                rs.close();
        }

        // Also update our live list of that result set.
        listings.set(bidRow, newS, null);
        display.setCurrent(listings);
        conn.commit();
    }
    else
    {
        conn.rollback();
        throw new SQLException("Failed to bid, someone beat you to it");
    }
}
catch (SQLException e)
{
    // Return to the bid form after showing the error.
    showError(bidForm, e);
    return;
```

```
    }
    finally
    {
        if (conn != null)
        {
            try
            {
                conn.close();
            }
            catch (Exception e)
            {
            }
        }
        conn = null;
        stmt = null;
    }

    // Exit without exception, then show the current bids
    showBids();
}

/**
 * Show an error condition.
 */
public void showError(Displayable d, Exception e)
{
    String s = e.toString();

    onErrorGoBackTo = d;
    errorForm = new Form("Error");
    errorForm.setTitle("SQL Error");
    errorForm.append(new StringItem("", s));
    errorForm.addCommand(backCommand);
    errorForm.setCommandListener(this);
    display.setCurrent(errorForm);
}

/**
 * Show the current bids.
 */
public void showBids()
{
    if (listings == null)
    {
        // If we have no current listings, lets set
        // them up.
        listings = new List("JdbcMe Bids", Choice.IMPLICIT);
        setupListingsFromOfflineData();
    }
    display.setCurrent(listings);
}

/**
 * Time to pause, free any space we don't need right now.
 */
public void pauseApp()
{
    display.setCurrent(null);
}
```

```
    /**
     * Destroy must cleanup everything.
     */
    public void destroyApp(boolean unconditional)
    {
    }
}
```

»

例: ToolboxME for iSeries、MIDP、および Toolbox for Java を使用する

以下のソースは、ToolboxME for iSeries アプリケーションが Mobile Information Device Profile (MIDP) および IBM Toolbox for Java を使用して、iSeries サーバーのデータおよびサービスにアクセスできる 1 つの方法を例示しています。

この例は、IBM Toolbox for Java 2 Micro Edition サポートに組み込まれている各機能を示しています。 このアプリケーションは、Tier0 デバイスがこれらの機能を使用できる 多くの方法のうちのいくつかを例示する、各種のページや画面を特徴としています。

作業プログラムとして構築されると、 以下のコード例はプログラム呼び出しマークアップ言語 (PCML) ファイルを使用し、 iSeries サーバー上のコマンドを実行します。

ソース・コードの作業バージョンを作成し、 サーバー・コマンドを実行するのに必要な PCML ソースを入手するには、 例をダウンロードする必要があります。 また、プログラム例を作成して実行する際の指示を検討することもできます。

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
////////////////////////////////////////////////////////////////////////////
//
// ToolboxME for iSeries example. This program is an example that shows how
// ToolboxME for iSeries can use PCML to access data and services on an
// iSeries server.
//
// This application requires that the qsyrusri.pcml file is present in the
// CLASSPATH of the MEServer.
//
////////////////////////////////////////////////////////////////////////////

import java.io.*;
import java.sql.*;
import java.util.Hashtable;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.rms.*;

import com.ibm.as400.micro.*;


public class ToolboxMidpDemo extends MIDlet implements CommandListener
{
    private Display   display_;

    // A ToolboxME system object.
    private AS400 system_;

    private List      main_ = new List("Toolbox MIDP Demo", Choice.IMPLICIT);

    // Create a form for each component.
    private Form      signonForm_;
    private Form      cmdcallForm_;
    private Form      pgmcallForm_;
    private Form      dataqueueForm_;
    private Form      aboutForm_;
```

```
    // Visable Text for each component.
    static final String SIGN_ON       = "SignOn";
    static final String COMMAND_CALL  = "CommandCall";
    static final String PROGRAM_CALL  = "ProgramCall";
    static final String DATA_QUEUE    = "DataQueue";
    static final String ABOUT         = "About";

    static final String NOT_SIGNED_ON = "Not signed on.";
    static final String DQ_READ       = "Read";
    static final String DQ_WRITE      = "Write";

    // A ticker to display the signon status.
    private Ticker    ticker_ = new Ticker(NOT_SIGNED_ON);

    // Commands that can be performed.
    private static final Command actionExit_   = new Command("Exit", Command.SCREEN, 0);
    private static final Command actionBack_   = new Command("Back", Command.SCREEN, 0);
    private static final Command actionGo_     = new Command("Go", Command.SCREEN, 1);
    private static final Command actionClear_  = new Command("Clear", Command.SCREEN, 1);
    private static final Command actionRun_    = new Command("Run", Command.SCREEN, 1);
    private static final Command actionSignon_ = new Command(SIGN_ON, Command.SCREEN, 1);
    private static final Command actionSignoff_= new Command("SignOff", Command.SCREEN, 1);

    private Displayable   onErrorGoBackTo_;  // the form to return to when done displaying the error
form

    // TextFields for the SignOn form.
    private TextField signonSystemText_ = new TextField("System", "rchasdm3", 20, TextField.ANY);
    private TextField signonUidText_ = new TextField("UserId", "JAVA", 10, TextField.ANY);
    private TextField signonPwdText_ = new TextField("Password", "JTEAM1", 10, TextField.PASSWORD);
// TBD temporary
    private TextField signonServerText_ = new TextField("MEServer", "localhost", 10, TextField.ANY);
    private StringItem signonStatusText_ = new StringItem("Status", NOT_SIGNED_ON);

    // TextFields for the CommandCall form.
    private TextField cmdText_ = new TextField("Command", "CRTLIB FRED", 256, TextField.ANY);  // TBD:
max size; TBD: TextBox???
    private StringItem cmdMsgText_ = new StringItem("Messages", null);
    private StringItem cmdStatusText_ = new StringItem("Status", null);

    // TextFields for the ProgramCall form.
    private StringItem pgmMsgDescription_ = new StringItem("Messages", null);
    private StringItem pgmMsgText_ = new StringItem("Messages", null);

    // TextFields for the DataQueue form.
    private TextField dqInputText_ = new TextField("Data to write", "Hi there", 30, TextField.ANY);
    private StringItem dqOutputText_ = new StringItem("DQ contents", null);
    private ChoiceGroup dqReadOrWrite_ = new ChoiceGroup("Action", Choice.EXCLUSIVE, new String[] {
DQ_WRITE, DQ_READ}, null);
    private StringItem dqStatusText_ = new StringItem("Status", null);


    /**
     * Creates a new ToolboxMidpDemo.
     **/
    public ToolboxMidpDemo()
    {
        display_ = Display.getDisplay(this);
        // Note: The KVM-based demo used TabbedPane for the main panel.  MIDP has no similar class, so
```

```
we use a List instead.
    }

    /**
     * Show the main screen.
     * Implements abstract method of class Midlet.
     **/
    protected void startApp()
    {
        main_.append(SIGN_ON, null);
        main_.append(COMMAND_CALL, null);
        main_.append(PROGRAM_CALL, null);
        main_.append(DATA_QUEUE, null);
        main_.append(ABOUT, null);

        main_.addCommand(actionExit_);
        main_.setCommandListener(this);

        display_.setCurrent(main_);
    }

    // Implements method of interface CommandListener.
    public void commandAction(Command action, Displayable dsp)
    {
        // All 'exit' and 'back' processing is the same.
        if (action == actionExit_)
        {
            destroyApp(false);

            notifyDestroyed();
        }
        else if (action == actionBack_)
        {
            // Return to main menu.
            display_.setCurrent(main_);
        }
        else if (dsp instanceof List)
        {
            List current = (List)dsp;

            // An action occurred on the main page
            if (current == main_)
            {
                int   idx = current.getSelectedIndex();

                switch (idx)
                {
                case 0:     // SignOn
                    showSignonForm();
                    break;
                case 1:     // CommandCall
                    showCmdForm();
                    break;
                case 2:     // ProgramCall
                    showPgmForm();
                    break;
                case 3:     // DataQueue
                    showDqForm();
                    break;
                case 4:     // About
```

```
                    showAboutForm();
                    break;
                default:    // None of the above
                    feedback("Internal error: Unhandled selected index in main: " + idx,
AlertType.ERROR);
                    break;
                }
            } // current == main
            else
                feedback("Internal error: The Displayable object is a List but is not main_.",
AlertType.ERROR);
        } // instanceof List
        else if (dsp instanceof Form)
        {
            Form current = (Form)dsp;

            if (current == signonForm_)
            {
                if (action == actionSignon_)
                {
                    // Create a ToolboxME system object.
                    system_ = new AS400(signonSystemText_.getString(), signonUidText_.getString(),
signonPwdText_.getString(), signonServerText_.getString());

                    try
                    {
                        // Connect to the iSeries.
                        system_.connect();

                        // Set the signon status text.
                        signonStatusText_.setText("Signed on.");

                        // Display a confirmation dialog that the user is signed on.
                        feedback("Successfully signed on.", AlertType.INFO, main_);

                        // Replace the SignOn button with SignOff.
                        signonForm_.removeCommand(actionSignon_);
                        signonForm_.addCommand(actionSignoff_);

                        // Update the ticker.
                        ticker_.setString("... Signed on to '" + signonSystemText_.getString() + "' as
'" + signonUidText_.getString() + "' via '" + signonServerText_.getString() + "' ... ");
                    }
                    catch (Exception e)
                    {
                        e.printStackTrace();

                        // Set the signon status text.
                        signonStatusText_.setText(NOT_SIGNED_ON);

                        feedback("Signon failed.  " + e.getMessage(), AlertType.ERROR);
                    }
                }
                else if (action == actionSignoff_)
                {
                    if (system_ == null)
                        feedback("Internal error: System is null.", AlertType.ERROR);
                    else
                    {
```

```
            try
            {
                // Disconnect from the iSeries.
                system_.disconnect();
                system_ = null;

                // Set the signon status text.
                signonStatusText_.setText(NOT_SIGNED_ON);

                // Display a confirmation dialog that the user is no longer signed on.
                feedback("Successfully signed off.", AlertType.INFO, main_);

                // Replace the SignOff button with SignOn.
                signonForm_.removeCommand(actionSignoff_);
                signonForm_.addCommand(actionSignon_);

                // Update the ticker.
                ticker_.setString(NOT_SIGNED_ON);
            }
            catch (Exception e)
            {
                feedback(e.toString(), AlertType.ERROR);

                e.printStackTrace();

                signonStatusText_.setText("Error.");

                feedback("Error during signoff.", AlertType.ERROR);
            }
        }
    }
    else  // None of the above.
    {
        feedback("Internal error: Action is not recognized.", AlertType.INFO);
    }
} // signonForm_
else if (current == cmdcallForm_)
{
    if (action == actionRun_)
    {
        // If the user has not signed on, display an alert.
        if (system_ == null)
        {
            feedback(NOT_SIGNED_ON, AlertType.ERROR);
            return;
        }

        // Get the command the user entered in the wireless device.
        String cmdString = cmdText_.getString();

        // If the command was not specified, display an alert.
        if (cmdString == null || cmdString.length() == 0)
            feedback("Specify command.", AlertType.ERROR);
        else
        {
            try
            {
                // Run the command.
                String[] messages = CommandCall.run(system_, cmdString);
```

```
                    StringBuffer status = new StringBuffer("Command completed with ");

                    // Check to see if their are any messages.
                    if (messages.length == 0)
                    {
                        status.append("no returned messages.");

                        cmdMsgText_.setText(null);

                        cmdStatusText_.setText("Command completed successfully.");
                    }
                    else
                    {
                        if (messages.length == 1)
                            status.append("1 returned message.");
                        else
                            status.append(messages.length + " returned messages.");

                        // If there are messages, display only the first message.
                        cmdMsgText_.setText(messages[0]);

                        cmdStatusText_.setText(status.toString());
                    }

                    repaint();
                }
                catch (Exception e)
                {
                    feedback(e.toString(), AlertType.ERROR);

                    e.printStackTrace();

                    feedback("Error when running command.", AlertType.ERROR);
                }
            }
        }
        else if (action == actionClear_)
        {
            // Clear the command text and messages.
            cmdText_.setString("");

            cmdMsgText_.setText(null);

            cmdStatusText_.setText(null);

            repaint();
        }
        else  // None of the above.
        {
            feedback("Internal error: Action is not recognized.", AlertType.INFO);
        }
    } // cmdcallForm_
    else if (current == pgmcallForm_)
    {
        if (action == actionRun_)
        {
            // If the user is not signed on before doing a program call, display an alert.
            if (system_ == null)
            {
```

```
                    feedback(NOT_SIGNED_ON, AlertType.ERROR);
                    return;
                }

                pgmMsgText_.setText(null);

                // See the PCML example in the Toolbox programmer's guide.
                String pcmlName = "qsyrusri.pcml"; // The PCML file we want to use.
                String apiName = "qsyrusri";

                // Create a hashtable that contains the input parameters for the program call.
                Hashtable parmsToSet = new Hashtable(2);
                parmsToSet.put("qsyrusri.receiverLength", "2048");
                parmsToSet.put("qsyrusri.profileName", signonUidText_.getString().toUpperCase());

                // Create a string array that contains the output parameters to retrieve.
                String[] parmsToGet = { "qsyrusri.receiver.userProfile",
                    "qsyrusri.receiver.previousSignonDate",
                    "qsyrusri.receiver.previousSignonTime",
                    "qsyrusri.receiver.daysUntilPasswordExpires"};

                // A string array containing the descriptions of the parameters to display.
                String[] displayParm = { "Profile", "Last signon Date", "Last signon Time",
"Password Expired (days)"};

                try
                {
                    // Run the program.
                    String[] valuesToGet = ProgramCall.run(system_, pcmlName, apiName, parmsToSet,
parmsToGet);

                    // Create a StringBuffer and add each of the parameters we retreived.
                    StringBuffer txt = new StringBuffer();
                    txt.append(displayParm[0] + ": " + valuesToGet[0] + "/n");

                    char[] c = valuesToGet[1].toCharArray();
                    txt.append(displayParm[1] + ": " + c[3]+c[4]+"/"+c[5]+c[6]+"/"+c[1]+c[2]  +
"/n");

                    char[] d = valuesToGet[2].toCharArray();
                    txt.append(displayParm[2] + ": " + d[0]+d[1]+":"+d[2]+d[3] + "/n");
                    txt.append(displayParm[3] + ": " + valuesToGet[3] + "/n");

                    // Set the displayable text of the program call results.
                    pgmMsgText_.setText(txt.toString());

                    StringBuffer status = new StringBuffer("Program completed with ");

                    if (valuesToGet.length == 0)
                    {
                        status.append("no returned values.");

                        feedback(status.toString(), AlertType.INFO);
                    }
                    else
                    {
                        if (valuesToGet.length == 1)
                            status.append("1 returned value.");
                        else
                            status.append(valuesToGet.length + " returned values.");
```

```
                        feedback(status.toString(), AlertType.INFO);
                    }
                }
                catch (Exception e)
                {
                    feedback(e.toString(), AlertType.ERROR);

                    e.printStackTrace();

                    feedback("Error when running program.", AlertType.ERROR);
                }
            }
            else if (action == actionClear_)
            {
                // Clear the program call results.
                pgmMsgText_.setText(null);

                repaint();
            }
        }  // pgmcallForm_
        else if (current == dataqueueForm_)  // DataQueue
        {
            if (action == actionGo_)
            {
                // If the user has not signed on before performing Data Queue actions, display an
alert.
                if (system_ == null)
                {
                    feedback(NOT_SIGNED_ON, AlertType.ERROR);

                    return;
                }

                // Create a library to create the data queue in.
                try
                {
                    CommandCall.run(system_, "CRTLIB FRED");
                }
                catch (Exception e)
                {
                }

                // Run a command to create a data queue.
                try
                {
                    CommandCall.run(system_, "CRTDTAQ FRED/MYDTAQ MAXLEN(2000)");
                }
                catch (Exception e)
                {
                    feedback("Error when creating data queue.  " + e.getMessage(),
AlertType.WARNING);
                }

                try
                {
                    // See which action was selected (Read or Write).
                    if
(dqReadOrWrite_.getString(dqReadOrWrite_.getSelectedIndex()).equals(DQ_WRITE))
```

```
                    {
                        // Write
                        dqOutputText_.setText(null);

                        // Get the text from the wireless device input to be written to the data
queue.
                        if (dqInputText_.getString().length() == 0)
                            dqStatusText_.setText("No data specified.");
                        else
                        {
                            // Write to the data queue.
                            DataQueue.write(system_, "/QSYS.LIB/FRED.LIB/MYDTAQ.DTAQ",
dqInputText_.getString().getBytes() );

                            dqInputText_.setString(null);

                            // Display the status.
                            dqStatusText_.setText("The 'write' operation completed.");
                        }
                    }
                    else  // Read
                    {
                        // Read from the data queue.
                        byte[] b = DataQueue.readBytes(system_, "/QSYS.LIB/FRED.LIB/MYDTAQ.DTAQ");

                        // Determine if the data queue contained entries or not and display the
appropriate message.
                        if (b == null)
                        {
                            dqStatusText_.setText("No dataqueue entries are available.");

                            dqOutputText_.setText(null);
                        }
                        else if (b.length == 0)
                        {
                            dqStatusText_.setText("Dataqueue entry has no data.");

                            dqOutputText_.setText(null);
                        }
                        else
                        {
                            dqStatusText_.setText("The 'read' operation completed.");

                            dqOutputText_.setText(new String(b));
                        }
                    }

                    repaint();
                }
                catch (Exception e)
                {
                    e.printStackTrace();

                    feedback(e.toString(), AlertType.ERROR);

                    feedback("Error when running command.  " + e.getMessage(), AlertType.ERROR);
                }
            }  // actionGo_
            else if (action == actionClear_)
            {
```

```
                    // Clear the data queue form.
                    dqInputText_.setString("");

                    dqOutputText_.setText(null);

                    dqReadOrWrite_.setSelectedFlags(new boolean[] { true, false});

                    dqStatusText_.setText(null);

                    repaint();
                }
                else  // None of the above.
                {
                    feedback("Internal error: Action is not recognized.", AlertType.INFO);
                }
            } // dataqueueForm_
            else if (current == aboutForm_)  // "About".
            {
                // Should never reach here, since the only button is "Back".
            }  // None of the above.
            else
                feedback("Internal error: Form is not recognized.", AlertType.ERROR);
        } // instanceof Form
        else
            feedback("Internal error: Displayable object not recognized.", AlertType.ERROR);
    }



    /**
     *  Displays the "About" form.
     **/
    private void showAboutForm()
    {
        // If the about form is null, create and append it.
        if (aboutForm_ == null)
        {
            aboutForm_ = new Form(ABOUT);
            aboutForm_.append(new StringItem(null, "This is a MIDP example application that uses the
Toolbox Micro Edition (ToolboxME)."));

            aboutForm_.addCommand(actionBack_);
            aboutForm_.setCommandListener(this);
        }

        display_.setCurrent(aboutForm_);
    }



    /**
     *  Displays the "SignOn" form.
     **/
    private void showSignonForm()
    {
        // Create the signon form.
        if (signonForm_ == null)
        {
            signonForm_ = new Form(SIGN_ON);
```

```
            signonForm_.append(signonSystemText_);
            signonForm_.append(signonUidText_);
            signonForm_.append(signonPwdText_);
            signonForm_.append(signonServerText_);
            signonForm_.append(signonStatusText_);
            signonForm_.addCommand(actionBack_);
            signonForm_.addCommand(actionSignon_);
            signonForm_.setCommandListener(this);
            signonForm_.setTicker(ticker_);
        }

        display_.setCurrent(signonForm_);
    }



    /**
     *  Displays the "CommandCall" form.
     **/
    private void showCmdForm()
    {
        // Create the command call form.
        if (cmdcallForm_ == null)
        {
            cmdcallForm_ = new Form(COMMAND_CALL);
            cmdcallForm_.append(cmdText_);
            cmdcallForm_.append(cmdMsgText_);
            cmdcallForm_.append(cmdStatusText_);
            cmdcallForm_.addCommand(actionBack_);
            cmdcallForm_.addCommand(actionClear_);
            cmdcallForm_.addCommand(actionRun_);
            cmdcallForm_.setCommandListener(this);
            cmdcallForm_.setTicker(ticker_);
        }

        display_.setCurrent(cmdcallForm_);
    }



    /**
     *  Displays the "ProgramCall" form.
     **/
    private void showPgmForm()
    {
        // Create the program call form.
        if (pgmcallForm_ == null)
        {
            pgmcallForm_ = new Form(PROGRAM_CALL);
            pgmcallForm_.append(new StringItem(null, "This calls the Retrieve User Information
(QSYRUSRI) API, and returns information about the current user profile."));
            pgmcallForm_.append(pgmMsgText_);
            pgmcallForm_.addCommand(actionBack_);
            pgmcallForm_.addCommand(actionClear_);
            pgmcallForm_.addCommand(actionRun_);
            pgmcallForm_.setCommandListener(this);
            pgmcallForm_.setTicker(ticker_);
        }

        display_.setCurrent(pgmcallForm_);
```

```
    }


    /**
     *  Displays the "DataQueue" form.
     **/
    private void showDqForm()
    {
        // Create the data queue form.
        if (dataqueueForm_ == null)
        {
            dataqueueForm_ = new Form(DATA_QUEUE);
            dataqueueForm_.append(dqInputText_);
            dataqueueForm_.append(dqOutputText_);
            dataqueueForm_.append(dqReadOrWrite_);
            dataqueueForm_.append(dqStatusText_);
            dataqueueForm_.addCommand(actionBack_);
            dataqueueForm_.addCommand(actionClear_);
            dataqueueForm_.addCommand(actionGo_);
            dataqueueForm_.setCommandListener(this);
            dataqueueForm_.setTicker(ticker_);
        }

        display_.setCurrent(dataqueueForm_);
    }


    private void feedback(String text, AlertType type)
    {
        feedback(text, type, display_.getCurrent());
    }

    /**
     *  This method is used to create a dialog and display feedback information using an Alert to the
user.
     **/
    private void feedback(String text, AlertType type, Displayable returnToForm)
    {
        System.err.flush();
        System.out.flush();

        Alert alert = new Alert("Alert", text, null, type);

        if (type == AlertType.INFO)
            alert.setTimeout(3000);  // milliseconds
        else
            alert.setTimeout(Alert.FOREVER);  // Require user to dismiss the alert.

        display_.setCurrent(alert, returnToForm);
    }


    // Force a repaint of the current form.
    private void repaint()
    {
        Alert alert = new Alert("Updating display ...", null, null, AlertType.INFO);
        alert.setTimeout(1000);  // milliseconds

        display_.setCurrent(alert, display_.getCurrent());
```

```
    }



    /**
     * Time to pause, free any space we don't need right now.
     * Implements abstract method of class Midlet.
     **/
    protected void pauseApp()
    {
        display_.setCurrent(null);
    }



    /**
     * Destroy must cleanup everything.
     * Implements abstract method of class Midlet.
     **/
    protected void destroyApp(boolean unconditional)
    {
        // Disconnect from the iSeries if the Midlet is being destroyed or exited.
        if (system_ != null)
        {
            try
            {
                system_.disconnect();
            }
            catch (Exception e)
            {
            }
        }
    }
}
```
«

