IBM

IBM i
プログラミング
IBM Toolbox for Java

7.1

IBM i

プログラミング
IBM Toolbox for Java

7.1

ご注意!

本書および本書で紹介する製品をご使用になる前に、 813 ページの『特記事項』に記載されている情報をお読みください。

本書は、IBM i 7.1 (製品番号 5770-SS1)、および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。このバージョンは、すべての RISC モデルで稼働するとは限りません。また CISC モデルでは稼働しません。

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

原典： IBM i
Programming
IBM Toolbox for Java
7.1

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

第1刷 2010.4

目次

IBM Toolbox for Java 1
IBM i 7.1 の新機能 1
IBM Toolbox for Java の PDF ファイル 2
IBM Toolbox for Java のインストールと管理 . . . 3
IBM Toolbox for Java インストールの管理 . . . 3
IBM Toolbox for Java のインストール. 4
システム・プロパティー 14
IBM Toolbox for Java クラス 24
アクセス・クラス 24
Commtrace クラス. 196
HTML クラス 205
ReportWriter クラス 240
リソース・クラス 242
セキュリティー・クラス. 245
サーブレット・クラス 249
ユーティリティー・クラス 257
Vaccess クラス 268
Graphical Toolbox および PDML 315
Graphical Toolbox の設定 321
ユーザー・インターフェースの作成 323
実行時のパネルの表示 327
GUI ビルダーによって生成されたヘルプ文書の編集 331
ブラウザーで Graphical Toolbox を使用する . . 335
GUI ビルダーの「パネル・ビルダー」ツールバー 339
IBM Toolbox for Java Bean. 342
JDBC 342
IBM Toolbox for Java JDBC support for IBM i 7.1 に対する機能拡張. 343
IBM Toolbox for Java JDBC support for IBM i 6.1 に対する機能拡張. 347
IBM Toolbox for Java JDBC support for IBM i 5.4 に対する機能拡張. 351
IBM Toolbox for JavaJDBC プロパティー . . . 356
JDBC SQL タイプ. 376
Proxy サポート. 377
Secure Sockets Layer および Java Secure Socket Extension 382
IBM Toolbox for Java 2 Micro Edition 383
ToolboxME の要件. 383
ToolboxME のダウンロードとセットアップ . . 383
ToolboxME を使用する上で重要な概念 384
ToolboxME クラス. 385
ToolboxME プログラムを作成して実行する . . 398
ToolboxME for iSeries の作業例 412
Extensible Markup Language のコンポーネント . . 413
プログラム呼び出しマークアップ言語 413
Graphical Toolbox および PDML 438
レコード・フォーマット・マークアップ言語 . . 443
XML パーサーおよび XSLT プロセッサー . . 454
Extensible Program Call Markup Language . . . 455
よく尋ねられる質問 (FAQ) 483
プログラミングに関するヒント 484
Java プログラムのシャットダウン 484
サーバー・オブジェクトの統合ファイル・システム・パス名 484
Java プログラムの接続の管理 486
IBM i Java 仮想マシン 493
独立補助記憶域プール (ASP) 498
例外 498
エラー・イベント 499
Trace クラス 501
IBM i 最適化 503
パフォーマンスの向上 505
クライアント・インストールおよびクラスの更新 507
AS400ToolboxJarMaker 508
Java 各国語サポート 509
IBM Toolbox for Java のサービスおよびサポート 509
コード例 510
例: アクセス・クラス 511
例: JavaBeans 599
例: Commtrace クラス 607
Graphical Toolbox の例 607
HTML クラスの例. 643
例: プログラム呼び出しマークアップ言語 (PCML) 667
例: ReportWriter クラス 676
例: リソース・クラス 693
例: RFML 697
例: プロファイル・トークン信任状を使用して IBM i スレッド ID を交換する 699
サーブレット・クラスの例 700
簡単なプログラミング例. 727
例: プログラミングに関するヒント 745
例: ToolboxME 746
例: ユーティリティー・クラス 766
例: Vaccess クラス 767
例: XPCML 798
IBM Toolbox for Java の関連情報 808

付録. 特記事項 813
プログラミング・インターフェース情報 815
商標 815
使用条件 815

IBM Toolbox for Java

IBM® Toolbox for Java™ は、Java プログラムを使用してシステム上のデータにアクセスするための、Java クラスのセットです。これらのクラスを使用して、システム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、IBM i 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 IBM i ホスト・サーバーを使用します。 IBM Toolbox for Java は Java に組み込まれた通信機能を使用するので、 IBM Toolbox for Java を使うために IBM i Access 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。

注: このコード例を使用することによって、お客様は 811 ページの『コードに関するライセンス情報および特記事項』 の条件に同意されたものとします。

IBM i 7.1 の新機能

IBM Toolbox for Java のトピック集に関する新情報や重要な変更情報についてお読みください。

IBM i 7.1 では、IBM Toolbox for Java が次のように変更されました。

- ライセンス・プログラム製品 JC1 (IBM Toolbox for Java) は、IBM i 7.1 の中に含まれなくなりました。このソフトウェア製品は SS1 (製品 ID 5770-SS1) オプション 3 の中に統合されました。これまでと同じ統合ファイル・システム・ディレクトリー内に同じ Java JAR ファイルが引き続き提供されますが、今後、これらはすべて 5770-SS1 の一部としてインストールされます。
- 次のクラスが、IBM i 以降追加されました。注のない限り、ここにリストしたすべてのクラスは、パッケージ "com.ibm.as400.access," に入っています。
  - AS400JDBCArray クラス
  - AS400JDBCArrayResultSet クラス
  - ErrorCodeParameter クラス
  - ObjectLockListEntry クラス
  - UserObjectsOwnedList クラス
  - UserObjectsOwnedListEntry クラス
  - com.ibm.as400.security.auth.ProfileTokenProvider インターフェース
  - com.ibm.as400.security.auth.DefaultProfileTokenProvider クラス
- 次のクラスで、大幅な拡張がありました。注のない限り、ここにリストしたすべてのクラスは、パッケージ "com.ibm.as400.access," に入っています。
  - 多数の JDBC クラス
  - CommandCall および ProgramCall - 新しいデフォルトのスレッド・セーフティー動作、新しいメソッド
  - IFSFile および IFSJavaFile - 新しいメソッド
  - AS400 - 新しいシステム・プロパティー、新しいメソッド
  - AS400ConnectionPool

- Trace
- DataArea
- SpooledFile

IBM Toolbox for Java の入手方法

IBM Toolbox for Java は、次のようなフォームで入手可能です。

- IBM Toolbox for Java JAR ファイルは、IBM i の統合ファイル・システム内のディレクトリー `/QIBM/ProdData/OS400/jt400/` の下にインストールされます。
- IBM Toolbox for Java は、オープン・ソース・バージョンでも入手できます。JTOpen Web サイトからコードをダウンロードし、詳細情報を入手することができます。

新規情報または変更情報の見分け方

技術上の変更が加えられた場所を見分けるのに役立つように、Information Center では以下のイメージを使用しています。

- » イメージにより、新規または変更された情報の開始点を示します。
- « イメージにより、新規または変更された情報の終了点を示します。

PDF ファイルでは、左マージンに新規および変更情報のリビジョン・バー (|) があります。

このリリースでの新機能または変更点についてのその他の情報は、「プログラム資料説明書」を参照してください。

IBM Toolbox for Java の PDF ファイル

この情報の PDF ファイルを表示または印刷できます。

本書の PDF 版を表示またはダウンロードするには、IBM Toolbox for Java を選択します。

以下の関連トピックの PDF を表示またはダウンロードすることができます。

- IBM Developer Kit for Java
- System i® デバッガー

その他の情報

Javadocs などを示した IBM Toolbox for Java トピックの zip パッケージは、IBM Toolbox for JavaおよびJTOpen の Web サイトからダウンロードすることができます。

注: ZIP 圧縮されたパッケージ内の情報は、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、その場合、そのリンクは動作しません。

PDF ファイルの保存

表示または印刷のために PDF をワークステーションに保存するには、以下のようにします。

1. ご使用のブラウザーで PDF リンクを右クリックする。
2. PDF をローカルに保存するオプションをクリックする。
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® Reader のダウンロード

これらの PDF を表示または印刷するには、Adobe Reader がご使用のシステムにインストールされている必要があります。このアプリケーションは、 Adobe Web サイト (www.adobe.com/products/acrobat/readstep.html) から無償でダウンロードできます。

IBM Toolbox for Java のインストールと管理

IBM Toolbox for Java を使用すると、システムのリソース、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。

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 IBM i システムです。 IBM i サーバーあるいは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、IBM 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。

IBM 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 IBM Toolbox for Java の個別管理
- ネットワークを使用して 1 台のサーバー上に IBM Toolbox for Java を単一共有形態でインストールして管理する、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 IBM 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 IBM Toolbox for Java をインストールすることの主な利点は、 IBM 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IBM 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。

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 IBM 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 IBM 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 IBM 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。 IBM i に組み込まれた NetServer を使用するか、あるいは IBM i Access for Windowsなどの、システム上のファイルへのアクセスを可能にする別の方法を使用することができます。

IBM Toolbox for Java のインストール

IBM Toolbox for Java 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。 IBM Toolbox for Java をインストールするには、これらのトピックを使用してください。

IBM Toolbox for Java での IBM i の要件

このトピックでは、IBM Toolbox for Java を使用するために、環境が満たす必要のある要件を詳述します。

注: IBM Toolbox for Java を使用する前に、環境に関係するワークステーション要件を必ず確認してください。

必要な IBM i 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 を実行する必要があります。

- 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。 IBM i ホスト・サーバー・オプションは、IBM i 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。
- IBM i 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 IBM i オプションの開始

コマンド行から、以下の手順に従って、必要な IBM i 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. IBM i 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 がシステムにインストール済みかどうかを判別する:

IBM Toolbox for Java がすでに IBM i サーバー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。

1. System i Navigatorで、使用したいシステムを選択し、サインオンします。
2. 「**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。
3. 「**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。

4. 「**詳細**」ペイン (右側のペイン) で、5770-SS1 製品オプション 3 がインストールされているかどうか確認します。この製品とオプションの組み合わせが見つかった場合、IBM Toolbox for Java は選択したサーバーにインストール済みです。

   注: IBM 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。

IBM Toolbox for Java がインストール済みでない場合は、5770-SS1 製品のオプション 3 をインストールすることにより、IBM Toolbox for Java をインストールできます。

QUSER プロファイルの検査:

IBM i 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、クライアント/サーバー環境で IBMToolbox for Java を実行するには、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. 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 IBM i 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)` と入力し、「**ENTER**」 を押す。
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、IBM i 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。

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 i ライセンス・プログラム(5770-DG1)
- IBM i オプション 34 (デジタル証明書マネージャー)

SSL について詳しくは、 382 ページの『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、または SSL を使用するための HTTP サーバー

IBM i サーバーでアプレット、サーブレット、または SSL を使用する場合は、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する必要があります。 IBM HTTP Server for i の詳細については、HTTP Server  の Web サイトを参照してください。

デジタル証明書マネージャーについて、および IBM HTTP Server を使用してデジタル証明書を作成および処理する方法については、デ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 IBM i 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 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 3 を使用して、IBM i サーバーのデータとリソースにアクセスするには、**接続しているサーバー**は、次のうちの 1 つを実行している必要があります。

- IBM i 7.1
- IBM i 6.1
- IBM i 5.4

以下の表は、さまざまなバージョンの IBM i に接続するための互換性の要件を示しています。

注: IBM Toolbox for Java は、上位互換性をサポートしていません。当該バージョンより新しいバージョンの IBM i を実行しているサーバーの場合は、IBM Toolbox for Java をインストールしたり、サーバーに接続したりできません。例えば、IBM i 5.2 に付属の IBM Toolbox for Java バージョンを使用している場合、それを IBM i 5.4 を実行しているサーバーに接続できません。

LPP	IBM i の付属バージョン	IBM i への接続
5722-JC1	5.3	5.1 以降
5722-JC1	5.4	5.2 以降
5761-JC1	6.1	5.3 以降
5770-SS1	7.1	5.4 以降

IBM i 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、 IBM i 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、IBM i JVM 上での実行時の IBM Toolbox for Java の操作にのみ影響します。

IBM i で稼働する際の IBM Toolbox for Java の最適化は、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。
- IBM i 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・ファイルが同じサーバー上にあるときにネイティブ・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。

パフォーマンスを向上させるためには、 IBM i 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 i ネイティブ最適化が組み込まれている jar ファイルを使用しない場合でも、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稼働します。

ToolboxME 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、IBM Toolbox for Java 2 Micro Edition 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。

Toolbox ME 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 (jt400Micro.jar) は、いわゆる JTOpen という、Toolbox for Java のオープン・ソース・バージョンに含まれています。 JTOpen に組み込まれている ToolboxME 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 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 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。

- サポートされているバージョンの Java Standard Edition
- 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 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 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 クラス。 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 IBM i 要件

IBM Toolbox for Java のためのワークステーション要件

ワークステーションが以下の要件を満たしていることを確認してください。

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

IBM 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:

IBM 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。

- サポートされている Java 2 Standard Edition (J2SE) Java 仮想マシンを使用することが推奨されている。 IBM Toolbox for Java の多くの新しい機能は、バージョン 1.4 以降の JVM を使用することが必須となっています。
- vaccess クラスまたは Graphical Toolbox を使用する場合には Swing が必要である。 Swing は J2SE に付属しています。 Sun 社の Java Foundation Classes Web サイトから Swing 1.1 をダウンロードすることもできます。以下の環境がテスト済みです。
  - Windows 2000
  - Windows XP
  - AIX® バージョン 4.3.3.1
  - Sun Solaris バージョン 5.7
  - IBM i 5.4 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:

IBM 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。

  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。IBM 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) プラグインなどの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
- IBM i 5.4 またはこれ以降を実行中のサーバーにワークステーションが接続されている

ToolboxME 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、IBM Toolbox for Java 2 Micro Edition 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。

Toolbox ME 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 (jt400Micro.jar) は、いわゆる JTOpen という、Toolbox for Java のオープン・ソース・バージョンに含まれています。 JTOpen に組み込まれている ToolboxME 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 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 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。