例: ユーティリティー・クラス

このセクションでは、ユーティリティー・クラスの資料に記載されているコーディング例をリストします。

IBM Toolbox Installer

- 例: AS400ToolboxInstaller クラスを使用する
- 例: AS400ToolboxInstaller を使用して IBM Toolbox for Java をインストールする
- 例: コマンド行から ACCESS パッケージをインストールする
- 例: コマンド行から Graphical Toolbox クラスを処理する

JarMaker

- 例: AS400.class とそれに従属するすべてのクラスを jt400.jar から抽出する
- 例: jt400.jar を 300K ごとのファイルの集合に分割する
- 例: JAR ファイルから使用しないファイルを除去する
- 例: -ccsid パラメーターを指定して変換テーブルを省略し、 400K バイト小さい JAR ファイルを作成する

»CommandPrompter

- 例: CommandPrompter を使用してコマンド用のプロンプトを出して実行する «

以下の特記事項は、IBM Toolbox for Java のすべての例に適用されます。

ここに含まれるすべてのプログラムは、現存するままの状態で提供され、いかなる保証条件も適用されません。第三者の権利の不侵害、商品性、特定目的適合性に関する黙示の保証の適用も一切ありません。

例: AS400ToolboxInstaller を使用して IBM Toolbox for Java のインストールと更新を行う

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
////////////////////////////////////////////////////////////////////////////
//
// Install/Update example.  This program uses the AS400ToolboxInstaller class
// to install and update the IBM Toolbox for Java package on the workstation.
//
// The program checks the target path for the IBM Toolbox for Java package.
// If the package is not found, it installs the package on the workstation.
// If the package is found it checks the source path for updates.  If
// updates are found they are copied to the workstation.
//
// Command syntax:
//    checkToolbox source target
//
// Where
//    source = location of the source files.  This name is in URL format.
//    target = location of the target files.
//
////////////////////////////////////////////////////////////////////////////

import java.io.*;
import java.util.*;
import java.net.*;
import utilities.*;

public class checkToolbox extends Object
{
   public static void main(String[] parameters)
   {
      System.out.println( " " );



      // Continue with the install/update only if both source and target
      // names were specified.

      if (parameters.length >= 2)
      {

         // The first parameter is the source for the files, the second is the target.

         String sourcePath = parameters[0];
         String targetPath = parameters[1];

         boolean installIt = false;
         boolean updateIt  = false;



         // Created a reader to get input from the user.

         BufferedReader inputStream = new BufferedReader(new InputStreamReader(System.in),1);

         try
         {
            // Point at the source package.  AS400ToolboxInstaller uses the URL
```

```
// class to access the files.

URL sourceURL  = new URL(sourcePath);



// See if the package is installed on the client.  If not, ask the user
// if install should be performed at this time.

if (AS400ToolboxInstaller.isInstalled("ACCESS", targetPath) == false)
{
   System.out.print("IBM Toolbox for Java is not installed.  Install now (Y/N):");

   String userInput = inputStream.readLine();

   if ((userInput.charAt(0) == 'y')   ||
       (userInput.charAt(0) == 'Y'))
      installIt = true;
}



// The package is installed.  See if updates need to be copied from the
// server.  If the target is out of data ask the user if update should
// be performed at this time.

else
{
   if (AS400ToolboxInstaller.isUpdateNeeded("ACCESS", targetPath, sourceURL) == true)
   {
      System.out.print("IBM Toolbox for Java is out of date.  Install fixes (Y/N):");

      String userInput = inputStream.readLine();

      if ((userInput.charAt(0) == 'y')   ||
          (userInput.charAt(0) == 'Y'))
         updateIt = true;
   }
   else
      System.out.println("Target directory is current, no update needed.");
}



// If the package needs to be installed or updated.

if (updateIt || installIt)
{
   // Copy the files from the server to the target.

   AS400ToolboxInstaller.install("ACCESS", targetPath, sourceURL);




   // Report that the install/update was successful.

   System.out.println(" ");

   if (installIt)
```

```
            System.out.println("Install successful!");
         else
            System.out.println("Update Successful!");



         // Tell the user what must be added to the CLASSPATH environment
         // variable.

         Vector classpathAdditions = AS400ToolboxInstaller.getClasspathAdditions();

         if (classpathAdditions.size() > 0)
         {
            System.out.println("");
            System.out.println("Add the following to the CLASSPATH environment variable:");

            for (int i = 0; i < classpathAdditions.size(); i++)
            {
               System.out.print("  ");

               System.out.println((String)classpathAdditions.elementAt(i));
            }
         }



         // Tell the user what can be revmoed from the CLASSPATH environment
         // variable.

         Vector classpathRemovals = AS400ToolboxInstaller.getClasspathRemovals();

         if (classpathRemovals.size() > 0)
         {
            System.out.println("");

            System.out.println("Remove the following from the CLASSPATH environment variable:");

            for (int i = 0; i < classpathRemovals.size(); i++)
            {
               System.out.print("  ");
               System.out.println((String)classpathRemovals.elementAt(i));
            }
         }
      }
   }

   catch (Exception e)
   {
      // If any of the above operations failed say the operation failed
      // and output the exception.

      System.out.println("Install/Update failed");
      System.out.println(e);
   }
}



// Display help text when parameters are incorrect.
```

```
      else
      {
         System.out.println("");
         System.out.println("");
         System.out.println("");
         System.out.println("Parameters are not correct.  Command syntax is:");
         System.out.println("");
         System.out.println("  checkToolbox sourcePath targetPath");
         System.out.println("");
         System.out.println("Where");
         System.out.println("");
         System.out.println("   sourcePath = source for IBM Toolbox for Java files");
         System.out.println("   targetPath = target for IBM Toolbox for Java files");
         System.out.println("");
         System.out.println("For example:");
         System.out.println("");
         System.out.println("   checkToolbox http://mySystem/QIBM/ProdData/HTTP/Public/jt400/
d://jt400");
         System.out.println("");
         System.out.println("");
      }

      System.exit(0);
   }
}
```

»

例: CommandPrompter を使用する

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// CommandPrompter example. This program uses CommandPrompter, CommandCall, and
// AS400Message to prompt for a command, run the command, and display any
// messages returned if the command does not run.
//
// Command syntax:
//    Prompter commandString
//
///////////////////////////////////////////////////////////////////////////

import com.ibm.as400.ui.util.CommandPrompter;
import com.ibm.as400.access.AS400;
import com.ibm.as400.access.AS400Message;
import com.ibm.as400.access.CommandCall;
import javax.swing.JFrame;
import java.awt.FlowLayout;
public class Prompter
{
public static void main ( String args[] ) throws Exception
  {
  JFrame frame = new JFrame();
  frame.getContentPane().setLayout(new FlowLayout());
  AS400 system = new AS400("mySystem", "myUserId", "myPasswd");
  String cmdName = args[0];

  // Launch the CommandPrompter
  CommandPrompter cp = new CommandPrompter(frame, system, cmdName);
  if (cp.showDialog() == CommandPrompter.OK)
    {
    String cmdString = cp.getCommandString();
    System.out.println("Command string: " + cmdString);

    // Run the command that was built in the prompter.
    CommandCall cmd = new CommandCall(system, cmdString);
    if (!cmd.run())
      {
      AS400Message[] msgList = cmd.getMessageList();
      for (int i = 0; i < msgList.length; ++i)
        {
          System.out.println(msgList[i].getText());
        }
```

```
        }
      }
    System.exit(0);
    }
}
```