- サポートされているバージョンの Java Standard Edition
- 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 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 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 クラス。 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 IBM i 要件

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、Swing 1.1 クラスを Sun Microsystems, Inc からダウンロードできます。 

IBM Toolbox for Java をシステムにインストールする

システムまたはシステムの区画をクライアントとして構成してあるときにのみ、 IBM Toolbox for Java を IBM i サーバーにインストールする必要があります。

IBM Toolbox for Java をインストールする前に、ご使用の IBM i のバージョンが、IBM 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。サーバーに IBM Toolbox for Java がすでにインストール済みかどうかを判別することをお勧めします。

IBM Toolbox for Java のインストール

5770-SS1 オプション 3 をインストールすることにより、IBM Toolbox for Java をインストールできます。 System i Navigatorまたはコマンド行を使用できます。

System i Navigatorを使用した IBM Toolbox for Java のインストール

System i Navigator を使用して IBM 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. System i Navigator 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、 IBM 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. 「**コマンドの実行**」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: 5770-SS1
   - デバイス: デバイス名あるいは保管ファイル
   - 復元される任意選択部分: 3

   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。

System i Navigator 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 IBM Toolbox for Java のインストール

コマンド行から IBM 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. 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. 「**5770-SS1 3 拡張基本ディレクトリー・サポート**」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

IBM 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。

- IBM 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 IBM 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 i5/OS® NetServer 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: IBM Toolbox for Java のクラスをアプリケーションで使用するときには、システムが IBM i の要件を満たしていることが必要です。

IBM 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 つ以上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 IBM 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。
   - IBM i 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、IBM 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 Web サイト  を参照してください。

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、関連した機能またはパッケージを使用するのにどの JAR ファイルを CLASSPATH に追加しなければならないかを示しています。

IBM 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	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
265 ページの『CommandHelpRetriever 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 XML パーサーおよび XSLT プロセッサー注 3
CommandPrompter 注 3	jt400.jar、jui400.jar、util400.jar 注 4、および XML パーサー 注 3
Commtrace クラス	jt400.jar (クライアント) または jt400Native.jar (サーバー) 注 1
HTML クラス	jt400.jar 注 1 さらに jt400Servlet.jar (クライアント) または jt400Native.jar (サーバー) 注 1
HTMLDocument クラス	HTML クラスに必要な同じ JAR ファイル、さらに XML パーサーおよび XSLT プロセッサー注 3
JCA クラス	jt400.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	jt400.jar (クライアント) 注 1 および jui400.jar注 5
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 6
PCML (開発およびランタイム、構文解析済みのもの) 注 7	jt400.jar (クライアント) または jt400Native.jar (サーバー) 注 1、 注 8、および XML パーサー 注 3
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 8
PDML (開発) 注 3	uitools.jar、jui400.jar、util400.jar 注 4、および XML パーサー 注 3
PDML (ランタイム、構文解析済みのもの) 注 3	jui400.jar、util400.jar 注 4、および XML パーサー 注 3
PDML (ランタイム、逐次化されたもの) 注 3	jui400.jar、および util400.jar 注 4
ReportWriter クラス	jt400.jar (クライアント) または jt400Native.jar (サーバー) 注 1、reportwriter の JAR ファイル 注 9、および XML パーサーおよび XSLT プロセッサー 注 3
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 XML パーサー 注 3
セキュリティー・クラス	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 さらに jt400Servlet.jar (クライアント) または jt400Native.jar (サーバー) 注 1
System i デバッガー 注 4	jt400.jar (クライアント) 注 1 および tes.jar
383 ページの『IBM Toolbox for Java 2 Micro Edition』	jt400Micro.jar (クライアント) 注 10 および jt400.jar (サーバー) または jt400Native.jar (サーバー)注 1
Vaccess クラス	jt400.jar (クライアント) 注 1

IBM 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
XPCML	jt400.jar (クライアント) または jt400Native.jar (サーバー) [注 1]、および XML パーサーおよび XSLT プロセッサー [注 3]

注 1: jt400.jar および jt400Native.jar を両方とも CLASSPATH に入れないでください。環境に最もあった JAR ファイルを選択し、CLASSPATH ではその JAR だけを使用します。

注 2: 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、commtrace、JCA、JDBC サポート、MEServer、PCML、resource、RFML、security、utilities、vaccess、および XPCML。
- **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** - 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。 jtAccess400.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、HTML、MEServer、PCML、resource、RFML、security、XPCML、および ネイティブ最適化。ネイティブ最適化は、IBM i JVM での実行時に IBM i 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、 IBM i 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 IBM i 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** - jt400Native11x.jar の代わりに、jt400Native.jar を使用してください。 jt400Native11x.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 3: XML パーサーまたは XSLT プロセッサーを使用する必要がある場合、それらが JAXP 準拠であることを確認してください。詳細は、以下のページを参照してください。

454 ページの『XML パーサーおよび XSLT プロセッサー』

注 4: CommandPrompter、PDML、または System i デバッガーを使用するには、 IBM Toolbox for Java の一部ではない追加の JAR ファイル、jhall.jar も必要とします。jhall.jar のダウンロードの詳細は、Sun JavaHelp Web サイト  を参照してください。

注 5: util400.jar には、入力を形式設定するためとコマンド行プロンプターを使うための IBM i 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 6: 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 7: 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システム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 5770-SS1 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 8: 開発時に PCML ファイルを直列化すると、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。詳しくは、414 ページの『PCML を使用して IBM i プログラム呼び出しを作成する』を参照してください。

注 9: 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 10: 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

アプリケーションが PCL データを IBM i 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 11: 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。

例えば、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。

  例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。

  例えば、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。

  例えば、 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』
- 17 ページの『CommandCall/ProgramCall プロパティー』
- 17 ページの『FTP プロパティー』
- 18 ページの『接続プロパティー』

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.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。

トレース・プロパティー	説明
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()` 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()` 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。

FTP プロパティー

FTP プロパティー	説明
com.ibm.as400.access.FTP.reuseSocket	「アクティブ」モードにおいて、(単一の FTP インスタンスにより) 複数のファイル転送でソケットを再利用するかどうかを指定します。 `true` の場合、ソケットは再利用されます。 `false` の場合、ファイル転送ごとに新しいソケットが作成されます。ある FTP オブジェクトに対して `FTP.setReuseSocket()` が実行された場合、そのオブジェクトについて、このプロパティーは無視されます。

接続プロパティー

接続プロパティー	説明
com.ibm.as400.access.AS400.guiAvailable	現在の環境に GUI 機能があるかどうかを示します。 true の場合、エラー状態の表示、追加情報の要求、またはパスワード変更要求のプロンプトがサインオン中に出される可能性があります。 false の場合、エラー状態または欠落情報の結果として例外が発生します。 IBM i アプリケーションとして稼働しているアプリケーション、またはサインオン・ユーザー・インターフェースを制御する必要があるアプリケーションでは、プロンプト・モードを false に設定して実行するのが適切な場合があります。デフォルトは true です。
com.ibm.as400.access.AS400.mustAddLanguageLibrary	システムでの実行時に、AS400 オブジェクトが適切な 2 次言語ライブラリーをライブラリー・リストに追加しようと試みるかどうかを示します。言語ライブラリーを設定すると、戻されるすべてのシステム・エラー・メッセージは、クライアント・ロケールに適した各国語で必ず戻されるようになります。 true の場合、システムでの実行時に AS400 オブジェクトは 2 次言語ライブラリーをライブラリー・リストに追加しようと試みます。 false の場合、AS400 オブジェクトは 2 次言語ライブラリーをライブラリー・リストに追加しようと試みません。デフォルトは false です。
com.ibm.as400.access.AS400.mustUseNetSockets	AS400 オブジェクトがインターネット・ドメイン・ソケットだけを使用するかどうかを指定します。 Java プログラムがシステムで実行されるとき、いくつかの IBM Toolbox for Java クラスは UNIX ドメイン・ソケット接続を作成します。 true の場合、インターネット・ドメイン・ソケットだけが AS400 オブジェクトによって使用されます。 false の場合、AS400 オブジェクトはインターネット・ドメイン・ソケットに加えて UNIX ドメイン・ソケット接続を使用することができます。デフォルトは false です。
com.ibm.as400.access.AS400.mustUseSockets	AS400 オブジェクトがソケットを使用するかどうかを指定します。 Java プログラムがシステムで実行されるとき、いくつかの IBM Toolbox for Java クラスは、システムへのソケット呼び出しではなく API 呼び出しによってデータにアクセスします。ソケット呼び出しの代わりに API 呼び出しが使用されるときには、クラスの動作が少し異なります。このような違いによってプログラムが影響を受ける場合、このプロパティーを true に設定することで、クラスが API 呼び出しの代わりにソケット呼び出しを使用するよう強制できます。デフォルトは false です。

接続プロパティー	説明
com.ibm.as400.access.AS400.mustUseSuppliedProfile	提供されているプロファイルだけを AS400 オブジェクトで使用するかどうかを指定します。 Java プログラムがシステムで実行されるとき、現在サインオンしているユーザー・プロファイル (*CURRENT) の情報を使用できます。 true の場合、ユーザー・プロファイルが提供される必要があります。 false の場合、ユーザー・プロファイルが提供されなければ、AS400 オブジェクトは現在のユーザー・プロファイル情報を取得します。デフォルトは false です。
com.ibm.as400.access.AS400.signonHandler	デフォルトのサインオン・ハンドラーを指定します。ある AS400 オブジェクトに対して AS400.setSignonHandler() が実行されたか、または AS400.setDefaultSignonHandler() が呼び出された場合、そのオブジェクトについて、このプロパティーは無視されます。
com.ibm.as400.access.AS400.threadUsed	ホスト・サーバーとの通信で AS400 オブジェクトがスレッドを使用するかどうかを指定します。 IBM Toolbox for Java にスレッドを使用させる場合、パフォーマンス上の効果がある可能性があります。アプリケーションが Enterprise Java Beans 仕様に準拠する必要がある場合には、スレッドを無効にする必要がある可能性があります。デフォルトは true です。

例: プロパティー・ファイル

この例は、プロキシー・サーバー、トレース・カテゴリー、コマンド呼び出し、プログラム呼び出し、ファイル転送、および接続のプロパティーを示します。

```
#=========================================================#
# 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 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


#-------------------------------------------------------#
# FTP system properties                                 #
#-------------------------------------------------------#

# This system property specifies whether the socket is reused
# for multiple file transfers (through a single FTP instance),
# when in "active" mode.
# If true, the socket is reused. If false, a new socket is
# created for each file transfer.
# This property is ignored for a given FTP object if
# FTP.setReuseSocket(true/false) has been performed on the object.
com.ibm.as400.access.FTP.reuseSocket=true


#-------------------------------------------------------#
# Connection system properties                          #
#-------------------------------------------------------#

# This system property specifies the default signon handler.
```

```
# This property is ignored for a given AS400 object if
# AS400.setSignonHandler() has been performed on
# the object, or if AS400.setDefaultSignonHandler()
# has been called.
com.ibm.as400.access.AS400.signonHandler=mypackage.MyHandler

# This system property specifies whether the Toolbox should
# assume that the current environment has GUI capability.
# This property is ignored for a given AS400 object if
# AS400.setGuiAvailable() has been performed on the object.
com.ibm.as400.access.AS400.guiAvailable=true

# This system property specifies whether the appropriate
# secondary language library must be added to the library list,
# when running on the system. By default the library is not added.
# This property is ignored for a given AS400 object if
# AS400.setMustAddLanguageLibrary() has been performed
# on the object.
com.ibm.as400.access.AS400.mustAddLanguageLibrary=true

# This system property specifies whether sockets
# must be used when communicating with the system.
# Setting this property to true directs the Toolbox to refrain
# from exploiting native optimizations, when running directly on
# the system.
# By default, when running directly on the system the Toolbox
# will exploit native optimizations if they are available,
# and bypass sockets.
# This property is ignored for a given AS400 object if
# AS400.setMustUseSockets() has been performed on the object.
com.ibm.as400.access.AS400.mustUseSockets=true

# This system property specifies whether Internet domain sockets
# must be used when communicating with the system.
# Setting this property to true directs the Toolbox to refrain
# from exploiting Unix sockets, when running directly on
# the system.
# By default, when running directly on the system the Toolbox
# will exploit Unix sockets if they are available,
# rather than Internet domain sockets.
# This property is ignored for a given AS400 object if
# AS400.setMustUseNetSockets() has been performed on the object.
com.ibm.as400.access.AS400.mustUseNetSockets=true

# This system property specifies whether the explicitly supplied
# profile must be used when communicating with the system.
# Setting this property to true directs the Toolbox to
# refrain from exploiting the currently signed-on profile
# by default, when running directly on the system.
# This property is ignored for a given AS400 object if
# AS400.setMustUseSuppliedProfile() has been performed on the object.
com.ibm.as400.access.AS400.mustUseSuppliedProfile=true

# This system property specifies whether threads are used
# when communicating with the host servers.
# By default, the AS400 object creates separate threads
# to listen on communication sockets to the host servers.
# Setting this property to false directs the Toolbox to refrain
# from creating separate threads for host server communications.
# This property is ignored for a given AS400 object if
# AS400.setThreadUsed() has been performed on the object.
com.ibm.as400.access.AS400.threadUsed=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 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");


/*-----------------------------------------------------------*/
/* FTP system properties                                     */
/*-----------------------------------------------------------*/

// This system property specifies whether the socket is reused
// for multiple file transfers (through a single FTP instance),
// when in "active" mode. If true, the socket is reused.
// If false, a new socket is created for each file transfer.
// This property is ignored for a given FTP object if
// FTP.setReuseSocket(true/false) has been performed on the object.
put ("com.ibm.as400.access.FTP.reuseSocket", "true");


/*-----------------------------------------------------------*/
/* Connection system properties                              */
/*-----------------------------------------------------------*/

// This system property specifies the default signon handler.
// This property is ignored for a given AS400 object if
// AS400.setSignonHandler() has been performed on
// the object, or if AS400.setDefaultSignonHandler()
// has been called.
put ("com.ibm.as400.access.AS400.signonHandler", "mypackage.MyHandler");

| // This system property specifies whether the Toolbox should
| // assume that the current environment has GUI capability.
| // This property is ignored for a given AS400 object if
| // AS400.setGuiAvailable() has been performed on the object.
| put ("com.ibm.as400.access.AS400.guiAvailable", "true");
|
| // This system property specifies whether the appropriate
| // secondary language library must be added to the library list,
| // when running on the system. By default the library is not added.
| // This property is ignored for a given AS400 object if
| // AS400.setMustAddLanguageLibrary() has been performed
| // on the object.
| put ("com.ibm.as400.access.AS400.mustAddLanguageLibrary", "true");
|
| // This system property specifies whether sockets
| // must be used when communicating with the system.
| // By default, when running directly on the system the Toolbox
| // will exploit native optimizations if they are available,
| // and bypass sockets.
| // This property is ignored for a given AS400 object if
| // AS400.setMustUseSockets() has been performed on the object.
| put ("com.ibm.as400.access.AS400.mustUseSockets", "true");
|
| // This system property specifies whether Internet domain sockets
| // must be used when communicating with the system.
| // Setting this property to true directs the Toolbox to refrain
| // from exploiting Unix sockets, when running directly on
```

```
|       // the system.
|       // By default, when running directly on the system the Toolbox
|       // will exploit Unix sockets if they are available,
|       // rather than Internet domain sockets.
|       // This property is ignored for a given AS400 object if
|       // AS400.setMustUseNetSockets() has been performed on the object.
|       put ("com.ibm.as400.access.AS400.mustUseNetSockets", "true");
|
|       // This system property specifies whether the explicitly supplied
|       // profile must be used when communicating with the system.
|       // Setting this property to true directs the Toolbox to
|       // refrain from exploiting the currently signed-on profile
|       // by default, when running directly on the system.
|       // This property is ignored for a given AS400 object if
|       // AS400.setMustUseSuppliedProfile() has been performed on the object.
|       put ("com.ibm.as400.access.AS400.mustUseSuppliedProfile", "true");
|
|       // This system property specifies whether threads are used
|       // when communicating with the host servers.
|       // By default, the AS400 object creates separate threads
|       // to listen on communication sockets to the host servers.
|       // Setting this property to false directs the Toolbox to refrain
|       // from creating separate threads for host server communications.
|       // This property is ignored for a given AS400 object if
|       // AS400.setThreadUsed() has been performed on the object.
|       put ("com.ibm.as400.access.AS400.threadUsed", "true");
    }
}
```

IBM Toolbox for Java クラス

IBM Toolbox for Java クラスは、他のすべての Java クラスと同様に、パッケージにカテゴリー化されます。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。

以下にリストされているパッケージに加えて、micro パッケージの詳細を調べることもできます。これを使用すれば、 383 ページの『IBM Toolbox for Java 2 Micro Edition』のトピックにあるとおり、IBM i データおよびサービスへのワイヤレス装置の直接アクセスを可能にする Java プログラムを作成することができます。

アクセス・クラス

IBM Toolbox for Java アクセス・クラスは、IBM i のデータとリソースを表します。

注: IBM Toolbox for Java は、IBM i 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 IBM i 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。

関連情報

EventLog クラス Javadoc
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との依存関係なしにログ記録するための手段を提供します。

アクセス・パッケージの要約

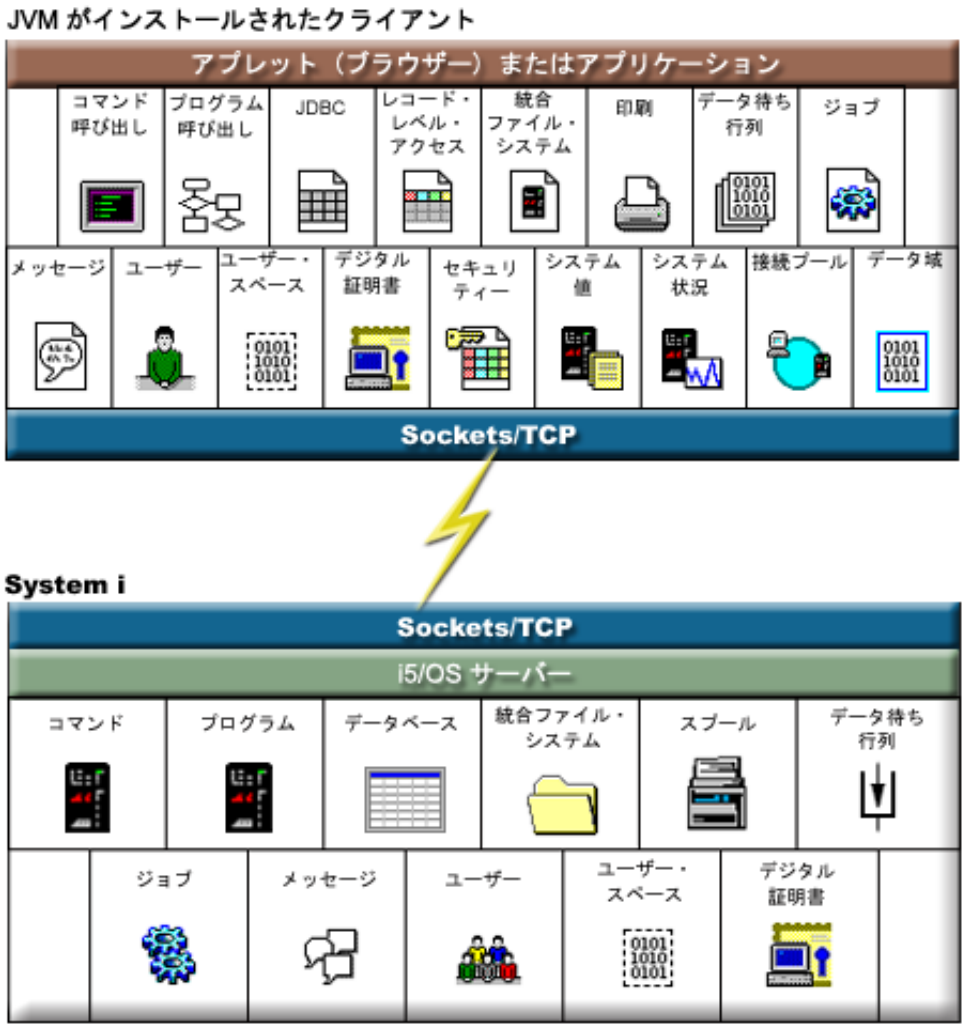
リソース・パッケージの要約

サーバーのアクセス・ポイント

IBM Toolbox for Java の アクセス・クラスは、 IBM i Access for Windows API を使用するのに似た機能を提供します。ただし、IBM i Access for Windows がなくても、これらのクラスを使用することができます。

アクセス・クラスは、既存のシステムをアクセス・ポイントとして使用します。各サーバーはシステム上の個別のジョブで実行し、ソケット接続時にデータ・ストリームを送受信します。

図 1: サーバーのアクセス・ポイント



AS400 クラス

IBM Toolbox for Java AS400 クラスは、サーバー上のサーバー・ジョブへのソケット接続のセットおよびサーバーに対するサインオン動作を管理します。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。

IBM i システムにアクセスするクラスのインスタンスを Java プログラムが使用する場合、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、コマンドをシステムに送信する前に AS400 オブジェクトが必要です。

AS400 オブジェクトが IBM i 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。詳しくは、493 ページの『IBM i 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サーバー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 Javadoc 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・トークンの取得証明書および関連したユーザー・プロファイルの認証
- プロファイル・トークンの設定証明書
- 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
- オペレーティング・システム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。

関連情報

AS400ConnectionPool Javadoc

AS400 Javadoc

デフォルト・ユーザー ID の管理:

ユーザーのサインオン情報を入力する回数を減らすには、デフォルト・ユーザー ID を使用します。 Java プログラムは、プログラムからユーザー ID が提供されない場合に、デフォルト・ユーザー ID を使用します。デフォルト・ユーザー ID は、 Java プログラムまたはユーザー・インターフェースによって設定することができます。デフォルト・ユーザー ID が設定されていない場合、「サインオン」ダイアログを使用してユーザーがデフォルト・ユーザー ID を設定することができます。

一度サーバーに対してデフォルト・ユーザー ID を設定したなら、「サインオン」ダイアログでデフォルト・ユーザー ID を変更することはできません。 AS400 オブジェクトが構成されると、 Java プログラムがユーザー ID およびパスワードを提供できるようになります。プログラムが AS400 オブジェクトにユーザー ID を提供する場合、デフォルト・ユーザー ID には影響しません。デフォルト・ユーザー ID をプログラムが設定または変更しようとする場合、プログラムは明示的にデフォルト・ユーザー ID setUseDefaultUser() を設定しなければなりません。詳細については、『プロンプト、デフォルト・ユーザー ID、およびパスワード・キャッシュの要約』を参照してください。

AS400 オブジェクトには、デフォルトのユーザー ID を取得、 設定、および除去するためのメソッドが備わっています。 Java プログラムは、 setUseDefaultUser() メソッドによってデフォルトのユーザー ID 処理を使用不可にすることもできます。デフォルト・ユーザー ID 処理が使用不可で、 Java アプリケーションがユーザー ID を提供していない場合、 AS400 オブジェクトは、サーバーに接続が行われるたびに、ユーザー ID の入力を要求します。

1 つの Java 仮想マシン内の同じ IBM i システムを表す AS400 オブジェクトはすべて、同じデフォルト・ユーザー ID を使用します。

以下の例では、2 つの AS400 オブジェクトを使用して、サーバーへの 2 種類の接続を作成します。サインオン時に「デフォルト・ユーザー ID (Default User ID)」ボックスをチェックしておくと、 2 度目の接続時にはユーザー ID の入力を求めるプロンプトは出されません。

```
// Create two AS400 objects to the same system.
AS400 sys1 = new AS400("mySystem.myCompany.com");
AS400 sys2 = new AS400("mySystem.myCompany.com");

// Start a connection to the command call service.
// The user is prompted for user ID and password.
sys1.connectService(AS400.COMMAND);

// Start another connection to the command call service.
// The user is not prompted.
sys2.connectService(AS400.COMMAND);
```

サーバーの最新の AS400 オブジェクトがガーベッジ・コレクションの対象になると、デフォルト・ユーザー ID 情報は廃棄されます。

パスワード・キャッシュを使用する:

パスワード・キャッシュを使うと、 IBM Toolbox for Java においてパスワードとユーザー ID 情報を保管することができ、それによって、接続が行われるたびに同じ情報をユーザーに尋ねるプロンプトが出なくなります。

以下のタスクを実行するには、 AS400 オブジェクトが提供するメソッドを使用します。

- パスワード・キャッシュを消去し、パスワード・キャッシュを使用不可にする。
- サインオン情報の入力の回数を最小にする。

パスワード・キャッシュは、 Java 仮想マシン内で特定の IBM i システムを表す AS400 オブジェクトすべてに対して適用されます。 Java では、仮想マシン間での情報の共用が許可されていないため、ある Java 仮想マシンにキャッシュされたパスワードは他の仮想マシンには見えません。最新の AS400 オブジェクトがガーベッジ・コレクションの対象になると、キャッシュは廃棄されます。「サインオン」ダイアログには、パスワードをキャッシュするかどうかを選択するオプションのチェック・ボックスがあります。 AS400 オブジェクトが構成されると、 Java プログラムはユーザー ID およびパスワードを提供することもできるようになります。コンストラクターに提供されたパスワードはキャッシュされません。

AS400 オブジェクトは、 パスワード・キャッシュを消去する (clear the password cache) ためのメソッド、および パスワード・キャッシュを使用不可にする (disable the password cache) ためのメソッドを提供します。詳細については、『プロンプト、デフォルト・ユーザー ID、およびパスワード・キャッシュの要約』を参照してください。

ユーザー ID とパスワードの入力要求 (プロンプト):

AS400 クラスの使用時には、サーバーへの接続時にユーザー ID およびパスワードの入力要求 (プロンプト) が起こる場合があります。入力要求 (プロンプト) は、Java プログラムでオフにすることができます。

Java プログラムで、 AS400 オブジェクトが表示するユーザー ID とパスワードのプロンプト・ウィンドウおよびメッセージ・ウィンドウをオフにすることができます。これが必要になるのは、たとえば、複数のクライアントのためにゲートウェイ上でアプリケーションを実行する場合などです。ゲートウェイ・マシンでプロンプトおよびメッセージが表示されると、ユーザーはプロンプトに対して応答することができません。このようなアプリケーションの場合、AS400 オブジェクトの setGuiAvailable() を使用して、すべてのプロンプトの表示をオフにすることができます。

詳細については、『プロンプト、デフォルト・ユーザー ID、およびパスワード・キャッシュの要約』を参照してください。

プロンプト、デフォルト・ユーザー ID、およびパスワード・キャッシュの要約:

Java プログラムは、ユーザー ID およびパスワードのキャッシュ入力要求をいつ表示するかを制御することができます。「サインオン」ダイアログからの情報を使用して、デフォルト・ユーザー ID の設定およびパスワードのキャッシュを行うことができます。入力要求時に検索される情報、および設定される情報について、次の表に要約しています。

この表では、 Java プログラムでデフォルト・ユーザー ID 処理およびパスワード・キャッシュが可能であり、「サインオン」ダイアログの「**デフォルト・ユーザー ID (Default User ID)**」ボックスと「**パスワードを保存 (Save Password)**」ボックスをチェックしたことを前提としています。

この表は、クライアント接続のために使用し、サーバー上で Java を実行するためには使用しないでください。

コンストラクターにシステムが提供される	コンストラクターにユーザー ID が提供される	コンストラクターにパスワードが提供される	デフォルト・ユーザーが設定される	ユーザー ID 用のパスワードがキャッシュに入れられる	マークされた設定を使用する結果
					システム名、ユーザー ID、およびパスワードのプロンプトが表示される。デフォルト・ユーザー ID が設定され、パスワードがキャッシュされる。