例: Vaccess クラス

このセクションでは、vaccess クラスの資料に記載されているコーディング例をリストします。

AS400Panes

- 例: AS400DetailsPane を作成して、 systemAS400DetailsPane に定義されているユーザーのリストを表示する
- 例: 詳細ペインの内容をロードしてから、フレームに追加する
- 例: AS400ListPane を使用してユーザーのリストを表示する
- 例: AS400DetailsPane を使用してコマンド呼び出しから戻されたメッセージを表示する
- 例: AS400TreePane を使用してディレクトリーのツリー・ビューを表示する
- 例: AS400ExplorerPane を使用してさまざまな印刷リソースを表示する

コマンド呼び出し

- 例: CommandCallButton を作成する
- 例: ActionCompletedListener を追加して、コマンドが生成するすべての iSeries メッセージを処理する
- 例: CommandCallMenuItem を使用する

データ待ち行列

- 例: DataQueueDocument を作成する
- 例: DataQueueDocument を使用する

エラー・イベント

- 例: エラー・イベントを処理する
- 例: エラー・リスナーを定義する
- 例: カスタマイズ・ハンドラーを使用してエラー・イベントを処理する

JDBC

- 例: JDBC ドライバーを使用して、表の作成と記入を行う
- 例: JDBC ドライバーを使用して、表の照会およびその内容の出力を行う
- 例: AS400JDBCDataSourcePane を作成する

ジョブ

- 例: VJobList を作成し、 AS400ExplorerPane にそのリストを表示する
- 例: explorer pane にジョブのリストを表示する

プログラム呼び出し

- 例: ProgramCallMenuItem を作成する
- 例: プログラムが生成したすべての iSeries メッセージを処理する
- 例: 2 つのパラメーターを追加する
- 例: アプリケーションで ProgramCallButton を使用する

レコード・レベルでのアクセス

- 例: RecordListTablePane オブジェクトを作成して、キーよりも小か等しいレコードをすべて表示する

SpooledFileViewer

- 例: スプール・ファイル・ビューアーを作成して、 iSeries 上にすでに作成されているスプール・ファイルを表示する

システム値

- 例: AS400Explorer ペインを使用してシステム値 GUI を作成する

ユーザーおよびグループ

- 例: AS400DetailsPane に VUserList を作成する
- 例: AS400ListPane を使用して、選択対象のユーザーのリストを作成する

以下の特記事項は、IBM Toolbox for Java のすべての例に適用されます。

コードの特記事項情報

IBM は、お客様に、すべてのプログラミング・コード・サンプルを使用することができる非独占的な使用権を許諾します。お客様は、このサンプル・コードから、お客様独自の特別のニーズに合わせた類似のプログラムを作成することができます。

すべてのサンプル・コードは、例として示す目的でのみ、IBM により提供されます。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。したがって 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。

ここに含まれるすべてのプログラムは、現存するままの状態で提供され、いかなる保証条件も適用されません。第三者の権利の不侵害、商品性、特定目的適合性に関する黙示の保証の適用も一切ありません。

例: VUserList を使用する

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////
//
// VUserList example.  This program presents a list of users on
// a system in a list pane, and allows selection of one or more
// users.
//
// Command syntax:
//    VUserListExample system
//
///////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class VUserListExample
{


    private static AS400ListPane listPane;


    public static void main (String[] args)
    {
        // If a system is not specified, display help text and
        // exit.
        if (args.length != 1)
        {
            System.out.println("Usage:  VUserListExample system");
            return;
        }

        try
        {
            // Create an AS400 object.  The system name is passed
            // as the first command line argument.
            AS400 system = new AS400 (args[0]);
```

```
        // Create a VUserList.  This represents a list of users
        // displayed in the list pane.
        VUserList userList = new VUserList (system);

        // Create a frame.
        JFrame f = new JFrame ("VUserList example");

        // Create an error dialog adapter.  This displays
        // any errors to the user.
        ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

        // Create a list pane to display the user list.
        // Use load to get the information from the server.
        listPane = new AS400ListPane (userList);
        listPane.addErrorListener (errorHandler);
        listPane.load ();

        // When the frame closes, report the selected
        // users and exit.
        f.addWindowListener (new WindowAdapter () {
            public void windowClosing (WindowEvent event)
            {
                reportSelectedUsers ();
                System.exit (0);
            }
        });

        // Layout the frame with the list pane.
        f.getContentPane ().setLayout (new BorderLayout ());
        f.getContentPane ().add ("Center", listPane);
        f.pack ();
        f.show ();
    }
    catch (Exception e)
    {
       System.out.println ("Error: " + e.getMessage ());
       System.exit (0);
    }
}


private static void reportSelectedUsers ()
{
```

```
        VObject[] selectedUsers = listPane.getSelectedObjects ();

        if (selectedUsers.length == 0)
            System.out.println ("No users were selected.");
        else
        {
            System.out.println ("The selected users were:");
            for (int i = 0; i < selectedUsers.length; ++i)
                System.out.println (selectedUsers[i]);
        }
    }


}
```

例: VMessageList を使用する

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////
//
// VMessageList example.  This program presents a details
// view of messages returned from a command call.
//
// Command syntax:
//    VMessageListExample system
//
// This source is an example of IBM Toolbox for Java "VMessageList".
//
///////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class VMessageListExample
{


    public static void main (String[] args)
    {
        // If a system was not specified, then display help text and
        // exit.
        if (args.length != 1)
        {
            System.out.println("Usage:  VMessageListExample system");
            return;
        }

        try
        {
            // Create an AS400 object.  The system name was passed
            // as the first command line argument.
            AS400 system = new AS400 (args[0]);

            // Create a CommandCall object a run the command.
            CommandCall command = new CommandCall (system);
            command.run ("CRTLIB FRED");

            // Create a VMessageList object with the messages
            // returned from the command call.
            VMessageList messageList = new VMessageList (command.getMessageList ());
```

```
            // Create a frame.
            JFrame f = new JFrame ("VMessageList example");

            // Create an error dialog adapter.  This will display
            // any errors to the user.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

            // Create a details pane to display the message list.
            // Use load to load the information.
            AS400DetailsPane detailsPane = new AS400DetailsPane (messageList);
            detailsPane.addErrorListener (errorHandler);
            detailsPane.load ();

            // When the frame closes, exit.
            f.addWindowListener (new WindowAdapter () {
                public void windowClosing (WindowEvent event)
                {
                    System.exit (0);
                }
            });

            // Layout the frame with the details pane.
            f.getContentPane ().setLayout (new BorderLayout ());
            f.getContentPane ().add ("Center", detailsPane);
            f.pack ();
            f.show ();
        }
        catch (Exception e)
        {
           System.out.println ("Error: " + e.getMessage ());
           System.exit (0);
        }
    }


}
```

例: VIFSDirectory を使用する

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////
//
// VIFSDirectory example.  This program presents a tree view of
// some directories in the integrated file system.
//
// Command syntax:
//    VIFSDirectoryExample system
//
// This source is an example of IBM Toolbox for Java "VIFSDirectory".
//
///////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class VIFSDirectoryExample
{


    public static void main (String[] args)
    {
        // If a system was not specified, then display help text and
        // exit.
        if (args.length != 1)
        {
            System.out.println("Usage:  VIFSDirectoryExample system");
            return;
        }

        try
        {
            // Create an AS400 object.  The system name was passed
            // as the first command line argument.
            AS400 system = new AS400 (args[0]);

            // Create a VIFSDirectory object which represents the root
            // of the directory tree that we are going to show.
            VIFSDirectory directory = new VIFSDirectory (system, "/QIBM/ProdData");

            // Create a frame.
            JFrame f = new JFrame ("VIFSDirectory example");
```

```
            // Create an error dialog adapter.  This will display
            // any errors to the user.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

            // Create a tree pane to present the directories hierarchically.
            // Load the information from the system.
            AS400TreePane treePane = new AS400TreePane (directory);
            treePane.addErrorListener (errorHandler);
            treePane.load ();

            // When the frame closes, exit.
            f.addWindowListener (new WindowAdapter () {
                public void windowClosing (WindowEvent event)
                {
                    System.exit (0);
                }
            });

            // Layout the frame with the tree pane.
            f.getContentPane ().setLayout (new BorderLayout ());
            f.getContentPane ().add ("Center", treePane);
            f.pack ();
            f.show ();
        }
        catch (Exception e)
        {
           System.out.println ("Error: " + e.getMessage ());
           System.exit (0);
        }
    }


}
```

例: VPrinters を使用する

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////
//
// VPrinters example.  This program presents various network
// print resources with an explorer pane.
//
// Command syntax:
//    VPrintersExample system
//
// This source is an example of IBM Toolbox for Java "VPrinters".
//
///////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class VPrintersExample
{


    public static void main (String[] args)
    {
        // If a system was not specified, then display help text and
        // exit.
        if (args.length != 1)
        {
            System.out.println("Usage:  VPrintersExample system");
            return;
        }

        try
        {
            // Create an AS400 object.  The system name was passed
            // as the first command line argument.
            AS400 system = new AS400 (args[0]);

            // Create a VPrinters object which represents the list
            // of printers attached to the system.
            VPrinters printers = new VPrinters (system);

            // Create a frame.
```

```
            JFrame f = new JFrame ("VPrinters example");

            // Create an error dialog adapter.  This will display
            // any errors to the user.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

            // Create an explorer pane to present the network print resources.
            // Use load to load the information from the system.
            AS400ExplorerPane explorerPane = new AS400ExplorerPane (printers);
            explorerPane.addErrorListener (errorHandler);
              explorerPane.load ();

            // When the frame closes, exit.
            f.addWindowListener (new WindowAdapter () {
                public void windowClosing (WindowEvent event)
                {
                    System.exit (0);
                }
            });

            // Layout the frame with the explorer pane.
            f.getContentPane ().setLayout (new BorderLayout ());
            f.getContentPane ().add ("Center", explorerPane);
            f.pack ();
            f.show ();
        }
        catch (Exception e)
        {
           System.out.println ("Error: " + e.getMessage ());
           System.exit (0);
        }
    }


}
```

例: CommandCallMenuItem を使用する

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////
//
// Command call menu item example.  This program demonstrates how to
// use a menu item that calls a server command.  It will display
// any messages that are returned in a dialog.
//
// Command syntax:
//    CommandCallMenuItemExample system
//
///////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class CommandCallMenuItemExample
{



    private static JFrame f;



    public static void main (String[] args)
    {
        // If a system was not specified, then display help text and
        // exit.
        if (args.length != 1)
        {
            System.out.println("Usage:  CommandCallMenuItemExample system");
            return;
        }

        try
        {
            // Create an AS400 object.  The system name was passed
            // as the first command line argument.
            AS400 system = new AS400 (args[0]);

            // Create a frame.
            f = new JFrame ("Command call menu item example"

            // Create an error dialog adapter.  This will display
            // any errors to the user.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);
```

```
    // Create a CommandCallMenuItem object to run the command.
    CommandCallMenuItem menuItem = new CommandCallMenuItem ("Clear library FRED",
        null, system, "CLRLIB FRED"
    menuItem.addErrorListener (errorHandler);

    // Add an action completed listener to display any
    // returned messages in a dialog.
    menuItem.addActionCompletedListener (new ActionCompletedListener ()
    {
        public void actionCompleted (ActionCompletedEvent event)
        {
            // Get the message list from the event source.
            CommandCallMenuItem item = (CommandCallMenuItem) event.getSource ();
            AS400Message[] messageList = item.getMessageList ();

            // Use an AS400DetailsPane to display the messages.
            VMessageList vmessageList = new VMessageList (messageList);
            AS400DetailsPane messageDetails = new AS400DetailsPane (vmessageList);
            messageDetails.load ();

            // Show the details in a dialog.
            JDialog dialog = new JDialog(f);
      dialog.getContentPane().setLayout(new BorderLayout());
      dialog.getContentPane().add("Center"messageDetails);
      dialog.pack();
      dialog.setVisible(true);
        }
    });

    // Create a menu with the item.
    JMenu menu = new JMenu ("Server Command Calls");
    menu.add (menuItem);

    JMenuBar menuBar = new JMenuBar ();
    menuBar.add (menu);

    f.getRootPane ().setJMenuBar (menuBar);

    // When the frame closes, exit.
    f.addWindowListener (new WindowAdapter () {
        public void windowClosing (WindowEvent event)
        {
            System.exit (0);
        }
    });

    // Layout the frame with the details pane.
    f.getContentPane ().setLayout (new BorderLayout ());
    f.setSize (300, 400);
    f.show ();
}
catch (Exception e)
{
```

```
            System.out.println ("Error: " + e.getMessage ());
            System.exit (0);
        }
    }


}
```

例: DataQueueDocument を使用する

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////
//
// Data queue document example.  This program demonstrates how to
// use a document that is associated with a server data queue.
//
// Command syntax:
//    DataQueueDocumentExample system read|write
//
///////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class DataQueueDocumentExample
{
    private static DataQueueDocument    dqDocument;
    private static JTextField           text;
    private static boolean              rw;

    public static void main (String[] args)
    {
        // If a system or read|write was not specified, then display
        // help text and exit.
        if (args.length != 2)
        {
            System.out.println("Usage:  DataQueueDocumentExample system read|write");
            return;
        }

        rw = args[1].equalsIgnoreCase ("read");
        String mode = rw ? "Read" : "Write";

        try
        {
            // Create two frames.
            JFrame f = new JFrame ("Data queue document example - " + mode);

            // Create an error dialog adapter.  This will display
            // any errors to the user.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

            // Create a working cursor adapter.  This will adjust
            // the cursor whenever a data queue is read or written.
            WorkingCursorAdapter cursorAdapter = new WorkingCursorAdapter (f);
```

```
// Create an AS400 object.  The system name was passed
// as the first command line argument.
AS400 system = new AS400 (args[0]);

// Create the data queue path name.
QSYSObjectPathName dqName = new QSYSObjectPathName ("QGPL",
    "JAVATALK", "DTAQ");

// Make sure the the data queue exists.
DataQueue dq = new DataQueue (system, dqName.getPath ());
try
{
    dq.create (200);
}
catch (Exception e)
{
    // Ignore exceptions.  Most likely, the data queue
    // already exists.
}

// Create a DataQueueDocument object.
dqDocument = new DataQueueDocument (system, dqName.getPath ());
dqDocument.addErrorListener (errorHandler);
dqDocument.addWorkingListener (cursorAdapter);

// Create a text field used to present the document.
text = new JTextField (dqDocument, "", 40);
text.setEditable (! rw);

// When the program runs, we need a way to control when
// the reads and writes take place.  We will let the
// use control this with a button.
Button button = new Button (mode);
button.addActionListener (new ActionListener ()
{
    public void actionPerformed (ActionEvent event)
    {
        if (rw)
            dqDocument.read ();
        else {
            dqDocument.write ();
            text.setText ("");
        }
    }
});

// When the the frame closes, exit.
f.addWindowListener (new WindowAdapter () {
    public void windowClosing (WindowEvent event)
```

```
                {
                    System.exit (0);
                }
            });

            // Layout the frame.
            f.getContentPane ().setLayout (new FlowLayout ());
            f.getContentPane ().add (text);
            f.getContentPane ().add (button);
            f.pack ();
            f.show ();
        }
        catch (Exception e)
        {
           System.out.println ("Error: " + e.getMessage ());
           System.exit (0);
        }
    }
}
```

AS400 JDBCDataSourcePane

AS400JDBCDataSourcePane クラスは、 AS400JDBCDataSource オブジェクトのプロパティー値を表します。 オプションで、AS400JDBCDataSource オブジェクトに変更を加えることができます。

AS400JDBCDataSourcePane は JComponent を拡張します。
AS400JDBCDataSourcePane を使用してデータ・ソースのプロパティーを表示するには、 データ・ソースを AS400JDBCDataSourcePane コンストラクターで指定するか、 または setDataSource() を使って AS400JDBCDataSourcePane が作成された後に設定することができます。 グラフィカル・ユーザー・インターフェース (GUI) に加えられた変更は、 applyChanges() を使用してデータ・ソース・オブジェクトに適用できます。

以下の例では、AS400JDBCDataSourcePane および OK ボタンを作成し、 それらをフレームに追加します。 GUI に加えられた変更は、OK をクリックするとデータ・ソースに適用されます。

例: AS400JDBCDataSourcePane を使用する

```
// Create a data source.
myDataSource = new AS400JDBCDataSource();

// Create a window to hold the pane and an OK button.
JFrame frame = new JFrame ("JDBC Data Source Properties");

// Create a data source pane.
dataSourcePane = new AS400JDBCDataSourcePane(myDataSource);

// Create an OK button
JButton okButton = new JButton("OK");

// Add an ActionListener to the OK button.  When OK is
// pressed, applyChanges() will be called to commit any
// changes to the data source.
okButton.addActionListener(new ActionListener()
   {
      public void actionPerformed(ActionEvent ev)
      {
         // Apply all changes made on the data source pane
         // to the data source.  If all changes are applied
```

```
            // successfully, get the data source from the pane.
            if (dataSourcePane.applyChanges())
            {
               System.out.println("ok pressed");
               myDataSource = dataSourcePane.getDataSource();
               System.out.println(myDataSource.getServerName());
            }
         }
      }
   );

   // Setup the frame to show the pane and OK button.
   frame.getContentPane ().setLayout (new BorderLayout ());
   frame.getContentPane ().add ("Center", dataSourcePane);
   frame.getContentPane ().add ("South", okButton);

   // Pack the frame.
   frame.pack ();

   //Display the pane and OK button.
   frame.show ();
```

例: VJobList を使用してジョブのリストを表示する

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////
//
// Job list example.  This program presents a list of jobs in an
// explorer pane.
//
// Command syntax:
//    VJobListExample system
//
// This source is an example of IBM Toolbox for Java "AS400ExplorerPane".
//
///////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class VJobListExample
{


    public static void main (String[] args)
    {
        // If a system was not specified, then display help text and
        // exit.
        if (args.length != 1)
        {
            System.out.println("Usage:  VJobListExample system");
            return;
        }

        try
        {
            // Create an AS400 object.  The system name was passed
            // as the first command line argument.
            AS400 system = new AS400 (args[0]);

            // Create a VJobList object which represents the list
            // of jobs named QZDASOINIT.
            VJobList jobList = new VJobList (system);
            jobList.setName ("QZDASOINIT");
```

```
            // Create a frame.
            JFrame f = new JFrame ("Job list example");

            // Create an error dialog adapter.  This will display
            // any errors to the user.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

            // Create an explorer pane to present the job list.
            // Use load to load the information from the system.
            AS400ExplorerPane explorerPane = new AS400ExplorerPane (jobList);
            explorerPane.addErrorListener (errorHandler);
            explorerPane.load ();

            // When the frame closes, exit.
            f.addWindowListener (new WindowAdapter () {
                public void windowClosing (WindowEvent event)
                {
                    System.exit (0);
                }
            });

            // Layout the frame with the explorer pane.
            f.getContentPane ().setLayout (new BorderLayout ());
            f.getContentPane ().add ("Center", explorerPane);
            f.pack ();
            f.show ();
        }
        catch (Exception e)
        {
           System.out.println ("Error: " + e.getMessage ());
           System.exit (0);
        }
    }


}
```

例: ボタンを使用してサーバー上のプログラムを呼び出す

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
///////////////////////////////////////////////////////////////////////
//
// Program call button example.  This program demonstrates how to
// use a button that calls a program on the server.  It will exchange data
// with the server program via an input and output parameter.
//
// Command syntax:
//    ProgramCallButtonExample system
//
// This source is an example of IBM Toolbox for Java "ProgramCallButton".
//
///////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class ProgramCallButtonExample
{

    private ProgramParameter   parm1, parm2, parm3, parm4, parm5;
    private JTextField         cpuField;
    private JTextField         dasdField;
    private JTextField         jobsField;


    // Create a ProgramCallButtonExample object, then call the
    // non-static version of main().  If we don't to this then
    // the class variables (parm1, parm2, ...) must be declared
    // static.  If they are static they cannot be used by the
    // action completed listener in Java 1.1.7 or 1.1.8.
    public static void main (String[] args)
    {
       ProgramCallButtonExample me = new ProgramCallButtonExample();
       me.Main(args);
    }

    public void Main (String[] args)
    {
        // If a system was not specified, then display help text and
```

```
// exit.
if (args.length != 1)
{
    System.out.println("Usage:  ProgramCallButtonExample system");
    return;
}

try
{
    // Create a frame.
    JFrame f = new JFrame ("Program call button example");

    // Create an error dialog adapter.  This will display
    // any errors to the user.
    ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

    // Create an AS400 object.  The system name was passed
    // as the first command line argument.
    AS400 system = new AS400 (args[0]);

    // Create the program path name.
    QSYSObjectPathName programName = new QSYSObjectPathName ("QSYS",
        "QWCRSSTS", "PGM");

    // Create a ProgramCallButton object.  The button
    // will have the text "Refresh" and no icon.
    ProgramCallButton button = new ProgramCallButton ("Refresh", null);
    button.setSystem (system);
    button.setProgram (programName.getPath ());
    button.addErrorListener (errorHandler);

    // The first parameter is an 64 byte output parameter.
    parm1 = new ProgramParameter (64);
    button.addParameter (parm1);

    // We use the second parameter to set the buffer size
    // of the first parameter.  We will always set this to
    // 64.  Remember that we need to convert the Java int
    // value 64 to the format used on the server.
    AS400Bin4 parm2Converter = new AS400Bin4 ();
    byte[] parm2Bytes = parm2Converter.toBytes (64);
    parm2 = new ProgramParameter (parm2Bytes);
    button.addParameter (parm2);

    // The third parameter is the status format.  We will
    // always use "SSTS0200".  This is a String value, and
    // again we need to convert it to the format used on the server.
    AS400Text parm3Converter = new AS400Text (8, system);
```

```
byte[] parm3Bytes = parm3Converter.toBytes ("SSTS0200");
parm3 = new ProgramParameter (parm3Bytes);
button.addParameter (parm3);

// The fourth parameter is the reset statistics parameter.
// We will always pass "*NO" as a 10 character String.
AS400Text parm4Converter = new AS400Text (10, system);
byte[] parm4Bytes = parm4Converter.toBytes ("*NO       ");
parm4 = new ProgramParameter (parm4Bytes);
button.addParameter (parm4);

// The fifth parameter is for error information.  It
// is an input/output parameter.  We will not use it
// for this example, but we need to set it to something,
// or else the number of parameters will not match
// what the server is expecting.
byte[] parm5Bytes = new byte[32];
parm5 = new ProgramParameter (parm5Bytes, 0);
button.addParameter (parm5);

// When the program runs, we will get a bunch of data.
// We need a way to display that data to the user.
// In this case, we will just use simple labels and text
// fields.
JLabel cpuLabel = new JLabel ("CPU Utilitization: ");
cpuField = new JTextField (10);
cpuField.setEditable (false);

JLabel dasdLabel = new JLabel ("DASD Utilitization: ");
dasdField = new JTextField (10);
dasdField.setEditable (false);

JLabel jobsLabel = new JLabel ("Number of active jobs: ");
jobsField = new JTextField (10);
jobsField.setEditable (false);

// When the frame closes, exit.
f.addWindowListener (new WindowAdapter ()
{
    public void windowClosing (WindowEvent event)
    {
        System.exit (0);
    }
});

// When the program is called, we need to process the
// information that comes back in the first parameter.
// The format of the data in this parameter was documented
```

```
// by the program we are calling.
button.addActionCompletedListener (new ActionCompletedListener ()
{
    public void actionCompleted (ActionCompletedEvent event)
    {
        try
        {
           // Get the data from the first parameter.
           // It is in the server format.
           byte[] parm1Bytes = parm1.getOutputData ();

           // Each of the pieces of data that we need
           // is an int.  We can create one converter
           // to do all of our conversions.
           AS400Bin4 parm1Converter = new AS400Bin4 ();

           // Get the CPU utilitization starting at byte 32.
           // Set this value in the corresponding text field.
           int cpu = parm1Converter.toInt (parm1Bytes, 32);
           cpuField.setText (Integer.toString (cpu / 10) + "%");

           // Get the DASD utilitization starting at byte 52.
           // Set this value in the corresponding text field.
           int dasd = parm1Converter.toInt (parm1Bytes, 52);
           dasdField.setText (Integer.toString (dasd / 10000) + "%");

           // Get the number of active jobs starting at byte 36.
           // Set this value in the corresponding text field.
           int jobs = parm1Converter.toInt (parm1Bytes, 36);
           jobsField.setText (Integer.toString (jobs));
        }
        catch (Exception e) { e.printStackTrace(); }
    }
});

// Layout the frame.
JPanel outputPanel = new JPanel ();
outputPanel.setLayout (new GridLayout (3, 2, 5, 5));
outputPanel.add (cpuLabel);
outputPanel.add (cpuField);
outputPanel.add (dasdLabel);
outputPanel.add (dasdField);
outputPanel.add (jobsLabel);
outputPanel.add (jobsField);

Panel buttonPanel = new Panel ();
buttonPanel.add (button);
```

```
            f.getContentPane ().setLayout (new BorderLayout ());
            f.getContentPane ().add ("Center", outputPanel);
            f.getContentPane ().add ("South", buttonPanel);
            f.pack ();
            f.show ();
        }
        catch (Exception e)
        {
           System.out.println ("Error: " + e.getMessage ());
           System.exit (0);
        }
    }


}
```

プログラミングに関するヒント

このセクションには、IBM Toolbox for Java を使用する上で役立つ、種々のヒントが収められています。

Java プログラムのシャットダウン
Java プログラムを正しくシャットダウンする方法がわかります。

統合ファイル・システム・パス名を使用する
プログラムでの統合ファイル・システム・パス名の使用について読みます。 このセクションでは、統合ファイル・システム・パス名、パラメーター、および特殊値を扱います。

接続の管理
ソケット接続を開始および終了するために、Enterprise JavaBean の仕様に適合させる方法を含め、 AS400 クラスの使用法を参照します。

OS/400 Java 仮想マシン (JVM) を使用する
OS/400 JVM 上での IBM Toolbox for Java クラスの実行について学びます。 このセクションでは、サーバー・リソースに最も効率的にアクセスする方法、 クラスを実行する方法、および考慮すべきサインオン要因を扱います。

»独立補助記憶域プール (IASP) に接続する
IASP への接続について読みます。 IASP とは、システム上の残りのストレージとは独立して、 オンラインにしたり、オフラインにすることができるディスク装置の集合です。 «

Toolbox for Java アクセス・クラス使用時のエラー処理
プログラム内で Toolbox for Java アクセス・クラスを使用する時、エラー処理に Toolbox for Java 例外クラスを使用することについてがわかります。

Toolbox for Java の Vaccess クラス使用時のエラー処理
プログラム内で Toolbox for Java の Vaccess クラスを使用する時、エラー処理に Toolbox for Java エラー・イベント・クラスを使用する方法を参照します。

トレース・クラスを使用する
プログラミング上の問題点を複製し、診断するのを助けるために、トレース・ポイントおよび診断メッセージをログに記録する働きをするトレース・クラスの使用について学びます。

プログラムの最適化
パフォーマンスを向上させるために、プログラムを最適化する方法がわかります。

OS/400 JVM の使用によりパフォーマンスを向上させる
OS/400 JVM を使用する際に得られるパフォーマンスの向上について読みます。

クライアント上での Toolbox for Java クラスの管理
クライアント上で IBM Toolbox for Java クラスを管理するために AS400ToolboxInstaller クラスを使用する方法について学びます。

JAR ファイルのパフォーマンスを向上させる
Toolbox for Java の JarMaker クラスを使用して、サイズが小さく、ロードが高速な IBM Toolbox for Java の JAR ファイルを作成する方法を参照します。

Java 各国語サポートを使用する
IBM Toolbox for Java および Java 各国語サポートの使用について読みます。

サービスおよびサポートの取得
これらの情報源を使用して Toolbox for Java のサポート・サービスを見つけます。

Java プログラムのシャットダウン

プログラムが適切にシャットダウンされるようにするためには、 Java プログラムが終了する前の最後の命令として System.exit(0) を発行しなければなりません。

注: サーブレットで System.exit(0) を使用することは避けてください。 それを行うと、Java 仮想マシン全体がシャットダウンされます。

IBM Toolbox for Java は、ユーザー・スレッドを使用してサーバーに接続します。 このため、System.exit(0) を発行しないと、 Java プログラムが適切にシャットダウンされなくなる可能性があります。

System.exit(0) の使用は必須ではありませんが、予防措置となります。 場合によっては、 Java プログラムを終了するためにこのコマンドを使用しなければならないこともあります。 また、必要ないときに System.exit(0) を使用しても、問題はありません。

サーバー・オブジェクトの統合ファイル・システム・パス名

Java プログラムがサーバー・オブジェクト (プログラム、ライブラリー、コマンド、 およびスプール・ファイルなど) を参照する場合には、統合ファイル・システム名を使用することが必要です。 統合ファイル・システム名は、 iSeries または AS/400e サーバー上の統合ファイル・システムのライブラリー・ファイル・システム内でアクセスするときのサーバー・オブジェクト名です。

パス名は、以下の各構成要素で構成されます。

パス名の構成要素	説明
library	オブジェクトが置かれているライブラリー。 library は、統合ファイル・システム・パス名に必須の部分です。 library は 10 文字以下にし、 その後に .lib を付けなければなりません。
object	統合ファイル・システム・パス名が表すオブジェクトの名前。 object は、統合ファイル・システム・パス名に必須の部分です。 オブジェクト名は 10 文字以下にし、その後に .type を付けなければなりません。 ただし、type はオブジェクトのタイプです。 タイプは、オブジェクトの処理 (WRKOBJ) などの制御言語 (CL) コマンドで、 OBJTYPE パラメーターの入力を求めることによって判別できます。
type	オブジェクトのタイプ。 object を指定するときに、 オブジェクトのタイプを指定しなければなりません。 (上記の object を参照してください。) type は 6 文字以下でなければなりません。
member	この統合ファイル・システム・パス名が表すメンバーの名前。 メンバーは、統合ファイル・システム・パス名のオプションの部分です。 指定できるのは、 オブジェクト・タイプが FILE のときだけです。 メンバー名は 10 文字以下にし、その後に .mbr を付けなければなりません。

統合ファイル・システム名を判別および指定するときには、 以下の条件に従ってください。

- スラッシュ (/) はパス区切り記号である。
- ルート・レベルのディレクトリー (QSYS.LIB) が、サーバー・ライブラリー構造を含む。
- サーバー・ライブラリー QSYS に置かれているオブジェクトのフォーマットが、次の形式である。

/QSYS.LIB/object.type

- その他のライブラリーに置かれているオブジェクトのフォーマットが、次の形式である。

/QSYS.LIB/library.LIB/object.type

- オブジェクト・タイプ拡張子は、そのタイプのオブジェクト用に使用されるサーバー省略語である。

タイプのリストを参照するには、 オブジェクト・タイプをパラメーターとして指定する CL コマンドを実行し、 F4 (プロンプト) を押して、タイプを調べてください。 たとえば、オブジェクトの処理 (WRKOBJ) コマンドはオブジェクト・タイプ・パラメーターをとります。