コンストラクターにシステムが提供される	コンストラクターにユーザー ID が提供される	コンストラクターにパスワードが提供される	デフォルト・ユーザーが設定される	ユーザー ID 用のパスワードがキャッシュに入れられる	マークされた設定を使用する結果
Yes					ユーザー ID およびパスワードのプロンプトが表示される。システム名は表示されるが、変更できない。デフォルト・ユーザー ID が設定され、パスワードがキャッシュされる。
Yes	**Yes**				パスワードのプロンプトが表示される。ユーザー ID は、表示され、変更できる。システム名は表示されるが、変更できない。デフォルト・ユーザー ID は変更されない。パスワードはキャッシュされる。
Yes	**Yes**	**Yes**			プロンプトの表示なし。デフォルト・ユーザー ID は変更されない。パスワードはキャッシュされない。
			Yes		システム名とパスワードのプロンプトが表示される。ユーザー ID は、表示され、変更できる。ユーザー ID を変更しても、デフォルト・ユーザー ID は変更されない。パスワードはキャッシュされる。

コンストラクターにシステムが提供される	コンストラクターにユーザー ID が提供される	コンストラクターにパスワードが提供される	デフォルト・ユーザーが設定される	ユーザー ID 用のパスワードがキャッシュに入れられる	マークされた設定を使用する結果
Yes			**Yes**		デフォルト・ユーザー ID のパスワードのプロンプトが表示される。ユーザー ID は、表示され、変更できる。システム名は表示されるが、変更できない。パスワードはキャッシュされる。
Yes			**Yes**	**Yes**	プロンプトの表示なし。デフォルト・ユーザー ID およびキャッシュされたパスワードを使用して接続。
Yes	**Yes**			**Yes**	プロンプトの表示なし。キャッシュされたパスワードを使用して特定のユーザーとして接続。
Yes	**Yes**		**Yes**	**Yes**	プロンプトの表示なし。キャッシュされたパスワードを使用して特定のユーザーとして接続。
Yes	**Yes**	**Yes**	**Yes**		プロンプトの表示なし。指定されたユーザーとして接続。

SecureAS400 クラス

SecureAS400 クラスは、暗号化データの送受信時の AS400 オブジェクトの使用を可能にします。 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サーバー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();
```

関連情報

SecureAS400 Javadoc

AS400 Javadoc

AS400JPing クラス

IBM Toolbox for Java AS400JPing クラスを使用すると、Java プログラムはホスト・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: AS400JPing を使用する

以下の例は、Java プログラム内で AS400JPing を使用してリモート・コマンド・サービスを PING する方法を示しています。

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

関連情報

AS400JPing クラス

BidiTransform クラス

IBM Toolbox for Java BidiTransform クラスは、IBM i 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 IBM i 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

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。
- IBM i 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 IBM i へのデータの変換 (Convert data) を行う。
- IBM i レイアウトから Java へのデータの変換 (Convert data) を行う。

例: AS400BidiTransform を使用する

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 system layout:
AS400BidiTransform abt;
abt = new AS400BidiTransform(424);
String dst = abt.toAS400Layout("some bidirectional string");
```

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

BidiConversionProperties クラス

BidiConversionProperties クラスは、文字セット・データの変換の制御に使用できる一連のプロパティーを提供します。

関連情報

BidiConversionProperties Javadoc

CallStackEntry クラス

CallStackEntry クラスは、サーバー・ジョブの特定スレッドの呼び出しスタック内のエントリーを表します。

このタイプのオブジェクトは、`Job.getCallStack()` を呼び出すことによって生成されます。

関連情報

CallStackEntry Javadoc

ClusteredHashTable クラス

IBM Toolbox for Java 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非永続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、そのデータに非永続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: 以下の解説では、IBM i のクラスター・テクノロジーに共通する概念と用語を理解されていることが前提となっています。詳細は、IBM iクラスター・テクノロジ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

例: ClusteredHashTable を使用する

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。

CommandCall クラス

CommandCall クラスを使用すると、Java プログラムが非対話式の IBM i 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 CommandCall 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 IBM i 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 オブジェクトはシステムに接続します。接続の管理については、接続の管理を参照してください。

Java プログラムと IBM i コマンドが同じサーバー上にある場合、IBM 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

以下の例は、さまざまな種類の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。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

例: コマンドを実行する

以下の例は、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);
```

例: ユーザー指定コマンドを実行する

514 ページの『例: CommandCall を使用する』は、ユーザーによって指定されたコマンドの実行方法を示しています。

関連情報

CommandCall Javadoc

AS400Message Javadoc

AS400 Javadoc

接続プール

接続プールを使用し、接続を共用して IBM i システムへの接続のセット (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 IBM Toolbox for Java サポートの一環として、Java プログラムで使用できる 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改善できます。たとえば、AS400ConnectionPool クラスを使用して、接続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プールが作成されるときに、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 クラスの使用時にシステム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 を使用する

516 ページの『例: AS400ConnectionPool を使用する』 では、AS400 オブジェクトの再使用の方法について説明します。

関連情報

AS400ConnectionPool Javadoc

AS400 Javadoc

データ域

IBM Toolbox for Java DataArea クラスは、IBM i 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。接続の管理については、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 IFS 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 IFS 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

例: DecimalDataArea の使用 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。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
   // Establish a connection to the server "MyServer".
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 IFS 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
- Read
- 書き込み

DataArea Javadoc

CharacterDataArea Javadoc

DecimalDataArea Javadoc

LocalDataArea Javadoc

LogicalDataArea Javadoc

DataAreaEvent Javadoc

DataAreaListener Javadoc

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 IBM i 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 IBM i 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、またはシステム・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。

コンバーター・クラスを使って、Java とご使用の システムの間の変換を高速および効率的に行います。 BinaryConverter は Java バイト配列と Java 単純タイプの間の変換を行います。 CharConverter は Java のストリング・オブジェクトと IBM i のコード・パッケージの間の変換を行います。詳細は、コンバーターのトピックを参照してください。

データ型

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型を使用します。次のようなデータ型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)

例: AS400DataType を使用する

プログラム・パラメーター用のデータの指定や、プログラム・パラメーターに戻されたデータの解釈を行うために、AS400DataType クラスを ProgramCall と一緒に使用した例は、以下のとおりです。

例: ProgramCall と共に AS400DataType クラスを使用する

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型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、IBM i 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 IBM i データが含まれている可能性があります。プログラムは、データ型クラスを使用して、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、IBM i 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型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。
- 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: レコード様式変換クラスを使用する

次の例は、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。

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

FieldDescription、RecordFormat、および Record の各クラスを使用する

AS400DataType Javadoc

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 IBM i 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	システム形式の符号付き 2 バイトの数値データと Java Short オブジェクト間の変換
AS400Bin4	システム形式の符号付き 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	システム形式の符号付き 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	システム形式の符号付き 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	システム形式のパック 10 進の数値データと Java BigDecimal オブジェクト間の変換
AS400UnsignedBin2	システム形式の符号なし 2 バイトの数値データと Java Integer オブジェクト間の変換
AS400UnsignedBin4	システム形式の符号なし 4 バイトの数値データと Java Long オブジェクト間の変換
AS400ZonedDecimal	システム形式のゾーン 10 進の数値データと Java BigDecimal オブジェクト間の変換

例

以下の例では、システム形式の数値データ型および Java int を使用したデータ変換を示します。

例: システム形式から Java int に変換する

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

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

// Convert from system 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();
```

例: Java int からシステム形式に変換する

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

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

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

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

テキスト変換:

文字データは、 IBM Toolbox for Java 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 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 形式のバイト配列を IBM i 形式に変換します。 toObject() メソッドは、 IBM i 形式のバイト配列を Java 形式に変換します。

AS400BidiTransform クラスは、 IBM i 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 IBM i 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、ストリング型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: テキスト・データの変換

次の例は、 DataQueueEntry オブジェクトがテキストを EBCDIC で戻すものとします。この例では、EBCDIC 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 system and the data has been
     // put in the following buffer.
     int textLength = 100;
     byte[] data = new byte[textLength];

     // Create a converter for the system data type.
Note a default
     // converter is being built.  This converter assumes the IBM i
     // 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 an IBM 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 IBM i 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);
```

関連情報

AS400Text Javadoc

AS400BidiTransform Javadoc

BidiStringType Javadoc

複合タイプの変換クラス:

このトピックでは、複合タイプの IBM Toolbox for Java 変換クラスについて説明します。

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

例: 複合データ型の変換

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 IBM i 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();
```

AS400Array Javadoc

AS400Structure Javadoc

FieldDescription クラス:

フィールド記述クラスを使用すると、データ型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、 IBM i 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

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

例: フィールド記述を作成する

次の例は、データ待ち行列内のエントリーは同じ形式であるものとします。各データには、以下のフィールド記述を使用して記述できるメッセージ番号 (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。

```
// 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");
```

RecordFormat クラスのあるインスタンス内にフィールド記述をグループ化することができます。フィールド記述を RecordFormat オブジェクトに追加する方法については、以下のページの例を参照してください。

『RecordFormat クラス』

RecordFormat クラス:

IBM Toolbox for Java 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。

例: フィールド記述をレコード様式に追加する

以下の例では、フィールド記述の例で作成したフィールド記述をレコード様式に追加します。

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

レコード様式からレコードを作成する方法については、以下のページの例を参照してください。

『Record クラス』

RecordFormat Javadoc

Record クラス:

IBM Toolbox for Java 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。
- レコード内にあるフィールドの数を検索する。
- 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。

例: レコードを読み取る

以下の例では、レコード様式の例で作成したレコード様式を使用します。

```
// 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");
```

関連情報

Record Javadoc

フィールドの内容の検索:

Java プログラムで一度に 1 フィールドのデータ、またはすべてのフィールドのデータを取得して、Record オブジェクトの内容を検索します。

名前または索引を使用して単一フィールドを検索するには、レコード・クラスの getField() メソッドを使用します。 getFields() メソッドでは、すべてのフィールドを Object[] として検索します。

Java プログラムは、Object (または Object[] の要素) が、検索したフィールドに対する適切な Java オブジェクトに戻るように型変換しなければなりません。フィールド・タイプに基づいて型変換される、適切な Java オブジェクトを次の表に示します。

フィールド・タイプ (DDS)	フィールド・タイプ (FieldDescription)	Java オブジェクト
BINARY (B)、length <= 4	BinaryFieldDescription	Short
BINARY (B)、length >= 5	BinaryFieldDescription	Integer
CHARACTER (A)	CharacterFieldDescription	String
DBCS Either (E)	DBCSEitherFieldDescription	String
DBCS Graphic (G)	DBCSGraphicFieldDescription	String
DBCS Only (J)	DBCSOnlyFieldDescription	String
DBCS Open (O)	DBCSOpenFieldDescription	String
DATE (L)	DateFieldDescription	String
FLOAT (F)、単精度	FloatFieldDescription	Float
FLOAT (F)、倍精度	FloatFieldDescription	Double
HEXADECIMAL (H)	HexFieldDescription	byte[]
PACKED DECIMAL (P)	PackedDecimalFieldDescription	BigDecimal
TIME (T)	TimeDecimalFieldDescription	String
TIMESTAMP (Z)	TimestampDecimalFieldDescription	String
ZONED DECIMAL (P)	ZonedDecimalFieldDescription	BigDecimal

関連情報

Record Javadoc

フィールドの内容の設定:

Java プログラムで setField() メソッドを使用して、 Record オブジェクトの内容を設定します。

Java プログラムは、設定するフィールドに対して適切な Java オブジェクトを指定しなければなりません。各フィールド・タイプに対する適切な Java オブジェクトを次の表に示します。

フィールド・タイプ (DDS)	フィールド・タイプ (FieldDescription)	Java オブジェクト
BINARY (B)、length <= 4	BinaryFieldDescription	Short
BINARY (B)、length >= 5	BinaryFieldDescription	Integer
CHARACTER (A)	CharacterFieldDescription	String
DBCS Either (E)	DBCSEitherFieldDescription	String
DBCS Graphic (G)	DBCSGraphicFieldDescription	String
DBCS Only (J)	DBCSOnlyFieldDescription	String
DBCS Open (O)	DBCSOpenFieldDescription	String
DATE (L)	DateFieldDescription	String
FLOAT (F)、単精度	FloatFieldDescription	Float
FLOAT (F)、倍精度	FloatFieldDescription	Double
HEXADECIMAL (H)	HexFieldDescription	byte[]
PACKED DECIMAL (P)	PackedDecimalFieldDescription	BigDecimal
TIME (T)	TimeDecimalFieldDescription	String
TIMESTAMP (Z)	TimestampDecimalFieldDescription	String

フィールド・タイプ (DDS)	フィールド・タイプ (FieldDescription)	Java オブジェクト
ZONED DECIMAL (P)	ZonedDecimalFieldDescription	BigDecimal

関連情報

Record Javadoc

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

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

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

以下の例は、LineDataRecordWriter クラスを使用してレコードを書き込む 1 つの方法を示しています。

```
// 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.");
      }
```

LineDataRecordWriter Javadoc

Record Javadoc

RecordFormat Javadoc

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。

IBM i データ待ち行列には、次のような特性があります。

- 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、順次およびキー順データ待ち行列をサポートします。

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。

例: DataQueue および DataQueueEntry を使用する

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。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
// 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);
```

関連情報

BaseDataQueue Javadoc

DataQueue Javadoc

KeyedDataQueue Javadoc

DataQueueEntry Javadoc

KeyedDataQueueEntry Javadoc

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、および待ち行列の記述) 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を処理する

次に示す順次データ待ち行列の例では、作成側が項目をデータ待ち行列に置く方法、および使用側がその待ち行列からその項目を取り出して処理する方法を説明します。

530 ページの『例: DataQueue クラスを使用して待ち行列にデータを書き込む』

533 ページの『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』

BaseDataQueue Javadoc

DataQueue Javadoc

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システム上に作成する。 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、待ち行列を記述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、指定されたキーに基づいて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: キー順データ待ち行列を処理する

次に示すキー順データ待ち行列の例では、作成側が項目をデータ待ち行列に置く方法、および使用側がその待ち行列からその項目を取り出して処理する方法を説明します。

538 ページの『例: KeyedDataQueue を使用する』

541 ページの『例: KeyedDataQueue クラスを使用してデータ待ち行列から項目を読み取る』

BaseDataQueue Javadoc

KeyedDataQueue Javadoc

デジタル証明書

デジタル証明書とは、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、デジタル署名付きのステートメントのことです。

Secure Sockets Layer (SSL) を使って安全な接続を確立するには、デジタル証明書が必要です。

デジタル証明書は、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 には、システム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。

AS400CertificateUserProfileUtil および AS400CertificateVldlUtil を使用するには、基本オペレーティング・システム・オプション 34 (デジタル証明書マネージャー) をインストールする必要があります。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。

例: デジタル証明書をすべてリストする

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。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
// 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
```