以下の表は、一般的に使用されるオブジェクト・タイプと各タイプの省略形のリストです。

オブジェクト・タイプ	省略語
コマンド	.CMD
データ待ち行列	.DTAQ
ファイル	.FILE
フォント・リソース	.FNTRSC
フォーム定義	.FORMDF
library	.LIB
member	.MBR
オーバーレイ	.OVL
ページ定義	.PAGDFN
ページ・セグメント	.PAGSET
プログラム	.PGM
出力待ち行列	.OUTQ
スプール・ファイル	.SPLF

統合ファイル・システム・パス名を指定する方法を判別するため、以下の説明を参考にしてください。

統合ファイル・システム名	説明
/QSYS.LIB/MY_LIB.LIB/MY_PROG.PGM	サーバー上の ライブラリー MY_LIB 内の プログラム MY_PROG

/QSYS.LIB/MY_LIB.LIB/MY_QUEUE.DTAQ	サーバー上の ライブラリー MY_LIB 内の データ待ち行列 MY_QUEUE
/QSYS.LIB/YEAR1998.LIB/MONTH.FILE/JULY.MBR	サーバー上の ライブラリー YEAR1998 内の ファイル MONTH 内の メンバー JULY

統合ファイル・システムの特殊値

さまざまな IBM Toolbox for Java クラスは、統合ファイル・システム・パス名の特殊値を認識します。 これらの特殊値の従来の形式 (iSeries コマンド行で使用される) は、 アスタリスクで始まります (*ALL)。 ですが、Toolbox for Java クラスで使用される Java プログラムでは、 これらの特殊値の形式は % 記号で始まり、終わります (%ALL%)。

注: 統合ファイル・システムでは、アスタリスクはワイルドカード文字です。

以下の表は、 特定のパス名の構成要素にこれら IBM Toolbox for Java クラスの特殊値のどれが認識するかを示します。 さらにこの表は、これらの特殊値の従来の形式が Toolbox for Java クラスで使用される形式とどのように異なるかも示します。

パス名の構成要素	従来の形式	Toolbox for Java の形式
ライブラリー名	*ALL	%ALL%
	*ALLUSR	%ALLUSR%
	*CURLIB	%CURLIB%
	*LIBL	%LIBL%
	*USRLIBL	%USRLIBL%
オブジェクト名	*ALL	%ALL%
メンバー名	*ALL	%ALL%
	*FILE	%FILE%
	*FIRST	%FIRST%
	*LAST	%LAST%

統合ファイル・システム名の作成と解析の詳細については、QSYSObjectPathNameクラスを参照してください。

統合ファイル・システムの概念に関する詳細については、統合ファイル・システムの概念を参照してください。

接続の管理

サーバーへの接続を作成し、開始し、その後終了できることは重要なことです。 以下の論議ではサーバーへの接続の管理を中心とする概念を説明し、さらにいくつかのコード例を提供します。

iSeries システムに接続するには、 Java プログラムで AS400 オブジェクトを作成する必要があります。 AS400 オブジェクトには、 iSeries サーバー・タイプごとに 1 つまでのソケット接続を含めることができます。 サービスはサーバー上のデータへのインターフェースであり、 サーバー上でのジョブに対応しています。

注: Enterprise JavaBeans (EJB) を作成するときには、 接続中のスレッドを許可しない EJB 仕様に従います。 IBM Toolbox for Java スレッド・サポートをオフにするとアプリケーションは遅くなる可能性がありますが、 EJB 仕様に従う必要があります。

各サーバーへのそれぞれの接続は、iSeries 上で独自のジョブを持ちます。 サーバーはそれぞれ、以下のものをサポートしています。

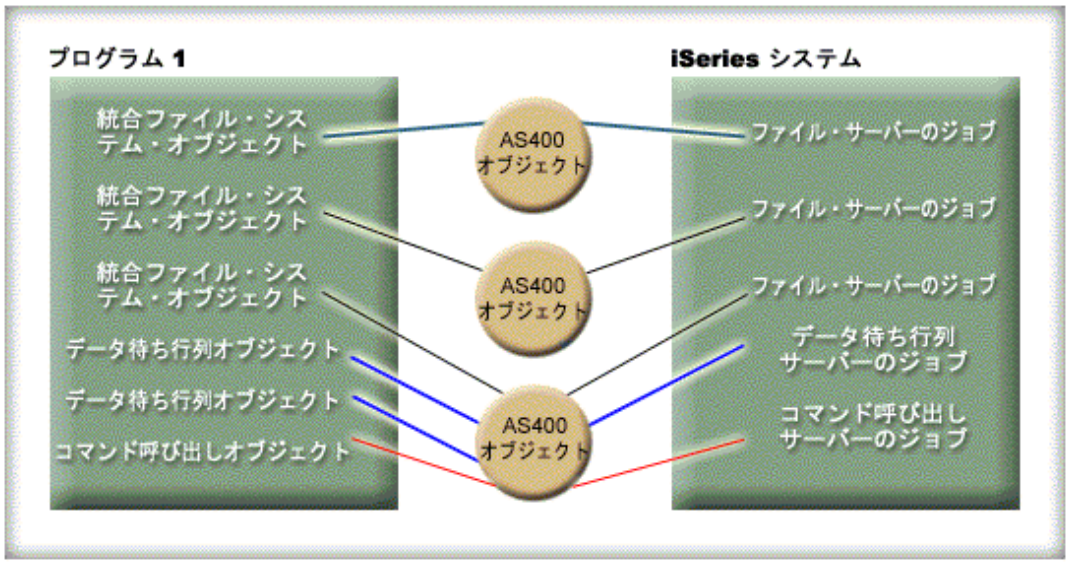
- JDBC
- プログラム呼び出しとコマンド呼び出し
- 統合ファイル・システム
- 印刷
- データ待ち行列
- レコード・レベルでのアクセス

注:

- アプリケーションが 2 つの事柄を同時にネットワーク印刷サーバーに要求することがない場合、 印刷クラスは AS400 オブジェクトごとに 1 つのソケット接続を使用します。
- 印刷クラスは、必要であれば、 ネットワーク印刷サーバーへの追加のソケット接続を作成します。追加の会話は、5 分間使用されないと切断されます。

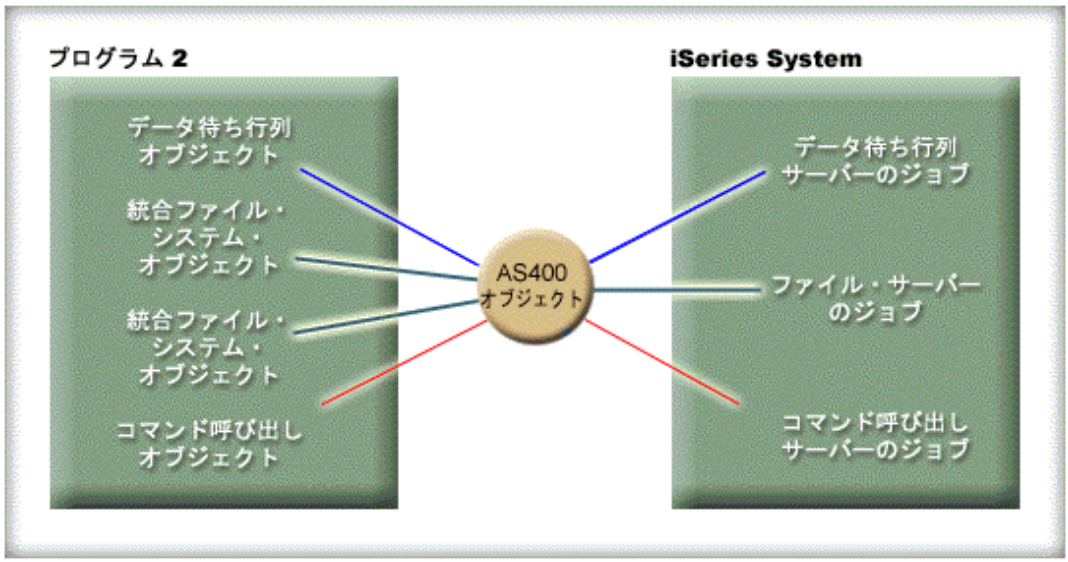
Java プログラムでは、iSeries への接続の数を制御することができます。 通信のパフォーマンスを最適化するために、Java プログラムでは、図 1 で示すように、 同じシステムについて複数の AS400 オブジェクトを作成することができます。 これにより、iSeries への複数のソケット接続が作成されます。

図 1: 同じ iSeries システムについて複数の AS400 オブジェクトとソケット接続を作成する Java プログラム



iSeries システム・リソースを節約するには、図 2 で示すように、 AS400 オブジェクトを 1 つだけ作成してください。このアプローチでは接続の数が減り、 そのことによって、システム上で使用されるリソースの量が減ります。

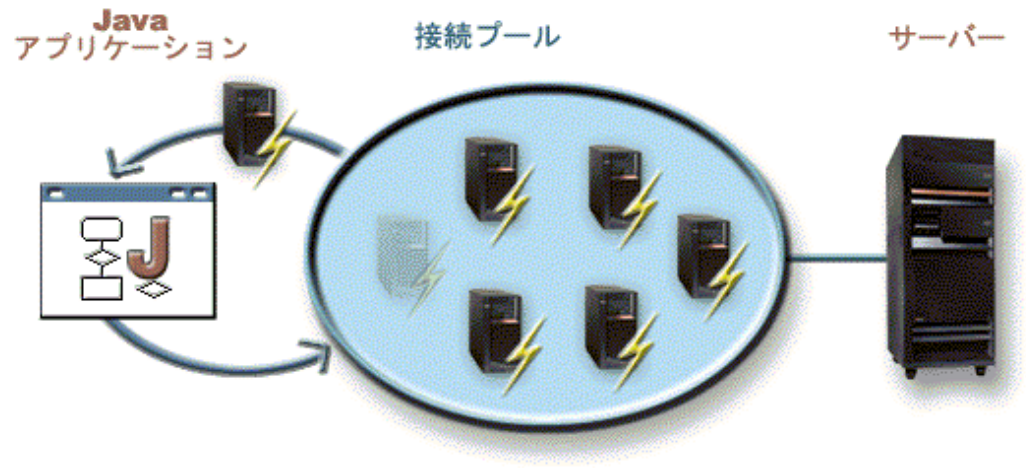
図 2: 同じ iSeries システムについて 1 つの AS400 オブジェクトとソケット接続を作成する Java プログラム



注: 多くの接続を作成するとシステム上で使用されるリソースの量が増えますが、 そうすることには利点があります。 多くの接続を持つことにより Java プログラムは並行で処理できるようになり、 それによりスループットが良くなり (秒当たりのトランザクション) またアプリケーションの速度も上がります。

図 3 で示すように、接続プールを使用することを選択して、接続を管理することもできます。 この方法により、ユーザー用に以前に確立された接続を再利用するため、 iSeries への接続にかかる時間を減らすことができます。

図 3: iSeries サーバーへの AS400ConnectionPool からの接続を取得する Java プログラム



以下の例では、AS400 オブジェクトを作成し、使用する方法を示します。

例 1: 以下の例では、同じ iSeries システムにコマンドを送信する 2 つの CommandCall オブジェクトが作成されます。 これらの CommandCall オブジェクトは同一の AS400 オブジェクトを使用するので、 システムへの接続は 1 つしか作成されません。

```
                    // Create an AS400 object.
AS400 sys = new AS400("mySystem.myCompany.com");

                    // Create two command call objects that use
                    // the same AS400 object.
CommandCall cmd1 = new CommandCall(sys,"myCommand1");
CommandCall cmd2 = new CommandCall(sys,"myCommand2");

                    // Run the commands.  A connection is made when the
                    // first command is run.  Since they use the same
```

```
                    // AS400 object the second command object will use
                    // the connection established by the first command.
cmd1.run();
cmd2.run();
```

例 2: 以下の例では、同じ iSeries システムにコマンドを送信する 2 つの CommandCall オブジェクトが作成されます。 これらの CommandCall オブジェクトは異なる AS400 オブジェクトを使用するので、 システムへの接続が 2 つ作成されます。

```
                    // Create two AS400 objects to the same server.
AS400 sys1 = new AS400("mySystem.myCompany.com");
AS400 sys2 = new AS400("mySystem.myCompany.com");

                    // Create two command call objects.  They use
                    // different AS400 objects.
CommandCall cmd1 = new CommandCall(sys1,"myCommand1");
CommandCall cmd2 = new CommandCall(sys2,"myCommand2");

                    // Run the commands.  A connection is made when the
                    // first command is run.  Since the second command
                    // object uses a different AS400 object, a second
                    // connection is made when the second command is run.
cmd1.run();
cmd2.run();
```

例 3: 以下の例では、同じ AS400 オブジェクトを使用して CommandCall オブジェクトと IFSFileInputStream オブジェクトが作成されます。 CommandCall オブジェクトと IFSFileInput Stream オブジェクトは iSeries 上の異なるサービスを使用するため、 2 つの接続が作成されます。

```
                    // Create an AS400 object.
AS400 newConn1 = new AS400("mySystem.myCompany.com");

                    // Create a command call object.
CommandCall cmd = new CommandCall(newConn1,"myCommand1");

                    // Create the file object.  Creating it causes the
                    // AS400 object to connect to the file service.
IFSFileInputStream file = new IFSFileInputStream(newConn1,"/myfile");

                    // Run the command.  A connection is made to the
                    // command service when the command is run.
cmd.run();
```

例 4: 以下の例では、AS400ConnectionPool を使用して iSeries 接続を取得します。 この例は (上記の例 3 のように) サービスを指定しないので、 コマンド接続へのサービスは、コマンドの実行時に行われます。

```
                    // Create an AS400ConnectionPool.
AS400ConnectionPool testPool1 = new AS400ConnectionPool();
                    // Create a connection.
AS400 newConn1 = testPool1.getConnection("myAS400", "myUserID", "myPassword");
                    // Create a command call object that uses the AS400 object.
CommandCall cmd = new CommandCall(newConn1,"myCommand1");
                    // Run the command.  A connection is made to the
                    // command service when the command is run.
cmd.run();
                    // Return connection to pool.
testPool1.returnConnectionToPool(newConn1);
```

例 5: 以下の例では、 AS400ConnectionPool を使用して、プールから接続を要求するときに特定のサービスに接続します。 これによって、コマンドの実行時に、 サービスへの接続に必要な時間を省くことができます (上記の例 4

を参照)。 接続がプールに戻されると、接続を入手するための次の呼び出しは、 同じ接続オブジェクトを戻すことができます。 つまり、作成にも使用にも、余分の接続時間が必要ないということです。

```
                    // Create an AS400ConnectionPool.
    AS400ConnectionPool testPool1 = new AS400ConnectionPool();
                    // Create a connection to the AS400.COMMAND service. (Use the service number
constants
                    // defined in the AS400 class (FILE, PRINT, COMMAND, DATAQUEUE, etc.))
    AS400 newConn1 = testPool1.getConnection("myAS400", "myUserID", "myPassword", AS400.COMMAND);
                    // Create a command call object that uses the AS400 object.
    CommandCall cmd = new CommandCall(newConn1,"myCommand1");
                    // Run the command.  A connection has already been made
                    // to the command service.
    cmd.run();
                    // Return connection to pool.
    testPool1.returnConnectionToPool(newConn1);
                    // Get another connection to command service.  In this case, it will return the
same
                    // connection as above, meaning no extra connection time will be needed either
now or when the
                    // command service is used.
    AS400 newConn2 = testPool1.getConnection("myAS400", "myUserID", "myPassword", AS400.COMMAND);
```

接続の開始および終了

Java プログラムでは、接続がいつ開始し、いつ終了するかを制御することができます。 デフォルトでは、サーバーからの情報が必要になったときに、接続が開始されます。 AS400 オブジェクトにおいて connectService() メソッドを呼び出してサーバーに事前接続することによって、 接続がいつ行われるのかを正確に制御できます。

上記の例 5 で説明しているように、 AS400ConnectionPool を使用すれば、connectService() メソッドを呼び出さなくても、 サービスに事前接続された接続を作成できます。

以下の例では、iSeries への接続を確立および切断する Java プログラムを示します。

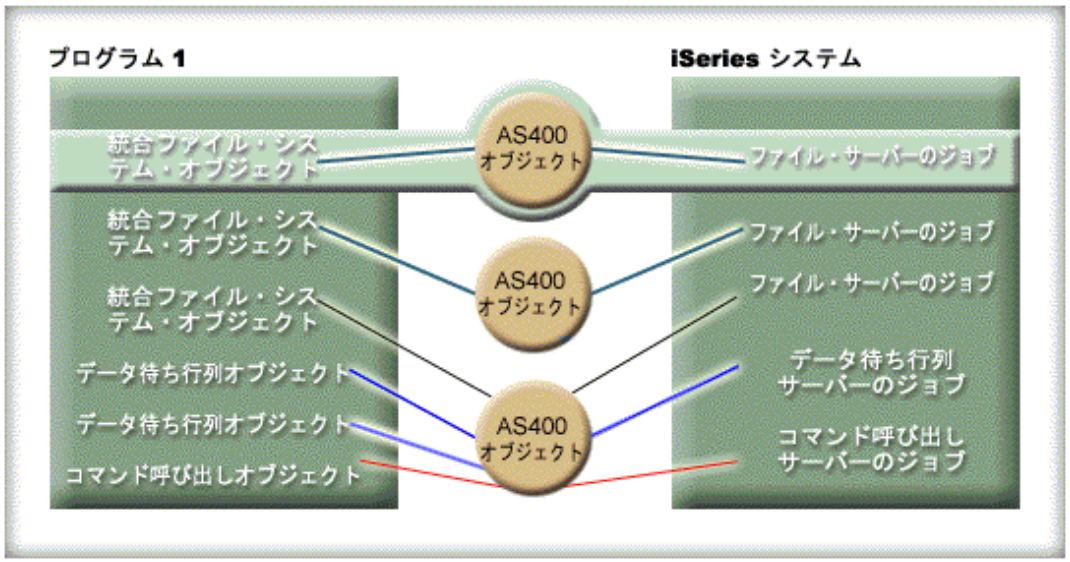
例 1: この例は、iSeries に事前接続する方法を示します。

```
                    // Create an AS400 object.
    AS400 system1 = new AS400("mySystem.myCompany.com");

                    // Connect to the command service.  Do it now
                    // instead of when data is first sent to the
                    // command service.  This is optional since the
                    // AS400 object will connect when necessary.
    system1.connectService(AS400.COMMAND);
```

例 2: 接続の開始後は、Java プログラムが切断を受け持ちます。 切断は、AS400 オブジェクトによって暗黙的に、あるいは Java プログラムによって明示的に行われます。Java プログラムでは、 AS400 オブジェクトにおいて disconnectService() メソッドを呼び出すことによって切断を行います。 パフォーマンスを向上させるために、 Java プログラムでは、サービスを使用し終えたときにのみ切断しなければなりません。 Java プログラムがサービスを使用し終える前に切断すると、AS400 オブジェクトは、 サービスからデータが必要となったときに再接続します (可能な場合)。

図 4 では、最初の統合ファイル・システム・オブジェクト接続について接続が切断される場合に、 すべての統合ファイル・システム・オブジェクト接続ではなく、 その AS400 オブジェクト接続の 1 つのインスタンスだけが終了される仕組みを示します。

図 4: AS400 オブジェクトのインスタンスで独自のサービスを使用している単一のオブジェクトが切断される

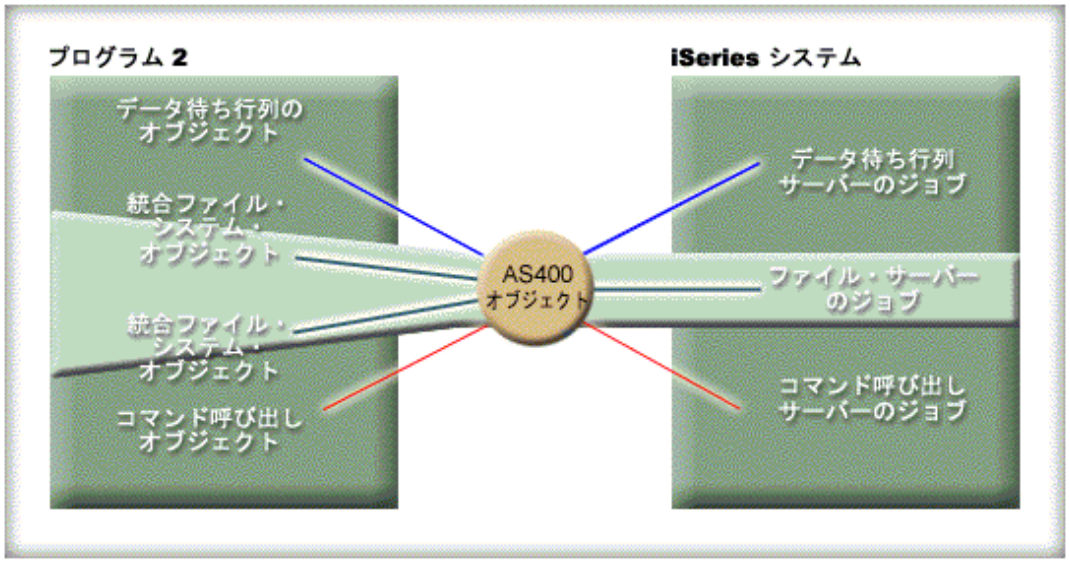


この例では、Java プログラムが接続を切断する方法を示します。

```
                    // Create an AS400 object.
AS400 system1 = new AS400("mySystem.myCompany.com");

                    // ... use command call to send several commands
                    // to the server.  Since connectService() was not
                    // called, the AS400 object automatically
                    // connects when the first command is run.

                    // All done sending commands so disconnect the
                    // connection.
system1.disconnectService(AS400.COMMAND);
```

例 3: 同じサービスを使用し、 同じ AS400 オブジェクトを共用する複数のオブジェクトは、接続を共用します。 接続が切断されると、図 5 で示すように、 AS400 オブジェクトのそれぞれのインスタンスで同じサービスを使用しているすべてのオブジェクトについて接続が終了されます。

図 5: AS400 オブジェクトのインスタンスで同じサービスを使用しているすべてのオブジェクトが切断される



たとえば、2 つの CommandCall オブジェクトが同一の AS400 オブジェクトを使用しているとします。

disconnectService() が呼び出されると、 接続は 2 つの CommandCall オブジェクト両方に関して終了します。 2 番目の CommandCall オブジェクトについて run() メソッドが呼び出されると、 AS400 オブジェクトはサービスに再接続することが必要になります。

```
                    // Create an AS400 object.
AS400 sys = new AS400("mySystem.myCompany.com");

                    // Create two command call objects.
CommandCall cmd1 = new CommandCall(sys,"myCommand1");
CommandCall cmd2 = new CommandCall(sys,"myCommand2");

                    // Run the first command
cmd1.run();

                    // Disconnect from the command service.
sys.disconnectService(AS400.COMMAND);

                    // Run the second command.  The AS400 object
                    // must reconnect to the server.
cmd2.run();

                    // Disconnect from the command service.  This
                    // is the correct place to disconnect.
sys.disconnectService(AS400.COMMAND);
```

例 4: すべての IBM Toolbox for Java クラスが自動的に再接続するわけではありません。 統合ファイル・システム・クラスにおける一部のメ ソッド呼び出しでは、 ファイルが変更されている可能性があるため再接続しません。 ファイルが切断されている間に、 他の何らかのプロセスによってそのファイルが削除されているか、 または内容が変更されている可能性があります。 以下の例では、2 つのファイル・オブジェクトが同一の AS400 オブジェクトを使用します。 disconnectService() が呼び出されると、接続は両方のファイル・オブジェクトについて終了します。 2 番目の IFSFileInputStream オブジェクトに対して read() を実行しても、 このオブジェクトがサーバーから切断されているために失敗します。

```
                    // Create an AS400 object.
AS400 sys = new AS400("mySystem.myCompany.com");

                    // Create two file objects.  A connection to the
                    // server is created when the first object is
                    // created.  The second object uses the connection
                    // created by the first object.
IFSFileInputStream file1 = new IFSFileInputStream(sys,"/file1");
IFSFileInputStream file2 = new IFSFileInputStream(sys,"/file2");

                    // Read from the first file, then close it.
int i1 = file1.read();
file1.close();

                    // Disconnect from the file service.
sys.disconnectService(AS400.FILE);

                    // Attempt to read from the second file.  This
                    // fails because the connection to the file service
                    // no longer exists.  The program must either
                    // disconnect later or have the second file use a
                    // different AS400 object (which causes it to
                    // have its own connection).
int i2 = file2.read();

                    // Close the second file.
```

```
file2.close();

                // Disconnect from the file service.  This
                // is the correct place to disconnect.
sys.disconnectService(AS400.FILE);
```

OS/400 Java 仮想マシン

IBM Toolbox for Java クラスは、 IBM Developer Kit for Java (OS/400) の Java 仮想マシン (JVM) 上で実行します。 実際には、それらのクラスは、 Java Development Kit (JDK) 1.1.x および Java 2 Software Development Kit (J2SDK) 仕様をサポートする、 すべてのプラットフォームでも実行します。

IBM Toolbox for Java クラスを OS/400 JVM 上で実行する場合は、以下のことを行ってください。

- OS/400 JVM での実行時に iSeries サーバー・リソースにアクセスするには、 OS/400 JVM と IBM Toolbox for Java クラスのいずれを使用するかを選択する。
- OS/400 JVM 上での IBM Toolbox for Java クラスの実行について調べる。
- OS/400 JVM でのシステム名、ユーザー ID、およびパスワードの設定について読む。

さまざまな Java プラットフォーム用の iSeries サーバー・サポートについての詳細は、 IBM Developer Kit for Java にある、 複数の JDK のサポートを参照してください。

OS/400 Java 仮想マシンと IBM Toolbox for Java クラスの比較

Java プログラムが IBM Developer Kit for Java (OS/400) の Java 仮想マシン (JVM) で実行されているときには、 常に、IBM リソースにアクセスするための少なくとも 2 つの方法があります。 次の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。

- Java に組み込まれた機能
- IBM Toolbox for Java クラス

いずれのインターフェースを使用するかを決めるにあたっては、 以下の要素を考慮してください。

- 場所 - プログラムが実行される場所は、 使用するインターフェース・セットを決定する際の最も重要な要素です。 プログラムは以下のどの状況に当てはまりますか?
  - クライアント上でのみ実行される
  - サーバー上でのみ実行される
  - クライアントとサーバーの両方で実行されるが、 どちらのケースもリソースは iSeries サーバー・リソースである
  - ある OS/400 JVM 上で実行してから、別の iSeries サーバー上でリソースにアクセスする
  - さまざまな種類のサーバー上で実行される

  プログラムがクライアントとサーバーの両方で実行され (ある iSeries サーバーが別の iSeries サーバーのクライアントになっている場合も含む)、 iSeries サーバー・リソースにのみアクセスする場合は、 IBM Toolbox for Java インターフェースを使用することが最善であると思われます。

  プログラムがさまざまな種類のサーバーにアクセスしなければならない場合には、 Java のネイティブ・インターフェースを使用することが最善であると思われます。
- 整合性/可搬性 - iSeries サーバー上で IBM Toolbox for Java を実行できるということは、 クライアント・プログラムとサーバー・プログラムの両方で同じインターフェースを使用できるということです。 クライアント・プログラムとサーバー・プログラムの両方について学習すべきインターフェースが 1 つならば、 生産性が向上します。

  ただし、IBM Toolbox for Java インターフェースへの書き込みによって、 サーバーへのプログラムの可搬性が低下します。

プログラムを iSeries サーバーだけでなく他のサーバーに対しても実行しなければならない場合は、 Java に組み込まれている機能を使用する方がよいでしょう。

- 複雑さ - IBM Toolbox for Java インターフェースは、 特に、iSeries サーバー・リソースに簡単にアクセスできるように作成されています。 IBM Toolbox for Java インターフェースを使用することの代わりとなる唯一の方法が、 リソースにアクセスするプログラムを作成し、 Java ネイティブ・インターフェース (JNI) を通じてそのプログラムと通信することであることが、頻繁にあります。

  Java のより高い中立性を保ち、リソースにアクセスするネイティブ・プログラムを作成することと、 可搬性のより低い IBM Toolbox for Java インターフェースを使用することのどちらが重要であるかを判断しなければなりません。
- 機能 - 多くの場合、IBM Toolbox for Java インターフェースは、 Java インターフェースよりも多くの機能を提供します。 たとえば、IBM Toolbox for Java ライセンス・プログラムの IFSFileOutputStream クラスには、java.io の FileOutputStream クラスよりも多くの機能があります。 ただし、IFSFileOutputStream を使用すると、プログラムは iSeries サーバーにしか使用できなくなります。 IBM Toolbox for Java クラスを使用することによって、 サーバーへの移植性を失うことになります。

  移植性が重要であるか、それとも追加の機能を利用したいかを判断しなければなりません。
- リソース - OS/400 JVM で実行されているときでも、 IBM Toolbox for Java クラスの多くはホスト・サーバーを通じて要求を行います。 したがって、2 番目のジョブ (サーバー・ジョブ) がリソースへのアクセス要求を実行することになります。

  この要求の場合、 Java プログラムのジョブのもとで実行される Java ネイティブ・インターフェースよりも消費するリソースが多くなる可能性があります。
- クライアントとしての iSeries サーバー - プログラムが iSeries サーバーで実行されており、 別の iSeries サーバー上でデータにアクセスする場合は、 IBM Toolbox for Java クラスを使用することが最善であると思われます。 これらのクラスを使用すると、別の iSeries サーバー上のリソースに簡単にアクセスすることができます。

  その例として、データ待ち行列のアクセスがあります。 IBM Toolbox for Java ライセンス・プログラムのデータ待ち行列インターフェースは、データ待ち行列リソースに簡単にアクセスするための手段となります。

  また、IBM Toolbox for Java を使用することにより、 プログラムをクライ

アントとサーバーの両方で実行し、iSeries サーバーにアクセスできるようになります。 さらに、ある iSeries サーバーで実行中のプログラムから、 別の iSeries サーバー上のデータ待ち行列にアクセスできるようにもなります。

代わりの方法は、データ待ち行列にアクセスする別個のプログラム (たとえば、C) を作成することです。 データ待ち行列へのアクセスが必要になったときに、 Java プログラムがこのプログラムを呼び出します。

この方法では、サーバーへの可搬性が高くなります。 つまり、データ待ち行列のアクセスを処理する 1 つの Java プログラムと、 サポートするサーバーごとに異なるバージョンのプログラムを持つことができます。