AS400CertificateUserProfileUtil Javadoc

AS400CertificateVldUtil Javadoc

AS400CertificateAttribute Javadoc

EnvironmentVariable クラス

IBM Toolbox for Java EnvironmentVariable クラスおよび EnvironmentVariableList クラスを使用して、IBM i 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。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
// Create the 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();
```

EnvironmentVariable Javadoc

EnvironmentVariableList Javadoc

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、次のとおりです。

- **エラー・タイプ:** スロー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。

例: スローされる例外をキャッチする

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。

注: 法律上の重要な情報に関しては、コードに関するライセンス情報および特記事項をお読みください。

```
// 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 that you want to track...

      default:
```

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

FileAttributes クラス

IBM Toolbox for Java FileAttributes クラスは、Get Attributes (Qp0lGetAttr) および Set Attributes (Qp0lSetAttr) API を介して検索および設定できる一連のファイル属性を表します。

オブジェクトの属性を取得するには、そのオブジェクトが存在していて、しかも呼び出し元がそれに対する権限を持っている必要があります。個々のファイル・システム、オブジェクト・タイプ、およびシステム操作リリースでサポートされている属性のみを、検索または設定することができます。

利用できる属性の完全なリストは、FileAttributes Javadoc に収められています。

関連情報

FileAttributes Javadoc

Get Attributes (Qp0lGetAttr) API

Set Attributes (Qp0lSetAttr) API

FTP クラス

IBM Toolbox for Java 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、サーバーに入れる。

例: FTP を使用してサーバーから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、 コードに関するライセンス情報および特記事項をお読みください。

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。

AS400FTP 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 IBM i 上の FTP サーバーのセマンティクスを理解します。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。

例: AS400FTP を使用してファイルをサーバーに保管する

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

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、 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");
```

FTP Javadoc

AS400FTP Javadoc

統合ファイル・システム・クラス

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、IBM i 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 IBM i 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。

さらに、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・クラスを介してシステム上のストリーム・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・システム側で宛先変更のメソッドを提供しなければなりません。たとえば、Windows 95 または 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 呼び出しをシステムへ転送するには、IBM i 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、IBM i 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 オブジェクトとシステムが接続されます。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
```

例: 統合ファイル・システム・クラスを使用する

549 ページの『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』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、システム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。

551 ページの『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』は、統合ファイル・システム・クラスを使って、システム上のディレクトリーの内容をリストする方法を示します。

IFSFile クラス:

IBM Toolbox for Java IFSFile クラスは、 IBM i 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() を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。

- 544 ページの『例: ディレクトリーを作成する』
- 545 ページの『例: IFSFile 例外を使用してエラーを追跡する』
- 546 ページの『例: .txt の拡張子が付けられたファイルをリストする』
- 546 ページの『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』

関連情報

IFSFile Javadoc

IFSJavaFile クラス:

この IBM Toolbox for Java クラスは、IBM i 統合ファイル・システム内のファイルを表し、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します。

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。

例: IFSJavaFile を使用する

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

以下の例では、IFSJavaFile クラスを使用する方法を示します。

```
// 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());
}
```

関連情報

IFSJavaFile Javadoc

IFSFileInputStream:

IBM Toolbox for Java 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 IBM iプ

ラットフォーム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アンロックする。
- 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。

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アンロックする。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。

例: IFSFileInputStream を使用する

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();
```

関連情報

IFSFileInputStream Javadoc

IFSTextFileInputStream クラス:

IFSTextFileInputStream は使用すべきでなく、クラス IFSFileReader によって置換されます。

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。保管されているデータが 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();
```

関連資料

『IFSFileReader』
統合ファイル・システムの文字ファイルの読み取りにはこのクラスを使用します。

IFSFileReader:

統合ファイル・システムの文字ファイルの読み取りにはこのクラスを使用します。

IFSFileReader は、文字ストリームの読み取りを意図したものです。IFSFileReader は、IFSTextFileOutputStream を置換します。

例: IFSFileReader を使用する

以下の例は、IFSFileReader の使用法を示します。

```
import java.io.BufferedReader;

  // Work with /File1 on the system eniac.
 AS400 system = new AS400("eniac");
 IFSFile file = new IFSFile(system, "/File1");
 BufferedReader reader = new BufferedReader(new IFSFileReader(file));

  // Read the first line of the file, converting characters.
 String line1 = reader.readLine();

  // Display the String that was read.
System.out.println(line1);

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

関連情報

IFSFileReader Javadoc

IFSFileOutputStream クラス:

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アンロックする。
- 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。

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アンロックする。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。

例: IFSFileOutputStream を使用する

以下の例では、IFSFileOutputStream 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");

     // 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();
```

IFSFileOutputStream Javadoc

IFSTextFileOutputStream クラス:

IFSTextFileOutputStream は使用すべきでなく、クラス IFSFileWriter によって置換されます。

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。

例: IFSTextFileOutputStream を使用する

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();
```

関連資料

『IFSFileWriter』
統合ファイル・システムの文字ファイルの書き込みには IFSFileWriter を使用します。IFSFileWriter は、文字ストリームの書き込みを意図したものです。

IFSFileWriter:

統合ファイル・システムの文字ファイルの書き込みには IFSFileWriter を使用します。IFSFileWriter は、文字ストリームの書き込みを意図したものです。

IFSFileWriter は、IFSTextFileOutputStream の置き換えです。

例: IFSFileWriter を使用する

以下の例は、IFSFileWriter の使用法を示します。

```
import java.io.PrintWriter;
import java.io.BufferedWriter;

// Work with /File1 on the system mysystem.
AS400 as400 = new AS400("mysystem");
IFSFile file = new IFSFile(system, "/File1");
PrintWriter writer = new PrintWriter(new BufferedWriter(new IFSFileWriter(file)));
// Write a line of text to the file, converting characters.
writer.println(text);

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

関連情報

IFSFileWriter Javadoc

IFSRandomAccessFile:

IBM Toolbox for Java 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 IBM i 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。
- ファイルから現行オフセットでデータを読み取る。
- 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。

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アンロックする。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。

例: IFSRandomAccessFile を使用する

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();
```

関連情報

IFSRandomAccessFile Javadoc

IFSFileDialog:

IBM Toolbox for Java 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。

このクラスは、IFSFile クラスを使用して、IBM i 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全探索します。このクラスのメソッドにより、 Java プログラムはダイアログのプッシュボタンにテキストを設定し、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。

例: IFSFileDialog を使用する

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、およびダイアログのプッシュボタンにテキストを設定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());
```

関連情報

FileFilter Javadoc

IFSFileDialog Javadoc

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 1 つ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)。

例: IFSKey を使用する

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();
```

関連情報

IFSKey Javadoc

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();
```

IFSSystemView:

IFSSystemView は、javax.swing.JFileChooser オブジェクトの構成時に使用する、IBM i 統合ファイル・システムへのゲートウェイを提供します。

JFileChooser は、ファイルのナビゲートや選択を行うダイアログを構築するための Java の標準的な方法です。

例: IFSSystemView を使用する

以下の例で、IFSSystemView の使用法を示します。

```
  import com.ibm.as400.access.AS400;
  import com.ibm.as400.access.IFSJavaFile;
  import com.ibm.as400.access.IFSSystemView;
  import javax.swing.JFileChooser;
  import java.awt.Frame;

// Work with directory /Dir on the system myAS400.
  AS400 system = new AS400("myAS400");
  IFSJavaFile dir = new IFSJavaFile(system, "/Dir");

  JFileChooser chooser = new JFileChooser(dir, new IFSSystemView(system));
```

```
  Frame parent = new Frame();
  int returnVal = chooser.showOpenDialog(parent);
  if (returnVal == JFileChooser.APPROVE_OPTION) {
IFSJavaFile chosenFile = (IFSJavaFile)(chooser.getSelectedFile());
    System.out.println("You selected the file named " +
                       chosenFile.getName());
  }
```

関連情報

IFSSystemView Javadoc

ISeriesNetServer クラス

ISeriesNetServer クラスは、サーバー上の NetServer サービスを表します。このクラスを使用すると、NetServer の状態と構成を照会し、変更できます。

ISeriesNetServer は、NetServer クラスを置き換えます。

関連情報

ISeriesNetServer Javadoc

JavaApplicationCall クラス

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

JavaApplicationCall Javadoc 参照資料の例では、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。

JavaApplicationCall

関連情報

JavaApplicationCall Javadoc

JDBC クラス

JDBC 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。

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 (JDBC 4.0 での拡張サポート)
CallableStatement は、SQL ストアード・プロシージャーを実行します。	JDK 1.1 (JDBC 4.0 での拡張サポート)
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 (JDBC 4.0 での拡張サポート)
Connection は、特定のデータベースへの接続を確立します。	JDK 1.1 (JDBC 4.0 での拡張サポート)
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 (JDBC 4.0 での拡張サポート)
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 (JDBC 4.0 での拡張サポート)
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 (JDBC 4.0 での拡張サポート)
PooledConnection は、接続プール管理用のフックとして機能します。	JDBC 2.0 Optional Package (JDBC 4.0 での拡張サポート)
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 (JDBC 4.0 での拡張サポート)
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 (JDBC 4.0 での拡張サポート)
ResultSetMetaData は、特定の ResultSet についての情報を提供します。	JDK 1.1 (JDBC 4.0 での拡張サポート)
RowId は、SQL ROWID 値を表します。	JDBC 4.0
RowSet は、ResultSet をカプセル化する接続された行のセットです。	JDBC 2.0 Optional Package
Savepoint トランザクションのよりち密な制御を提供します。	JDBC 3.0 API
SQLXML は、SQL XML 値を表します。	JDBC 4.0
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 (JDBC 4.0 での拡張サポート)

サポートされる JDBC インターフェース	必要な API
StatementEvent は、PooledConnection に登録されたすべての StatementEventListeners に送信されます。これが起きるのは、PooledConnection に関連した PreparedStatement のクローズをドライバーが確認した場合か、またはドライバーによって無効と判断された場合です。	
StatementEventListener は、登録されて、Statement プール内の PreparedStatements 上でイベントが発生すると通知を受け取ります。	JDBC 4.0
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

関連資料

342 ページの『JDBC』
JDBC 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。

554 ページの『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』
このプログラムは、IBM Toolbox for Java JDBC ドライバーを使用して、テーブルを作成し、そこにデータを取り込みます。

565 ページの『例: JDBCQuery を使用してテーブルを照会する』
このプログラムは、IBM Toolbox for Java JDBC ドライバーを使用して、テーブルを照会し、その内容を出力します。

関連情報

JTOpen: The Open Source version of the IBM Toolbox for Java

AS400JDBCConnectionPoolDataSource Javadoc

AS400JDBCPooledConnection Javadoc

RowID

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

以下の例では、AS400JDBCBlob クラスを使用して BLOB から読み取ったり BLOB を更新する方法を示します。

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

AS400JDBCBlob Javadoc

AS400JDBCBlobLocator Javadoc

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します。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。

例: CallableStatement を使用する

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();
```

AS400JDBCCallableStatement Javadoc

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、356 ページの『IBM Toolbox for JavaJDBC 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

以下の例では、AS400JDBCClob クラスを使用して CLOB から読み取ったり CLOB を更新したりする方法を示します。

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

AS400JDBCClob Javadoc

AS400JDBCClobLocator Javadoc

AS400JDBCConnection クラス:

AS400JDBCConnection クラスは、特定の DB2® for IBM i 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。詳しくは、86 ページの『JDBC ドライバーの登録』 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、356 ページの『IBM Toolbox for JavaJDBC 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、保管ポイント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。これらや他の機能拡張について詳しくは、352 ページの『バージョン 5 リリース 3 に対する JDBC 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。詳細は、26 ページの『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、IBM i 5.2 以降を実行しているサーバーへ接続する場合、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、保管ポイントを使用する
  - 保管点を設定する
  - 保管点をロールバックする
  - 保管ポイントをリリースする

AS400JDBCConnection Javadoc

AS400JDBCConnectionPool:

AS400JDBCConnectionPool クラスは、JDBC 2.0 Optional Package API 用の IBM Toolbox for Java サポートの一部として利用可能な Java プログラムで使用できる 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 (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();
```

AS400JDBCConnectionPool Javadoc

AS400JDBCConnectionPool Javadoc

AS400JDBCConnectionPoolDataSource Javadoc

AS400JDBCManagedConnectionPoolDataSource クラス:

JDBC 接続プールは、接続の再利用を可能にする管理対象の接続プールです。

AS400JDBCManagedConnectionPoolDataSource クラスは、ユーザー・アプリケーションが独自の管理コードをインプリメントする手間を省くことによって、接続プールの保守を簡素化します。接続プールは、JDBC DataSource インターフェース・メカニズムと組み合わせて使用します。クラス AS400JDBCManagedConnectionPoolDataSource には、javax.sql.DataSource インターフェースのインプリメンテーションが入っています。このクラスは、接続プール・メカニズムの機能性を管理し、通常の DataSource と同様にそこにアクセスできるようにします。自己管理 JDBC 接続プール・マネージャーをセットアップして使用するには、その一環として、Java Naming and Directory Interface (JNDI) を使用して接続プールの管理対象データ・ソースをバインドします。自己管理の JDBC 接続プールが機能するには、JNDI が必要です。

以下のプロパティーを使って、接続プールの属性を構成することができます。

- InitialPoolSize
- MinPoolSize
- MaxPoolSize
- MaxLifetime
- MaxIdleTime
- PropertyCycle

- ReuseConnections

詳細は、setX() メソッド上の Javadocs を参照してください。ただし X は、プロパティーです。

例: AS400JDBCManagedConnectionPoolDataSource クラスの使用:

以下の例は、AS400JDBCManagedConnectionPoolDataSource クラスの使用を説明しています。AS400JDBCManagedConnectionPoolDataSource クラスは、ユーザー・アプリケーションが独自の管理コードをインプリメントする手間を省くことによって、接続プールの保守を簡素化します。

注: このコード例を使用することによって、お客様は 811 ページの『コードに関するライセンス情報および特記事項』 の条件に同意されたものとします。

例 1

この簡単な例は、AS400JDBCManagedConnectionPoolDataSource クラスの基本的な用法を示しています。

```
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.sql.DataSource;

import com.ibm.as400.access.AS400JDBCManagedConnectionPoolDataSource;
import com.ibm.as400.access.AS400JDBCManagedDataSource;


public class TestJDBCConnPoolSnippet
{
    void test()
    {
        AS400JDBCManagedConnectionPoolDataSource cpds0 = new AS400JDBCManagedConnectionPoolDataSource();

        // Set general datasource properties.  Note that both connection pool datasource (CPDS) and managed
               // datasource (MDS) have these properties, and they might have different values.
        cpds0.setServerName(host);
        cpds0.setDatabaseName(host);//iasp can be here
        cpds0.setUser(userid);
        cpds0.setPassword(password);


        cpds0.setSavePasswordWhenSerialized(true);

        // Set connection pooling-specific properties.
        cpds0.setInitialPoolSize(initialPoolSize_);
        cpds0.setMinPoolSize(minPoolSize_);
        cpds0.setMaxPoolSize(maxPoolSize_);
        cpds0.setMaxLifetime((int)(maxLifetime_/1000));  // convert to seconds
        cpds0.setMaxIdleTime((int)(maxIdleTime_/1000));  // convert to seconds
        cpds0.setPropertyCycle((int)(propertyCycle_/1000));  // convert to seconds
        //cpds0.setReuseConnections(false);  // do not re-use connections

        // Set the initial context factory to use.
        System.setProperty(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.fscontext.RefFSContextFactory");


        // Get the JNDI Initial Context.
        Context ctx = new InitialContext();

        // Note: The following is an alternative way to set context properties locally:
        //   Properties env = new Properties();
        //   env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.fscontext.RefFSContextFactory");
        //   Context ctx = new InitialContext(env);

        ctx.rebind("mydatasource", cpds0);  // We can now do lookups on cpds, by the name "mydatasource".

        // Create a standard DataSource object that references it.

        AS400JDBCManagedDataSource mds0 = new AS400JDBCManagedDataSource();
        mds0.setDescription("DataSource supporting connection pooling");
        mds0.setDataSourceName("mydatasource");
        ctx.rebind("ConnectionPoolingDataSource", mds0);

        DataSource dataSource_ = (DataSource)ctx.lookup("ConnectionPoolingDataSource");
```

```
        AS400JDBCManagedDataSource mds_ = (AS400JDBCManagedDataSource)dataSource_;

        boolean isHealthy = mds_.checkPoolHealth(false);  //check pool health

        Connection c = dataSource_.getConnection();

    }

}
```

例 2

この例は、AS400JDBCManagedConnectionPoolDataSource クラスの用法をさらに詳しく示しています。

```
import java.awt.TextArea;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.OutputStream;
import java.io.PrintStream;
import java.util.Vector;
import java.util.Properties;

import java.sql.Connection;
import javax.sql.DataSource;
import java.sql.ResultSet;
import java.sql.Statement;
import javax.naming.*;
import java.util.Date;
import java.util.ArrayList;
import java.util.Random;
import com.ibm.as400.access.AS400;
import com.ibm.as400.access.AS400JDBCManagedConnectionPoolDataSource;
import com.ibm.as400.access.AS400JDBCManagedDataSource;
import com.ibm.as400.access.Trace;


public class TestJDBCConnPool
{

    private static final boolean DEBUG = false;

    // If you turn this flag on, be sure to also turn on the following flag:
    //   AS400JDBCConnection.TESTING_THREAD_SAFETY.
    private static final boolean TESTING_THREAD_SAFETY = false;

    private static String userid;
    private static String password;
    private static String host;

    // Note: For consistency, all time values are stored units of milliseconds.
    private int initialPoolSize_;  // initial # of connections in pool
    private int minPoolSize_;      // max # of connections in pool
    private int maxPoolSize_;      // max # of connections in pool
    private long maxLifetime_;     // max lifetime (msecs) of connections in pool
    private long maxIdleTime_;     // max idle time (msecs) of available connections in pool
    private long propertyCycle_;   // pool maintenance frequency (msecs)

    private int numDaemons_;       // # of requester daemons to create
    private static long timeToRunDaemons_; // total duration (msecs) to let the daemons run
    private long daemonMaxSleepTime_;  // max time (msecs) for requester daemons to sleep each cycle
    private long daemonMinSleepTime_;  // min time (msecs) for requester daemons to sleep each cycle
    private long poolHealthCheckCycle_;  // # of msecs between calls to checkPoolHealth()

    private boolean keepDaemonsAlive_ = true;  // When this is false, the daemons shut down.

    private DataSource dataSource_;
    private AS400JDBCManagedDataSource mds_;

    private final Object daemonSleepLock_ = new Object();

    private Random random_ = new Random();

    static
    {
```

```
        try {
            Class.forName("com.ibm.as400.access.AS400JDBCDriver");
        }
        catch(Exception e){
            System.out.println("Unable to register JDBC driver.");
            System.exit(0);
        }
    }

    public static void main(String[] args)
    {
        host = args[0];
        userid = args[1];
        password = args[2];
        timeToRunDaemons_ = (new Integer(args[3])).intValue() * 1000; //milliseconds
        //args[3]=time to run in seconds
        TestJDBCConnPool cptest = new TestJDBCConnPool();

        cptest.setup();
        cptest.runTest();
    }

    public void setup()
    {
        try
        {
            if (DEBUG)
              System.out.println("TESTING_THREAD_SAFETY flag is "
                                 + (TESTING_THREAD_SAFETY ? "true" : "false"));

            if (TESTING_THREAD_SAFETY)
            {
                // Adjust values for performing thread-intensive stress testing.
                // NOTE: This assumes that the AS400JDBCConnection class has also been modified to
                             // not make actual connections to an actual server.
                // To do this, edit AS400JDBCConnection.java, changing its TESTING_THREAD_SAFETY
                             // flag to 'false', and recompile.
                minPoolSize_ = 100;
                maxPoolSize_ = 190;
                initialPoolSize_ = 150;  // this should get reset to maxPoolSize_
                numDaemons_ = 75;
                if (timeToRunDaemons_ == 0) {
                    timeToRunDaemons_ = 180*1000; // 180 seconds == 3 minutes
                }
            }
            else
            {  // Set more conservative values, as we'll be making actual connections to an
               // actual server, and we don't want to monopolize the server.
                minPoolSize_ = 5;
                maxPoolSize_ = 15;
                initialPoolSize_ = 9;
                numDaemons_ = 4;
                if (timeToRunDaemons_ == 0) {
                    timeToRunDaemons_ = 15*1000; // 15 seconds
                }
            }
            maxLifetime_ = (int)timeToRunDaemons_ / 3;
            maxIdleTime_ = (int)timeToRunDaemons_ / 4;
            propertyCycle_ = timeToRunDaemons_ / 4;
            poolHealthCheckCycle_ = Math.min(timeToRunDaemons_ / 4, 20*60*1000);
            // at least once every 20 minutes (more frequently if shorter run-time)
            daemonMaxSleepTime_ = Math.min(timeToRunDaemons_ / 3, 10*1000);
            // at most 10 seconds (less if shorter run-time)
            daemonMinSleepTime_ = 20;  // milliseconds

            if (DEBUG)
               System.out.println("setup: Constructing "
                             + "AS400JDBCManagedConnectionPoolDataSource (cpds0)");
            AS400JDBCManagedConnectionPoolDataSource cpds0 = new AS400JDBCManagedConnectionPoolDataSource();
// Set datasource properties.  Note that both CPDS and MDS have these
            // properties, and they might have different values.
            cpds0.setServerName(host);
            cpds0.setDatabaseName(host);//iasp can be here
            cpds0.setUser(userid);
            cpds0.setPassword(password);

            cpds0.setSavePasswordWhenSerialized(true);

            // Set connection pooling-specific properties.
```

```
        cpds0.setInitialPoolSize(initialPoolSize_);
        cpds0.setMinPoolSize(minPoolSize_);
        cpds0.setMaxPoolSize(maxPoolSize_);
        cpds0.setMaxLifetime((int)(maxLifetime_/1000));  // convert to seconds
        cpds0.setMaxIdleTime((int)(maxIdleTime_/1000));  // convert to seconds
        cpds0.setPropertyCycle((int)(propertyCycle_/1000));  // convert to seconds
        //cpds0.setReuseConnections(false);  // don't re-use connections

        // Set the initial context factory to use.
        System.setProperty(Context.INITIAL_CONTEXT_FACTORY,
                           "com.sun.jndi.fscontext.RefFSContextFactory");

        // Get the JNDI Initial Context.
        Context ctx = new InitialContext();

        // Note: The following is an alternative way to set context properties locally:
        //   Properties env = new Properties();
        //   env.put(Context.INITIAL_CONTEXT_FACTORY,
        //           "com.sun.jndi.fscontext.RefFSContextFactory");
        //   Context ctx = new InitialContext(env);

        ctx.rebind("mydatasource", cpds0);
                  // We can now do lookups on cpds, by the name"mydatasource".

        if (DEBUG)
          System.out.println("setup: lookup(\"mydatasource\"" + ")");
       //   AS400JDBCManagedConnectionPoolDataSource cpds1 =
       //   (AS400JDBCManagedConnectionPoolDataSource)ctx.lookup("mydatasource");
       // if (DEBUG) System.out.println("setup: cpds1.getUser() == |" + cpds1.getUser() + "|");

        // Create a standard DataSource object that references it.

        if (DEBUG)
           System.out.println("setup: Constructing AS400JDBCManagedDataSource (mds0)");
        AS400JDBCManagedDataSource mds0 = new AS400JDBCManagedDataSource();
        mds0.setDescription("DataSource supporting connection pooling");
        mds0.setDataSourceName("mydatasource");
        ctx.rebind("ConnectionPoolingDataSource", mds0);

        if (DEBUG)
          System.out.println("setup: lookup(\"ConnectionPoolingDataSource\"" + ")");
        dataSource_ = (DataSource)ctx.lookup("ConnectionPoolingDataSource");
        //dataSource_.setLogWriter(output_);
        if (DEBUG)
          System.out.println("setup: dataSource_.getUser() == |" +
                ((AS400JDBCManagedDataSource)dataSource_).getUser() + "|");

        mds_ = (AS400JDBCManagedDataSource)dataSource_;
    }
    catch (Exception e)
    {
        e.printStackTrace();
        System.out.println("Setup error during Trace file creation.");
    }
}

void displayConnectionType(Connection conn, boolean specifiedDefaultId)
{
    if (conn instanceof com.ibm.as400.access.AS400JDBCConnectionHandle)
        System.out.print("("+ (specifiedDefaultId ? "+" : "-") + "P)");
    else
        System.out.print("("+ (specifiedDefaultId ? "+" : "-") + "NP)");
}


/**
 *  Gets and returns connections from and to a connection pool for a while.
 **/
public void runTest()
{
    boolean ok = true;
    try
    {
        System.out.println("Started test run at " + new Date());

        if (DEBUG)
          System.out.println("Checking health just after datasource creation "
                     + "(we expect that the pool does not exist yet) ...");
        if (mds_.checkPoolHealth(true)) {
```

```
    ok = false;
    System.out.println("\nERROR: Pool exists prior to first getConnection().");
}

// Verify some setters/getters for JDBC properties.
System.out.println("Verifying setters/getters ...");

mds_.setAccess("read only");
if (!mds_.getAccess().equals("read only")) {
    ok = false;
    System.out.println("\nERROR: getAccess() returned unexpected value: "
            + "|" + mds_.getAccess()+"|");
}

boolean oldBool = mds_.isBigDecimal();
boolean newBool = (oldBool ? false : true);
mds_.setBigDecimal(newBool);
if (mds_.isBigDecimal() != newBool) {
    ok = false;
    System.out.println("\nERROR: isBigDecimal() returned unexpected value: "
                        + "|"+mds_.isBigDecimal()+"|");
}
mds_.setBigDecimal(oldBool);

int oldInt = mds_.getBlockCriteria();
int newInt = (oldInt == 2 ? 1 : 2);
mds_.setBlockCriteria(newInt);
if (mds_.getBlockCriteria() != newInt) {
    ok = false;
    System.out.println("\nERROR: getBlockCriteria() returned unexpected value: "
                       + "|"+mds_.getBlockCriteria()+"|");
}
mds_.setBlockCriteria(oldInt);

// Verify some setters and getters for socket properties.

oldBool = mds_.isKeepAlive();
newBool = (oldBool ? false : true);
mds_.setKeepAlive(newBool);
if (mds_.isKeepAlive() != newBool) {
    ok = false;
    System.out.println("\nERROR: isKeepAlive() returned unexpected value: "
                       + "|"+mds_.isKeepAlive()+"|");
}
mds_.setKeepAlive(oldBool);

oldInt = mds_.getReceiveBufferSize();
newInt = (oldInt == 256 ? 512 : 256);
mds_.setReceiveBufferSize(newInt);
if (mds_.getReceiveBufferSize() != newInt) {
    ok = false;
    System.out.println("\nERROR: getReceiveBufferSize() returned unexpected value: "
                       + "|"+mds_.getReceiveBufferSize()+"|");
}
mds_.setReceiveBufferSize(oldInt);


System.out.println("CONNECTION 1");
Object o = dataSource_.getConnection();
System.out.println(o.getClass());
System.out.println("******LOOK ABOVE*******");
Connection c1 = dataSource_.getConnection();

if (DEBUG)
  displayConnectionType(c1, true);

if (DEBUG)
  System.out.println("Checking health after first getConnection() ...");
if (!mds_.checkPoolHealth(true)) {
    ok = false;
    System.out.println("\nERROR: Pool is not healthy after first getConnection().");
}

if (!TESTING_THREAD_SAFETY)
{
    try
    {
        c1.setAutoCommit(false);
        if (DEBUG)
```