OS/400 Java 仮想マシン上での IBM Toolbox for Java クラスの実行

以下の情報は、 IBM Developer Kit for Java (OS/400) の Java 仮想マシン (JVM) 上で IBM Toolbox for Java クラスを実行する際の特別の考慮事項です。

JDBC

OS/400 JVM 上で実行されるプログラムでは、IBM 提供の次の 2 つの JDBC ドライバーが使用可能です。

- IBM Toolbox for Java JDBC ドライバー
- IBM Developer Kit for Java JDBC ドライバー

IBM Toolbox for Java JDBC ドライバーは、 プログラムがクライアント/サーバー環境で実行される場合に適しています。

IBM Developer Kit for Java JDBC ドライバーは、 プログラムが iSeries サーバー上で実行される場合に適しています。

同じプログラムがワークステーションとサーバーの両方で実行される場合には、 プログラム内にドライバー名をコーディングするのではなく、 システム・プロパティーを通じて正しいドライバーをロードしなければなりません。

プログラム呼び出し

プログラムを呼び出すための一般的な方法には、次の 2 つがあります。

- IBM Toolbox for Java の ProgramCall クラス
- Java ネイティブ・インターフェース (JNI) 呼び出しの使用

IBM Toolbox for Java ライセンス・プログラムの ProgramCall クラスには、 任意の iSeries サーバー・プログラムを呼び出すことができるという利点があります。

JNI では、iSeries サーバー・プログラムを呼び出すことができない可能性があります。 JNI の利点は、サーバー・プラットフォーム間での移植性が高いことにあります。

コマンド呼び出し

コマンドを呼び出すための一般的な方法には、次の 2 つがあります。

- IBM Toolbox for Java の CommandCall クラス
- java.lang.runtime.exec()

CommandCall クラスは、 コマンドの完了後に Java プログラムで使用できるようになるメッセージのリストを生成します。 このメッセージのリストは、 java.lang.runtime.exec() では使用できません。

java.lang.runtime.exec() には多くのプラットフォーム間での可搬性があるため、 プログラムがさまざまなタイプのサーバー上のファイルにアクセスしなければならない場合には、 java.lang.runtime.exec() を使用すると効果的です。

統合ファイル・システム

iSeries サーバーの統合ファイル・システム内のファイルにアクセスするための一般的な方法

- IBM Toolbox for Java ライセンス・プログラムの IFSFile クラス
- java.io の一部であるファイル・クラス

IBM Toolbox for Java の統合ファイル・システム・クラスには、 java.io クラスよりも提供される機能が多いという利点があります。 IBM Toolbox for Java クラスは、アプレット内でも機能し、また、 ワークステーションからサーバーに到達するための宛先変更のメソッド (iSeries Access for Windows など) を必要としません。

java.io クラスは、多数のプラットフォーム間での移植性があり、このことが利点となっています。 プログラムがさまざまなタイプのサーバー上のファイルにアクセスしなければならない場合には、 java.io を使用すると効果的です。

クライアント上で java.io クラスを使用する場合には、 サーバー・ファイル・システムに到達するための宛先変更のメソッド (iSeries Access for Windows など) が必要になります。

OS/400 Java 仮想マシン内の AS400 オブジェクトでのシステム名、 ユーザー ID、およびパスワードの設定

Java プログラムが IBM Developer Kit for Java (OS/400) の Java 仮想マシン (JVM) で実行されている場合、 AS400 オブジェクトはシステム名、ユーザー ID、 およびパスワードについて特殊値を使用できます。

プログラムを OS/400 JVM で実行するときには、いくつかの特殊値とその他の考慮事項に注意してください。

- プログラムがサーバー上で実行されている場合には、 ユーザー ID およびパスワードのプロンプトが使用不可にされます。 サーバー環境でのユーザー ID およびパスワード値の詳細については、AS400 オブジェクトのユーザー ID とパスワードに関する要約を参照してください。
- AS400 オブジェクトでシステム名、ユーザー ID、またはパスワードが設定されていない場合、 AS400 オブジェクトは Java プログラムを開始したジョブのユーザー ID とパスワードを使用して現行のサーバーに接続します。 v4r3 以前のマシンに接続する際にレコード・レベルでのアクセスを使用するときには、 パスワードを指定する必要があります。 v4r4 以降のマシンに接続する際には、 IBM Toolbox for Java の他のコンポーネントと同様、 サインオンされたユーザー・パスワードを配布することができます。
- 特殊値 localhost は、 システム名として使用することができます。 この場合、AS400 オブジェクトは現行のサーバーに接続します。
- 特殊値 *current は、 AS400 オブジェクトでユーザー ID またはパスワードとして使用することができます。 この場合、Java プログラムを開始したジョブのユーザー ID またはパスワード (あるいはその両方) が使用されます。 *current に関する詳細は、以下の注を参照してください。
- 特殊値*current は、 ある iSeries サーバーの OS/400 JVM で実行されている Java プログラムが、 別の iSeries サーバー上のリソースにアクセスするときに、 AS400 オブジェクトでユーザー ID またはパスワードとして使用できます。 この場合、受動システムへの接続時には、起動システム上の Java プログラムを開始したジョブのユーザー ID とパスワードが使用されます。 *current に関する詳細は、以下の注を参照してください。

  注:

  - レコード・レベルでのアクセスおよび V4R3 以前の使用時には、 Java プログラムはパスワードを "*current" に設定できません。 レコード・レベルでのアクセスを使用する場合、 システム名には "localhost"、ユーザー ID には "*current" が有効ですが、 Java プログ

ラムでパスワードを指定する必要があります。

- *current は、バージョン 4 リリース 3 (V4R3) 以降で実行されているシステムでのみ機能します。 V4R2 システムで実行されているシステムでは、 パスワードとユーザー ID を指定する必要があります。

以下の例では、OS/400 JVM と一緒に AS400 オブジェクトを使用する方法を示します。

例 1: Java プログラムが OS/400 JVM で実行される場合、 そのプログラムでシステム名、ユーザー ID、またはパスワードを指定する必要はありません。

レコード・レベルでのアクセスの使用時には、 パスワードを指定する必要があります。

これらの値が提供されない場合、AS400 オブジェクトは、 Java プログラムを開始したジョブのユーザー ID とパスワードを使用してローカル・システムに接続します。

プログラムが AS/400 用の OS/400 JVM で実行される場合、 システム名を localhost に設定することは、システム名を設定しないことと同じです。 以下の例では、現行のサーバーに接続する方法を示します。

```
// Create two AS400 objects.  If the Java program is running in the
// OS/400 JVM, the behavior of the two objects is the same.
// They will connect to the current server using the user ID and
// password of the job that started the Java program.
AS400 sys  = new AS400()
AS400 sys2 = new AS400("localhost")
```

例 2: Java プログラムは、 OS/400 JVM で実行される場合でも、ユーザー ID およびパスワードを設定することができます。 これらの値は、Java プログラムを開始したジョブのユーザー ID とパスワードをオーバーライドします。

以下の例では、Java プログラムは現行のサーバーに接続しますが、 そのプログラムを開始したジョブのものとは異なるユーザー ID およびパスワードを使用します。

```
// Create an AS400 object.  Connect to the current server but do
// not use the user ID and password of the job that started the
// program. The supplied values are used.
AS400 sys = new AS400("localhost", "USR2", "PSWRD2")
```

例 3: あるサーバーで実行されている Java プログラムが、 他の iSeries システムのリソースにアクセスし、使用することができます。

ユーザー ID とパスワードに *current が使用されている場合、 Java プログラムがターゲット・サーバーにアクセスするときには、 そのプログラムを開始したジョブのユーザー ID とパスワードが使用されます。

以下の例では、Java プログラムはあるサーバー上で実行されていますが、 別のサーバーのリソースを使用します。 Java プログラムが別のサーバーに接続するときには、 そのプログラムを開始したジョブのユーザー ID とパスワードが使用されます。

```
// Create an AS400 object. This program will run on one server
// but will connect to a second server (called "target").
// Because *current is used for user ID and password, the user
// ID and password of the job that started the program will be
// used when connecting to the second server.
AS400 target = new AS400("target", "*current", "*current")
```

独立補助記憶域プール (IASP)

独立補助記憶域プール (IASP) は、 システム上の他の記憶域からは独立してオンラインまたはオフラインにできるディスク装置の集合です。 IASP には以下のいずれかが含まれます。

- 1 つ以上のユーザー定義ファイル・システム
- 1 つ以上の外部ライブラリー

各 IASP には、そこに含まれるデータに関連して必要なすべてのシステム情報が含まれます。 ですからシステムがアクティブになっている間は、IASP をオフラインに、またはオンラインに、 あるいはシステムを切り替えることができます。

詳細については、独立 ASP およびユーザー ASP を参照してください。

"database name" JDBC プロパティーまたは AS400JDBCDataSource クラスからの setDatabaseName() メソッドを使用して、接続先の ASP を指定できます。

他のすべての Toolbox for Java クラス (IFSFile、Print、DataQueues など) は、サーバーへ接続するユーザー・プロファイルのジョブ記述によって指定される IASP を使用します。

OS/400 の最適化

IBM Toolbox for Java ライセンス・プログラムは、Java で書かれているため、 承認された Java 仮想マシン (JVM) があるどのプラットフォームでも稼働します。 IBM Toolbox for Java クラスは、実行場所に関係なく同様に機能します。

OS/400 には、 iSeries JVM での実行時に IBM Toolbox for Java の動作を向上させる追加のクラスがあります。 iSeries JVM での実行時に、同じ iSeries に接続する際には、 サインオン動作とパフォーマンスが向上します。 OS/400 では、バージョン 4 リリース 3 から、追加のクラスを組み込みました。

最適化を使用可能にする

IBM Toolbox for Java は 2 つのパッケージで出荷されます。 すなわち、個別のライセンス・プログラムとして、または OS/400 と一緒に出荷されます。

- ライセンス・プログラム 5722-JC1。 IBM Toolbox for Java のライセンス・プログラム・バージョンでは、ファイルは出荷時に次のディレクトリーに入っています。

  ```
  /QIBM/ProdData/http/public/jt400/lib
  ```

  これらのファイルには OS/400 の最適化は含まれていません。 クライアント上での IBM Toolbox for Java の実行と動作の一貫性を持たせる場合には、これらのファイルを使用してください。
- OS/400。IBM Toolbox for Java は、OS/400 の出荷時に次のディレクトリーに入っています。

  ```
  /QIBM/ProdData/OS400/jt400/lib
  ```

  これらのファイルには、iSeries JVM での実行中に IBM Toolbox for Java を最適化するクラスは含まれていません。

詳細については、JAR ファイルについての情報の注 1 を参照してください。

サインオンの考慮事項

OS/400 で提供される追加のクラスにより、 Java プログラムは、IBM Toolbox for Java にシステム名、ユーザー ID、 およびパスワード情報を提供するための追加のオプションを持つことになります。

iSeries リソースへのアクセス時には、IBM Toolbox for クラスがシステム名、 ユーザー ID、およびパスワードを持っていなければなりません。

- クライアントでの実行時には、 システム名、ユーザー ID、およびパスワードは Java プログラムによって提供されます。 あるいは、IBM Toolbox for Java はサインオン・ダイアログを通じてこれらの値をユーザーから取得します。
- iSeries Java 仮想マシンでの実行時には、 IBM Toolbox for Java にもう 1 つのオプ

ションがあります。 Java プログラムを開始したジョブのユーザー ID とパスワードを使用して、 現行 (ローカル) サーバーに要求を送ることができます。

追加のクラスを使用すると、 ある iSeries で実行されている Java プログラムが別の iSeries 上のリソースにアクセスするときにも、 現行ジョブのユーザー ID とパスワードを使用することができます。 このケースでは、Java プログラムでシステム名を設定し、 ユーザー ID とパスワードに特殊値 "*current" を使用します。

レコード・レベルでのアクセス V4R4 以降を使用する場合には、 Java プログラムはパスワードを "*current" のみにしか設定できません。 それ以外の場合、レコード・レベルでのアクセスを使用するときには、 システム名には "localhost"、ユーザー ID には "*current" が有効ですが、 Java プログラムでパスワードを指定する必要があります。

Java プログラムは、システム名、ユーザー ID、 およびパスワードの値を AS400 オブジェクト内で設定します。

ジョブのユーザー ID とパスワードを使用するために、 Java プログラムでは、ユーザー ID およびパスワードとして "*current" を使用するか、 あるいはユーザー ID およびパスワード・パラメーターを持たないコンストラクターを使用することができます。

現行の iSeries を使用するために、Java プログラムでは、 システム名として "localhost" を使用するか、 あるいはデフォルト・コンストラクターを使用することができます。 つまり、次の指定は、

```
AS400 system = new AS400();
```

次の指定と同じです。

```
AS400 system = new AS400("localhost", "*current", "*current");
```

以下の例では、2 つの AS400 オブジェクトを作成します。 2 つのオブジェクトの動作は同じです。つまり、両方とも、 ジョブのユーザー ID とパスワードを使用して現行の iSeries に対してコマンドを実行します。 一方のオブジェクトはユーザー ID およびパスワードとして特殊値を使用し、 他方はデフォルト・コンストラクターを使用し、ユーザー ID またはパスワードを設定しません。

```
                    // Create an AS400 object. Since the default
                    // constructor is used and system, user ID and
                    // password are never set, the AS400 object sends
                    // requests to the local iSeries using the job's
                    // user ID and password. If this program were run
                    // on a client, the user would be prompted for
                    // system, user ID and password.
AS400 sys1 = new AS400();

                    // Create an AS400 object. This object sends
```

```
                    // requests to the local iSeries using the job's
                    // user ID and password. This object will not work
                    // on a client.
AS400 sys2 = new AS400("localhost", "*current", "*current");

                    // Create two command call objects that use the
                    // AS400 objects.
CommandCall cmd1 = new CommandCall(sys1,"myCommand1");
CommandCall cmd2 = new CommandCall(sys2,"myCommand2");

                    // Run the commands.
cmd1.run();
cmd2.run();
```

以下の例では、2 番目の iSeries システムを表す AS400 オブジェクトを作成します。"*current" が使用されているため、2 番目の (ターゲット) iSeries では、 Java プログラムを実行している iSeries からのジョブのユーザー ID とパスワードが使用されます。

```
                    // Create an AS400 object. This object sends
                    // requests to a second iSeries using the user ID
                    // and password from the job on the current iSeries.
AS400 sys = new AS400("mySystem.myCompany.com", "*current", "*current");


                    // Create a command call object to run a command
                    // on the target iSeries.
CommandCall cmd = new CommandCall(sys,"myCommand1");


                    // Run the command.
cmd.run();
```

パフォーマンスの向上

OS/400 によって提供される追加のクラスにより、 iSeries 用の Java 仮想マシンで実行される Java プログラムのパフォーマンスが向上します。 パフォーマンスが向上するのは、使用される通信機能がより少なくなるケースや、 サーバー・プログラムを呼び出す代わりに iSeries API が使用されるケースです。

ダウンロード時間の短縮

最低限の数の IBM Toolbox for Java クラス・ファイルをダウンロードするには、 AS400ToolboxJarMaker ツールとともに proxy サーバーを使用します。

高速な通信

JDBC および統合ファイル・システム・アクセスを除くすべての IBM Toolbox for Java 機能では、 iSeries 用の Java 仮想マシンで実行されている Java プログラムは実行速度がより速くなります。 プログラムの実行が速くなるのは、Java プログラムと、 要求を行うサーバー上のサーバー・プログラムとの通信時に使用される通信コードがより少ないためです。