```
          System.out.println("SELECT * FROM QIWS.QCUSTCDT");
        Statement s = c1.createStatement();
        ResultSet rs = s.executeQuery("SELECT * FROM QIWS.QCUSTCDT");
        while (rs.next ()) {
            if (DEBUG)
              System.out.println(rs.getString(2));
        }
        rs.close();
        s.close();
    }
    catch (Exception e) {
        e.printStackTrace();
        if (DEBUG)
          System.out.println("Checking health after fatal connection error ...");
        if (!mds_.checkPoolHealth(true)) {
          ok = false;
          System.out.println("\nERROR: Pool is not healthy after fatal connection "
                             + "error.");
        }
    }
}

System.out.println("CONNECTION 2");
Connection c2 = dataSource_.getConnection(userid, password);
if (DEBUG)
  displayConnectionType(c2, false);
System.out.println("CONNECTION 3");
Connection c3 = dataSource_.getConnection();
if (DEBUG)
  displayConnectionType(c3, true);
c1.close();

if (DEBUG)
  System.out.println("Checking health after first close() ...");
if (!mds_.checkPoolHealth(true)) {
    ok = false;
    System.out.println("\nERROR: Pool is not healthy after first close().");
}

System.out.println("CONNECTION 4");
Connection c4 = dataSource_.getConnection();
if (DEBUG) displayConnectionType(c4, true);

c1.close();  // close this one again
c2.close();
c3.close();
c4.close();

if (DEBUG) System.out.println("Checking health after last close() ...");
if (!mds_.checkPoolHealth(true)) {
    ok = false;
    System.out.println("\nERROR: Pool is not healthy after last close().");
}

// Start the test daemons.
System.out.println("Starting test daemons");
startThreads();

// Run the test daemons for a while; check pool health periodically.

long startTime = System.currentTimeMillis();
long endTime = startTime + timeToRunDaemons_;
while (System.currentTimeMillis() < endTime)
{
    System.out.print("h");
    // Let the daemons run for a while, then check pool health.
    try {
        Thread.sleep(poolHealthCheckCycle_);
    }
    catch (InterruptedException ie) {}
    if (!mds_.checkPoolHealth(true)) {
        ok = false;
        System.out.println("\nERROR: Pool is not healthy after test daemons started.");
    }
}

// Stop the test daemons.
System.out.println("\nStopping test daemons");
stopThreads();
```

```
        if (DEBUG)
          System.out.println("Checking health after connectionGetter daemons have run...");
        if (!mds_.checkPoolHealth(true)) {
            ok = false;
            System.out.println("\nERROR: Pool is not healthy after test daemons stopped.");
        }

        if (!TESTING_THREAD_SAFETY)
        {
            System.out.println("CONNECTION 5");
            Connection c = dataSource_.getConnection();
            if (DEBUG) displayConnectionType(c, true);
            c.setAutoCommit(false);
            if (DEBUG) System.out.println("SELECT * FROM QIWS.QCUSTCDT");
            Statement s = c.createStatement();
            ResultSet rs = s.executeQuery("SELECT * FROM QIWS.QCUSTCDT");
            while (rs.next ()) {
                if (DEBUG) System.out.println(rs.getString(2));
            }
            rs.close();
            s.close();
            c.close();            }

        System.out.println("\nClosing the pool...");
        mds_.closePool();

        if (DEBUG)
          System.out.println("Checking health after pool closed ...");
        Trace.setTraceJDBCOn(true);  // make sure the final stats get printed out
        Trace.setTraceOn(true);
        if (!mds_.checkPoolHealth(true)) {
            ok = false;
            System.out.println("\nERROR: Pool is not healthy after pool closed.");
        }

        System.out.println();
        if(ok==true)
            System.out.println("test ran ok");
        else
            System.out.println("test failed");
    }
    catch (Exception e)
    {
        System.out.println(e);
        e.printStackTrace();
    }
    finally {
        System.out.println("Ended test at " + new Date());
    }
}

void startThreads()
{
    // Create a bunch of threads that call getConnection().
    Thread[] threads = new Thread[numDaemons_];
    for (int i=0; i<numDaemons_; i++)
    {
        ConnectionGetter getter;
        // Flip a coin to see if this daemon will specify the default uid, or unique uid.
        if (random_.nextBoolean())
        {
            getter = new ConnectionGetter(userid,password);
            if (TESTING_THREAD_SAFETY) {  // we can use fictional userid
                getter = new ConnectionGetter("Thread"+i, "Pwd"+i);
            }
            else {  // must use a real userid
                getter = new ConnectionGetter(userid,password);
            }
        }
        else
            getter = new ConnectionGetter(null, null);

        threads[i] = new Thread(getter, "["+i+"]");
        threads[i].setDaemon(true);
    }

    // Start the threads.
    for (int i=0; i<numDaemons_; i++)
```

```
    {
        threads[i].start();
    }
}

void stopThreads()
{
    // Tell the threads to stop.
    keepDaemonsAlive_ = false;
    synchronized (daemonSleepLock_) {
        daemonSleepLock_.notifyAll();
    }

    // Wait for the daemons to stop.
    try {
        Thread.sleep(3000);
    }
    catch (InterruptedException ie) {}
}


// ConnectionGetter ---------------------------------------------------------------

/**
     Helper class. This daemon wakes up at random intervals and either
     gets another connection from the connection pool or returns a
     previously-gotten connection to the pool.
 **/
private final class ConnectionGetter implements Runnable
{
    private String uid_;
    private String pwd_;
    private boolean useDefaultUid_;
    private long maxSleepTime_;
    private String threadName_;
    private boolean firstConnection_ = true;
    ArrayList connections_ = new ArrayList();
    // list of connections that this getter currently 'owns'.

    ConnectionGetter(String uid, String pwd) {
        uid_ = uid;
        pwd_ = pwd;
        if (uid_ == null) useDefaultUid_ = true;
        else useDefaultUid_ = false;
        maxSleepTime_ = daemonMaxSleepTime_;  // our own copy that we can adjust
    }

    public void run()
    {
        threadName_ = Thread.currentThread().getName();
        if (DEBUG) System.out.println("ConnectionGetter("+threadName_+") Starting up");

        try
        {
            while (keepDaemonsAlive_)
            {
                try
                {
                    // Pick a random sleep-time, between min and max values.
                    long sleepTime = Math.max((long)(maxSleepTime_ * random_.nextFloat()),
                            daemonMinSleepTime_);
                    // Note: Must never call wait(0), because that waits forever.
                    synchronized (daemonSleepLock_) {
                        try {
                            daemonSleepLock_.wait(sleepTime);
                            System.out.print(".");
                        }
                        catch (InterruptedException ie) {}
                    }
                    if (!keepDaemonsAlive_) break;

                    // Decide by chance whether to request another connection or return a
                    // previously-obtained connection.
                    Connection conn;
                    if (random_.nextBoolean())  // Leave the decision to chance.
                    { // Request another connection.
                        if (useDefaultUid_)
                        {
                            if (DEBUG)
```

```
                        System.out.println("ConnectionGetter("+threadName_+") - get()");
                    conn = dataSource_.getConnection();
                }
                else
                {
                    if (DEBUG)
                       System.out.println("ConnectionGetter("+threadName_+") - "
                                          + "get("+uid_+",***)");
                    conn = dataSource_.getConnection(uid_, pwd_);
                }

                if (conn == null) {
                    System.out.println("ConnectionGetter("+threadName_+") ERROR: "
                                       + "getConnection() returned null");
                }
                else
                {
                    // Occasionally "forget" that we own a connection, and neglect to
                    // close it.
                    // Orphaned connections should eventually exceed their maximum
                    // lifetime and get "reaped" by the connection manager.
                    float val = random_.nextFloat();
                    if (firstConnection_ || val < 0.1) {
                       // 'orphan' a few gotten connections
                       firstConnection_ = false;
                    }
                    else {
                        connections_.add(conn);
                    }
                    if (DEBUG) displayConnectionType(conn, useDefaultUid_);
                    if (conn instanceof com.ibm.as400.access.AS400JDBCConnectionHandle)
                    {  // We got a pooled connection.  Try speeding up our cycle time.
                        if (maxSleepTime_ > 100)
                          maxSleepTime_--;
                        else
                          maxSleepTime_ = 100;
                    }
                    else
                    {  // We didn't get a pooled connection.  That means that the pool
                       // must be at capacity.  Slow down our cycle time a bit.
                        maxSleepTime_ = maxSleepTime_ + 50;
                    }
                }
            }
            else { // Close a connection that we currently own.
                if (connections_.size() != 0) {
                    conn = (Connection)connections_.remove(0);
                    conn.close();
                }
            }
          }  // inner try
          catch (Exception e)
          {
              e.printStackTrace();
          }
        } // outer while
      }  // outer try
      finally
      {
          if (DEBUG)
            System.out.println("ConnectionGetter("+threadName_+") Stopping");
          // Return all the connections that I still have in my list.
          for (int i=0; i<connections_.size(); i++) {
              Connection conn = (Connection)connections_.remove(0);
              try { conn.close(); } catch (Exception e) { e.printStackTrace(); }
          }

      }
    }

  }  // internal class 'ConnectionGetter'
}
```

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();
```

関連情報

AS400JDBCDatabaseMetaData Javadoc

AS400JDBCDataSource クラス:

AS400JDBCDataSource クラスは、IBM i 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は、 808 ページの『IBM Toolbox for Java の関連情報』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 server.
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、サーバー上の統合ファイル・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 IBM i 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 IBM i 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");
```

AS400JDBCDataSource Javadoc

AS400JDBCConnectionPoolDataSource Javadoc

AS400JDBCPooledConnection Javadoc

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 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

  IBM 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");
  ```

  IBM 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、IBM Toolbox for Java 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 ());
  ```

  DriverManager.registerDriver() を呼び出すと、その結果として IBM Toolbox for Java JDBC ドライバーが 2 度登録されます。 AS400JDBCDriver が JVM によってロードされるときに 1 度、さらに registerDriver() メソッドを明示的に呼び出すときに 1 度です。これは DriverManager 実装の結果であり、IBM Toolbox for Java はこれを制御できません。 DriverManager のレジストリーにドライバーが 2 度リストされるとしても、通常は問題にはなりません。ただし、登録済みの両方のドライバー・リスト項目が使用されるような状況が発生することがあります。例えば、間違ったパスワードを使って接続の取得を試みた場合に、これが発生する可能性があります。登録済みの最初のドライバーでの接続取得が失敗すると、DriverManager は登録済みの 2 番目のドライバーを試行します。

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、486 ページの『Java プログラムの接続の管理』 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。詳細は、 26 ページの『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: システム名が指定されていない URL の使用

この例では、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。

```
// Connect to unnamed system.
// User receives prompt to type system name.
Connection c = DriverManager.getConnection("jdbc:as400:");
```

例: サーバー・データベースへの接続 (デフォルト SQL 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 SQL schema or properties are specified.
Connection c  = DriverManager.getConnection("jdbc:as400://mySystem");
```

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

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

例: サーバー・データベースへの接続およびプロパティーを指定するための java.util.Properties の使用

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、 356 ページの『IBM Toolbox for JavaJDBC 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: サーバー・データベースへの接続およびプロパティーを指定するための URL の使用

```
// 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: サーバー・データベースへの接続およびユーザー ID とパスワードの指定

```
// 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: データベースからの切断

サーバーから切断するには、Connecting オブジェクト上で close() メソッドを使用します。以下のステートメントを使用して、前の例で作成された接続をクローズします。

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

AS400JDBCDriver Javadoc

AS400JDBCParameterMetaData クラス:

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の取得または位取りの取得を行う

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);
```

関連情報

AS400JDBCParameterMetaData Javadoc

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。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、パフォーマンスが改善される場合があります。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。

例: PreparedStatement を使用する

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();
```

関連情報

AS400JDBCPreparedStatement Javadoc

ResultSet クラス:

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。

列索引を使用してもパフォーマンスは改善されますが、ResultSet を使用することにより、名前によって列を取得および更新できます。

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	isFirst
afterLast	isLast
beforeFirst	last
first	moveToCurrentRow
getRow	moveToInsertRow
isAfterLast	previous
isBeforeFirst	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、絶対位置決めもサポートしています。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。

注: IBM i では、スクロール可能な非認識カーソル用の読み取り専用アクセスのみが許可されています。IBM Toolbox for Java は、ResultSet が同時読み取り専用である場合、ResultSet スクロール非認識カーソルをサポートします。ResultSet タイプが非認識および同時更新可能に指定されている場合、ResultSet タイプは認識に変更され、ユーザーに対して警告が出されます。

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 インターフェースを通して利用できる更新メソッドの詳細なリストは、AS400JDBCResultSet Javadoc の中の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。

IBM i 5.2 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、パフォーマンスが低下する可能性があります。

AS400JDBCResultSet Javadoc

AS400ResultSetMetaData Javadoc

AS400JDBCRowSet クラス:

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 または AS400JDBCConnectionPoolDataSource のインスタンスを使用して、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();
```

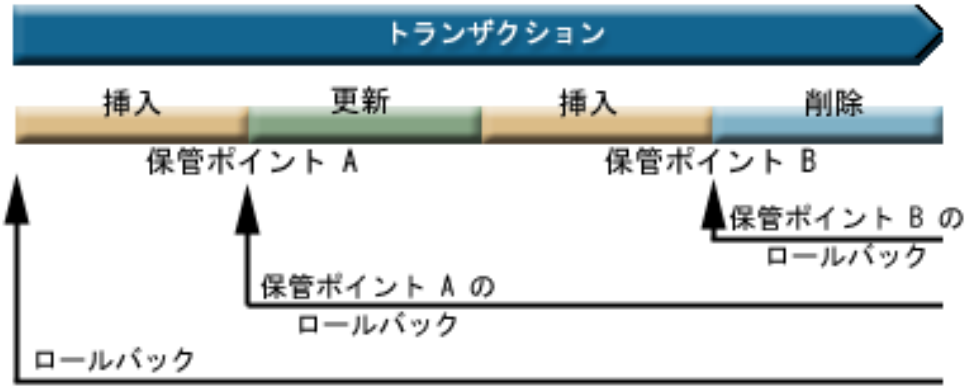
AS400JDBCRowSet Javadoc

AS400JDBCResultSet Javadoc

AS400JDBCSavepoint クラス:

IBM Toolbox for Java 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。保管ポイントを使用すると、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ポイントを使用する



たとえば、図 1 は 2 つの保管ポイント 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ポイント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ポイント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。一度保管ポイント A にロールバックすると、後から保管ポイント 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。保管ポイント 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: 保管ポイントを使用する

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ポイントを設定します。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、この保管ポイント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

保管ポイントを使用するには、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

IBM 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、保管ポイント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、 IBM Toolbox for Java はカーソルを移動またはクローズしません。

保管ポイント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ポイントまでに実行されたアクションのみを取り消します。その保管ポイント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、特定の保管ポイントより前に実行された作業は含まれますが、保管ポイント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、すべての保管ポイント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、保管ポイントを解放できます。

制約事項

以下の制約事項は保管ポイントを使用する際に適用されます。

- 保管ポイントの名前は固有でなければなりません。
- 保管ポイントが解放される、コミットされる、またはロールバックされるまで、保管ポイント名は再使用できません。
- 保管ポイントを有効にするには、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。保管ポイントの使用時に自動コミットを使用可能にすると、例外がスローされます。
- 保管ポイントは、XA 接続では無効です。 XA 接続で保管ポイントを使用すると、例外がスローされます。
- サーバーは IBM i 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。IBM i 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) の場合に保管ポイントを使用すると、例外がスローされます。

  AS400JDBCSavePoint Javadoc

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 インターフェース』: 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します。
- 89 ページの『PreparedStatement インターフェース』 - 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します。
- 71 ページの『CallableStatement インターフェース』 - 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します。 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、パフォーマンスが改善される場合があります。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ます。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、同時に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();
```