JDBC および統合ファイル・システム・アクセスでは、 これらの機能をより速く実行する機能がすでに存在するため、最適化されていません。 iSeries での実行時には、 IBM Toolbox for Java に付属する JDBC ドライバーではなく、 iSeries 用の JDBC ドライバーを使用することができます。 サーバー上のファイルにアクセスするには、 IBM Toolbox for Java に付属する統合ファイル・システム・アクセス・クラスではなく、 java.io を使用することができます。

iSeries API の直接呼び出し

IBM Toolbox for Java の次のクラスのパフォーマンスは、 これらのクラスが、要求を実行するサーバー・プログラムを呼び出す代わりに、 iSeries API を直接呼び出すため、向上します。

- AS400Certificate クラス
- CommandCall
- DataQueue
- ProgramCall
- レコード・レベルのデータベース・アクセス・クラス
- ServiceProgramCall
- UserSpace

API が直接呼び出されるのは、ユーザー ID とパスワードが、 Java プログラムを実行しているジョブのユーザー ID とパスワードと一致する場合だけです。 パフォーマンスを向上させるには、ユーザー ID およびパスワードが、 Java プログラムを開始するジョブのユーザー ID およびパスワードと一致しなければなりません。 最善の結果を得るには、システム名として "localhost"、 ユーザー ID として "*current"、 パスワードとして "*current" を使用してください。

ポート・マッピングの変更

ポート・マッピング・システムが変更され、 より速くポートにアクセスできるようになりました。 このような変更が行われる前は、ポートへの要求はポート・マッパーへ送られました。 そこで、使用できるポートを iSeries が判別し、 要求を送ったユーザーにそのポートを戻しました。 今では、サーバーに使用するポートを指示するか、 デフォルト・ポートを使用するように指定できます。 このオプションにより、 サーバーがポートを決めるのにかかる無駄な時間が節約されます。 WRKSRVTBLE コマンドを使用すれば、 サーバーのポートのリストを表示したり変更したりできます。

ポート・マッピングを改良するため、 以下のメソッドが AS400 クラスに追加されました。

- getServicePort
- setServicePort
- setServicePortsToDefault

MRI に関する変更

今回 MRI ファイルは、プロパティー・ファイルではなく、 クラス・ファイルとして IBM Toolbox for Java プログラムに付属しています。 iSeries の場合、プロパティー・ファイルよりも、 クラス・ファイルの方がメッセージを速く見つけることができます。 コンピューターが検索する最初の場所に MRI ファイルが保管されているため、 ResourceBundle.getString() の実行速度が速くなっています。 さらに、クラス・ファイルへの変更によって、 サーバーがストリングの変換版を見つける速度も上がっています。

コンバーター

以下の 2 つのクラスによって、 Java と iSeries との間の変換をより速く、より効率的に行えます。

- バイナリー・コンバーター: Java バイト配列と Java 単純タイプの間の変換を行います。
- 文字コンバーター: Java のストリング・オブジェクトと iSeries のコード・

パッケージの間の変換を行います。

また、 現在 IBM Toolbox for Java には、100 以上の通常使用される CCSID 用の独自の変換テーブルが組み込まれています。 以前は、IBM Toolbox for Java がほとんどすべてのテキスト変換用に Java に据え置かれていました。 Java が正しい変換テーブルを持っていない場合には、 IBM Toolbox for Java がサーバーから変換テーブルをダウンロードしていました。

IBM Toolbox for Java は、認識する CCSID についてすべてのテキスト変換を実行します。 不明な CCSID を見つけると、Java が変換を処理するように試みます。 IBM Toolbox for Java がサーバーから変換テーブルをダウンロードすることはなくなります。 この技法は、IBM Toolbox for Java アプリケーションがテキスト変換を実行するのにかかる時間を大幅に削減します。 この新しいテキスト変換を利用するためにユーザーがアクションをとる必要はありません。 パフォーマンス向上はすべて、基礎となるコンバーター・テーブルで行われます。

Java プログラム作成 (CRTJVAPGM) コマンドについてのパフォーマンスのヒント

Java アプリケーションを iSeries 用の Java 仮想マシン (JVM) で実行する場合、IBM Java Toolbox for Java の ZIP ファイルまたは JAR ファイルから Java プログラムを作成することにより、 パフォーマンスを大幅に向上できます。 このプログラムを作成するには、 iSeries コマンド行で CRTJVAPGM を入力します。 (CRTJVAPGM コマンドの詳細については、オンライン・ヘルプ情報を参照してください。) CRTJVAPGM コマンドを使用することにより、 Java アプリケーションの開始時に作成される (IBM Toolbox for Java クラスに含められる) Java プログラムを保管します。 作成された Java プログラムを保管しておけば、始動処理時間を短縮できます。 始動処理時間を短縮できるのは、 Java アプリケーションを開始するたびにサーバー上の Java プログラムを再作成する必要がなくなるからです。

IBM Toolbox for Java の V4R2 または V4R3 バージョンを使用する場合、jt400.zip ファイルまたは jt400.jar ファイルに対して CRTJVAPGM コマンドを実行することはできません。 このファイルのサイズが大き過ぎるからです。 ただし、jt400Access.zip ファイルに対しては実行できる場合があります。 V4R3 では、IBM Toolbox for Java ライセンス・プログラムに追加ファイル jt400Access.zip が含まれています。 jt400Access.zip には、ビジュアル・クラスではなく、アクセス・クラスだけが含まれています。

V4R5 (またはそれ以前のバージョン) のシステムで Java アプリケーションを実行する際には、 jt400Access.zip を使用してください。 V5R1 システムで Java アプリケーションを実行する際には、 jt400Native.jar を使用してください。

jt400Native.jar に対しては、すでに CRTJVAPGM コマンドが実行されています。

Java 各国語サポート

Java は一連の各国語をサポートしていますが、それはサーバーがサポートしている言語の一部に過ぎません。

ローカル・ワークステーションで使用されている言語が Java によってサポートされない場合など、 言語間のミスマッチが発生すると、IBM Toolbox for Java ライセンス・プログラムは、 英語のエラー・メッセージを発行することがあります。

IBM Toolbox for Java のサービスおよびサポート

サービスおよびサポートを得るためには、以下の情報源をご利用ください。

Toolbox for Java のトラブルシューティング情報
Toolbox for Java 使用時の問題解決の助けとしてこの情報を使用します。

JTOpen/Toolbox for Java フォーラム
Toolbox for Java を使用する Java プログラマーのコミュニティーに参加します。 このフォーラムは、他の Java プログラマーおよび Toolbox for Java の開発者自身から援助やアドバイスを受ける上で役立ちます。

サーバー・サポート
IBM サーバー・サポート Web サイトを使用して、 iSeries サーバーの技術計画およびサポートを能率的に行う助けとなるツールおよびリソースについて調べます。

ソフトウェア・サポート
IBM ソフトウェア・サポート・サービス Web サイトを使用して、IBM が提供する、広範囲にわたるソフトウェア・サポート・サービスを見つけます。

IBM Toolbox for Java (5722-JC1) のサポート・サービスは、 iSeries ソフトウェア・プロダクトの通常のご使用条件のもとで提供されます。 サポート・サービスには、プログラム・サービス、音声サポート、 およびコンサルティング・サービスが含まれています。 詳細については、IBM 担当員にお問い合わせください。

IBM Toolbox for Java プログラムの障害の解決については、プログラム・サービスまたは音声サポートでサポートされ、 アプリケーション・プログラミング/デバッグ問題の解決については、コンサルティング・サービスでサポートされています。

IBM Toolbox for Java アプリケーション・プログラム・インターフェース (API) 呼び出しについては、 次のいずれかに該当する場合を除き、コンサルティング・サービスでサポートされています。

- 比較的単純なプログラムでも再現され、 明らかに Java API の障害である場合。
- 資料情報の詳細に関する質問。

- サンプルや資料がある場所に関する質問。

IBM Toolbox for Java ライセンス・プログラムに添付されているサンプル・プログラムを含め、 プログラミングに関するすべての支援は、コンサルティング・サービスでサポートされています。 追加のサンプルは、インターネットの iSeries ホーム・ページ で入手することができます。 ただし、これらのサンプルはサポートの対象ではありません。

IBM Toolbox for Java ライセンス・プログラム・プロダクトには、問題解決情報が添付されています。 IBM Toolbox for Java に障害の可能性があると思われる場合には、 そのエラーを再現する簡単なプログラムが必要となります。

IBM Toolbox for Java の関連情報

以下のリストには、IBM Toolbox for Java の情報に関連した Web サイトおよび Information Center のトピックが含まれています。

IBM Toolbox for Java のリソース

IBM Toolbox for Java についてさらに学ぶには、以下のサイトを利用してください。

- IBM Toolbox for Java および JTOpen: Service Pack、パフォーマンス上のヒント、例、その他多くの情報が提供されています。 また、この情報の ZIP パッケージ (javadoc を含む) をダウンロードすることもできます。
- IBM Toolbox for Java よく尋ねられる質問 (FAQ): パフォーマンス、トラブルシューティング、JDBC およびその他に関する質問の答えが提供されています。
- IBM Toolbox for Java および JTOpen フォーラム: Toolbox for Java を使用する Java プログラマー および Toolbox for Java の開発者自身のコミュニティーとのコミュニケーションを諮るための、 効果的な方法が提供されています。

» IBM Toolbox for Java 2 Micro Edition のリソース

ToolboxME for iSeries およびワイヤレス・テクノロジーの Java インプリメンテーションについてさらに学ぶには、 以下のサイトを利用してください。

- IBM Toolbox for Java および JTOpen: ToolboxME for iSeries についての詳細情報が提供されています。
- IBM alphaWorks Wireless: ダウンロードおよび開発リソースへのリンクを含めた新しいワイヤレス・テクノロジーについての情報が提供されています。
- Sun Java 2 Platform, Micro Edition: 以下の点を含めた Java ワイヤレス・テクノロジーについての追加情報が提供されています。
  - K Virtual Machine (KVM)
  - Connected Limited Device Configuration (CLDC)
  - Mobile Information Device Profile (MIDP)
- Java Wireless Developer: Java ワイヤレス・アプリケーション開発者のための広範な技術情報が提供されています。

- ワイヤレス・アプリケーション開発ツール:
  - IBM WebSphere Studio Device Developer
  - Java 2 Platform Micro Edition, Wireless Toolkit

Java

Java は、移植可能なオブジェクト指向のアプリケーションおよびアプレットの開発を可能にするプログラミング言語です。 Java についてさらに学ぶには、以下のサイトを利用してください。

- IBM developerWorks Java テクノロジー・ゾーン: ここで提供されている情報、教育、およびツールは、Java、IBM 製品、 およびビジネス・ソリューションを作り出すためのその他のテクノロジーを使用する上で助けになります。
- IBM alphaWorks Java: ダウンロードおよび開発リソースへのリンクを含めた新しい Java テクノロジーについての情報が提供されています。
- Sun Microsystems の "The Source for Java Technology": 最新のテクノロジーを含め、Java のさまざまな使用法に関する情報が提供されています。
- Java for iSeries, PartnerWorld for Developers: Java に関する情報と、IBM のビジネス・パートナーが Java をどのように使用でき、 また、どのように使用しているかが示されています。

Java Naming and Directory Interface

- Java Naming and Directory Interface(TM) (JNDI): JNDI の概要、技術情報、例、および利用可能なサービス・プロバイダーのリストが提供されています。
- iSeries 400 Directory Services (LDAP): OS/400 上の LDAP (Lightweight Directory Access Protocol) に関する情報が提供されています。

» Java Secure Socket Extension

- Java Secure Socket Extension (JSSE): JSSE の簡単な概要と、詳細情報を調べるためのリンクが示されています。 «

サーブレット

サーブレットは、サーバー上で実行され、 1 つ以上のクライアント (それぞれはブラウザーで実行される) から、 1 つ以上のデータベースへの要求を媒介する小さいプログラムです。 サーブレットは Java でプログラミングされるため、 要求を単一プロセスの内部でマルチスレッドとして実行することが可能になり、 それによりシステム・リソースを節約することができます。 サーブレットについてさらに学ぶには、以下のサイトを利用してください。

- IBM Websphere, PartnerWorld for Developers: サーブレット・ベースの Web アプリケーション・サーバーについての情報が提供されています。
- Java Servlet technology: サーブレットを理解し、使用する上で助けになる技術情報、説明、およびツールが提供されています。

XHTML

XHTML は、HTML 4.0 の後継言語として提唱されています。 これは HTML 4.0 に基づいていますが、XML のような拡張性が組み込まれています。 XHTML についてさらに学ぶには、以下のサイトを利用してください。

- The Web Developer's Virtual Library: 例および追加情報へのリンクを含む XHTML の概要が提供されています。
- W3C: XHTML 規格および XHTML 勧告についての技術情報が提供されています。

XML

Extensible Markup Language (XML) は、人とコンピューターの両方にとって容易に理解できる方法で情報を記述し、 編成することを可能にするメタ言語です。 メタ言語では、文書マークアップ言語とその構造を定義することができます。 XML についてさらに学ぶには、以下のサイトを利用してください。

- IBM developerWorks XML zone: IBM が XML に関して行っている作業、および XML が e-commerce を促進する方法についての専用サイトが提供されています。
- IBM alphaWorks XML: ダウンロードおよび開発リソースへのリンクを含めた、 XML 規格および XML ツールの登場についての情報が提供されています。
- XML Support on iSeries, PartnerWorld for Developers: XML に関する情報と、IBM のビジネス・パートナーが XML をどのように使用でき、 また、どのように使用しているかが示されています。

- W3C XML: XML 開発者のためのテクニカル・リソースが提供されています。
- XML.com: コンピューター産業での XML に関する最新情報が提供されています。
- XML.org: 業界ニュース、イベント予定表などを含む XML コミュニティーに関するニュースおよび情報が提供されています。
- XMLephant: XML を学習するためのリソースが提供されています (その他の多くの XML のサイトへのリンクを含む)。
- XML Cover Pages: XML、SGML、および XSL や XSLT のような関連する XML 規格についての広範囲のオンライン解説書が提供されています。

その他の情報

- IBM HTTP Server for iSeries: IBM HTTP Server for iSeries に関する情報、リソース、およびヒントが提供されています。
- iSeries Access for Windows: ダウンロード、FAQ、および付加的なサイトへのリンクを含む iSeries Access for Windows についての情報が提供されています。
- IBM WebSphere Host On-Demand: S/390、iSeries、および DEC/Unix の各エミュレーションをサポートする、 ブラウザー・ベースのエミュレーターについての情報が提供されています。
- IBM Support and downloads: IBM ハードウェア・サポートおよびソフトウェア・サポートを提供しているポータル・サイトです。

コードの特記事項情報

本書には、プログラミング・サンプルが含まれています。

IBM は、お客様に、すべてのプログラミング・コード・サンプルを使用することができる非独占的な使用権を許諾します。お客様は、このサンプル・コードから、お客様独自の特別のニーズに合わせた類似のプログラムを作成することができます。

すべてのサンプル・コードは、例として示す目的でのみ、IBM により提供されます。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。したがって 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。

ここに含まれるすべてのプログラムは、現存するままの状態で提供され、いかなる保証条件も適用されません。第三者の権利の不侵害、商品性、特定目的適合性に関する黙示の保証の適用も一切ありません。