AS400JDBCStatement Javadoc

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。

注: V5R3 より前では、データベース・ホスト・サーバーは Job Scoped Locks 用 XA API (XA モデル) を使用しました。V5R3 およびこれ以降のリリースでは、データベース・ホスト・サーバーはすべての MTS 機能に、Transaction Scoped Locks 用 XA API (NTS モデル) を使用します。これらの API の相違についての詳細は、XA API を参照してください。

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();
```

Job クラス

アクセス・パッケージ内にある IBM Toolbox for Java Jobs クラスを使用すると、Java プログラムがジョブ情報の検索および変更を行うことができます。

Job 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

例

以下の例では、Job、JobList、および JobLog クラスのいくつかの使用法を示します。最初の例では、Job クラスでキャッシュを使用する 1 つの方法を示します。他の例へのリンクがサンプル・コードのすぐ後に続きます。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

例: 値の設定および取得を行う際にキャッシュを使用する

```
try {

    // Creates AS400 object.
    AS400 as400 = new AS400("systemName");

    // Constructs a Job object
    byte[] internalJobID = "xxxxxxxxxxxxxxxx".getBytes();
    Job job = new Job(as400, internalJobID);

    // 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 system.
    job.commitChanges();

    // Set job information to system directly(without cache).
    job.setCacheChanges(false);
    job.setRunPriority(60);
} catch (Exception e)
{
    System.out.println("error :" + e);
}
```

以下の例では、特定のユーザーに属するジョブのリスト方法、ジョブ状況情報を持ったジョブのリスト方法、およびジョブ・ログ内のメッセージの表示方法を示します。

567 ページの『例: JobList を使用してジョブ ID 情報のリストを表示する』

569 ページの『例: JobList を使用してジョブのリストを取得する』

572 ページの『例: JobLog を使用してジョブ・ログ内のメッセージを表示する』

アクセス・パッケージの Javadoc

Job クラス:

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、変更したりすることができます。

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。

例

サンプル・コードについて詳しくは、Job クラスの Javadoc 参照資料を参照してください。この例では、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を示します。

Job Javadoc

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、 IBM i ジョブをリストできます。

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、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。

例: JobList を使用する

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 server.
 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());
 }
```

Job Javadoc

JobLog クラス:

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。

例: JobLog を使用する

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 server
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());
        }

    }
}
```

Job Javadoc

Message クラス

IBM Toolbox for Java AS400Message クラスとそれに関連したクラスは、サーバーから戻されたメッセージを表します。

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 IBM i 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 IBM i 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。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
// 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: メッセージ・リストを使用する

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。

- 514 ページの『例: CommandCall を使用する』
- 585 ページの『例: ProgramCall を使用する』

QueuedMessage

QueuedMessage クラスは、AS400Message クラスを拡張します。

QueuedMessage クラスは、 IBM i 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。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
// The message queue is on this system.
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、 IBM i 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。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
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 IBM i メッセージ待ち行列と対話することができます。

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。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
// The message queue is on this system.
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());
}
```

AS400Message Javadoc

QueuedMessage Javadoc

アクセス・パッケージの要約

MessageFile Javadoc

MessageQueue Javadoc

NetServer クラス

NetServer クラスの使用は推奨されなくなり、ISeriesNetServer クラスで置き換えられました。

NetServer クラスは、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

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
// Create a system object to represent the 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.
```

ObjectReferences クラス

IBM Toolbox for Java ObjectReferences クラスは、Retrieve Object References (QP0LROR) API を介して取り出せるオブジェクトに対する統合ファイル・システム参照に関する一連の情報を表します。

参照とは、個別タイプのアクセスであるか、または統合ファイル・システム・インターフェースの使用時にオブジェクトに対して設定されたロックのことです。参照タイプが互いに競合しあわない限り、オブジェクトは、複数の参照を並行して持つことがあります。このクラスは、オブジェクトに対して現在かけられているバイト範囲ロックに関する情報を戻しません。

ユーザーは、オブジェクトの参照を取得するには、各ディレクトリーに対する実行 (*X) データ権限を予め保有している必要があります。ユーザーは、オブジェクトの参照を取得するには、そのオブジェクトに対する読み取り (*R) データ権限を保有している必要があります。

詳細は、ObjectReferences Javadoc を参照してください。

関連情報

ObjectReferences Javadoc

Retrieve Object References (QP0LROR) API

Permission クラス

IBM Toolbox for Java Permission クラスを使うと、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: Permission を使用する

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

以下の例では、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。

- DLOPermission
- QSYSPermission
- RootPermission

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: UserPermission を使用する

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

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());
}
```

Permission Javadoc

UserPermission Javadoc

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。DLO は、QDLS に保管されます。

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。

権限値	説明
*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、Security reference トピックを参照してください。

例: DLOPermission を使用する

次の例では、 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());
}
```

関連情報

DLOPermission Javadoc

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 IBM i 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。

システム定義の権限値	説明
*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。

表 1. **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	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。									

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()

さまざまな権限に関する詳細は、Security reference トピックを参照してください。オブジェクト権限の付与および編集をするために CL コマンドを使用する時の詳細については、CL コマンド 「オブジェクト権限を付与する (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());
}
```

QSYSPermission Javadoc

106 ページの『UserPermission クラス』

RootPermission:

RootPermission クラスは、ルート・ディレクトリー構造に収めら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。 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。

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したり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。

さまざまな権限に関する詳細は、Security reference トピック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());
}
```

関連情報

RootPermission Javadoc

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、用紙定義、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。

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。

例

- 例: スプール・ファイルの作成には、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。
- 例: SCS スプール・ファイルの作成には、 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。
- 例: スプール・ファイルの読み取りでは、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。
- 例: スプール・ファイルを読み取り変換するでは、スプール・ファイル・データを読み取る際に、異なる変換結果を得るための PrintObjectPageInputStream および PrintObjectTransformedInputStream の使用法を示します。

- 例: スプール・ファイルのコピーでは、コピーするファイルを含む同じ待ち行列にスプール・ファイルをコピーする方法が示されています。
- 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) では、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。
- 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) 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。
- 例: スプール・ファイルを同期でリストするには、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。

関連情報

PrintObject Javadoc

印刷オブジェクトのリスト:

IBM Toolbox for Java 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。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。

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。

PrintObjectList イベント	イベントが引き渡された時
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。

例

以下の例では、スプール・ファイルをリストするさまざまな方法を示します。

579 ページの『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)』では、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。

582 ページの『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)』 では、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。

584 ページの『例: スプール・ファイルを同期でリストする』 では、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。

PrintObjectList Javadoc

SpooledFileList Javadoc

AFPResource Javadoc

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、印刷オブジェクトを処理するための、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 PrintObjectjavadoc/com/ibm/as400/access/PrintObject.html#NAVBAR_TOP 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 Javadoc

PrintObjectList Javadoc

OutputQueue Javadoc

Printer Javadoc

PrinterFile Javadoc

SpooledFile Javadoc

WriterJob Javadoc

PrintObject 属性の取り出し:

属性 ID と、基本 PrintObject クラスのいくつかのメソッドの 1 つ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。

使用できるメソッドには、次のようなものがあります。

- 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

SpooledFile 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。

PrintObject Javadoc

SpooledFile Javadoc

PrintParameterList Javadoc

AFP リソース属性:

このトピックでは、AFP リソースの、検索および設定できる属性をリストしています。

属性の検索

該当する getIntegerAttribute()、getStringAttribute()、または getFloatAttribute() メソッドを使って、AFP リソースの以下の属性を検索することができます。

- ATTR_AFP_RESOURCE - AFP リソース統合ファイル・システム・パス
- ATTR_OBJEXTATTR - オブジェクト拡張属性
- ATTR_DESCRIPTION - テキスト記述
- ATTR_DATE - ファイルがオープンされた日付
- ATTR_TIME - ファイルがオープンされた時刻
- ATTR_NUMBYTES - 読み取り/書き込みのバイト数

属性の設定

AFP リソース用に属性を設定することはできません。

出力待ち行列の属性:

このトピックでは、出力待ち行列で利用できる属性をリストしています。

属性の検索

該当する getIntegerAttribute()、getStringAttribute()、または getFloatAttribute() メソッドを使って、出力待ち行列の属性を検索することができます。

- ATTR_AUTHCHCK - 検査権限
- ATTR_DATA_QUEUE - データ待ち行列統合ファイル・システム名
- ATTR_DISPLAYANY - ファイルの表示
- ATTR_JOBSEPRATR - ジョブ区切り
- ATTR_NUMFILES - ファイル数
- ATTR_NUMWRITERS - 待ち行列への書き出しを開始したプログラムの数
- ATTR_OPCNTRL - オペレーター制御
- ATTR_ORDER - 待ち行列上のファイルの順序
- ATTR_OUTPUT_QUEUE - 出力待ち行列の統合ファイル・システム名
- ATTR_OUTQSTS - 出力待ち行列の状況
- ATTR_PRINTER - プリンター
- ATTR_SEPPAGE - 区切りページ
- ATTR_DESCRIPTION - テキスト記述
- 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
- ATTR_USER_TRANSFORM_PROG - ユーザー変換プログラムの統合ファイル・システム名
- ATTR_USER_DRIVER_PROG - ユーザー・ドライバー・プログラムの統合ファイル・システム名
- ATTR_WTRJOBNAME - 書き出しプログラムのジョブ名
- ATTR_WTRJOBNUM - 書き出しプログラムのジョブ番号
- ATTR_WTRJOBSTS - 書き出しプログラムのジョブ状況
- ATTR_WTRJOBUSER - 書き出しプログラムのジョブ・ユーザー名

属性の設定

出力待ち行列の属性を設定することはできません。

プリンターの属性:

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の属性を検索することができます。

属性の検索

- ATTR_AFP - 高機能印刷
- ATTR_ALIGNFORMS - 用紙の位置合わせ

- ATTR_ALWDRTPRT - 直接印刷可能
- ATTR_BTWNCPYSTS - コピー間状況
- ATTR_BTWNFILESTS - ファイル間状況
- ATTR_CODEPAGE - コード・ページ
- ATTR_CHANGES - 変更
- ATTR_DEVCLASS - 装置クラス
- ATTR_DEVMODEL - 装置型式
- ATTR_DEVTYPE - 装置タイプ
- ATTR_DEVSTATUS - 装置状況
- ATTR_DRWRSEP - 区切りページの用紙入れ
- ATTR_ENDPNDSTS - 終了保留中状況
- ATTR_FILESEP - ファイル区切り
- 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_FORMTYPE - 用紙タイプ
- ATTR_FORMTYPEMSG - 用紙タイプ・メッセージ
- ATTR_FORMFEED - 用紙送り
- ATTR_CHAR_ID - 図形文字セット
- ATTR_HELDSTS - 保留中状況
- ATTR_HOLDPNDSTS - 保留保留中状況
- ATTR_JOBUSER - ジョブのユーザー
- ATTR_MFGTYPE - メーカー、機種、および型式
- ATTR_MESSAGE_QUEUE - メッセージ待ち行列の統合ファイル・システム名
- ATTR_ONJOBQSTS - ジョブ待ち行列上状況
- ATTR_OUTPUT_QUEUE - 出力待ち行列の統合ファイル・システム名
- ATTR_OVERALLSTS - 全体的な状況
- ATTR_POINTSIZE - ポイント・サイズ
- ATTR_PRINTER - プリンター
- ATTR_PRTDEVTYPE - 印刷装置タイプ
- ATTR_PUBINF_COLOR_SUP - パブリッシュ情報: カラー・サポート
- ATTR_PUBINF_PPM_COLOR - パブリッシュ情報: 分当たりページ数 (カラー)
- ATTR_PUBINF_PPM - パブリッシュ情報: 分当たりページ数 (モノクローム)
- ATTR_PUBINF_DUPLEX_SUP - パブリッシュ情報: 両面印刷サポート
- ATTR_PUBINF_LOCATION - パブリッシュ情報: ロケーション
- ATTR_RMTLOCNAME - リモート・ロケーション名
- ATTR_SPOOLFILE - スプール・ファイル名
- ATTR_SPLFNUM - スプール・ファイル番号
- ATTR_STARTEDBY - 開始したユーザー
- ATTR_DESCRIPTION - テキスト記述
- 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
- ATTR_USER_TRANSFORM_PROG - ユーザー変換プログラムの統合ファイル・システム名
- ATTR_USER_DRIVER_PROG - ユーザー・ドライバー・プログラムの統合ファイル・システム名
- ATTR_SCS2ASCII - SCS から ASCII への変換
- ATTR_WTNGDATASTS - データ待ち中状況
- ATTR_WTNGDEVSTS - 装置待ち中状況
- ATTR_WTNGMSGSTS - メッセージ待ち中状況
- ATTR_WTRAUTOEND - 書き出しプログラムの自動終了時期
- ATTR_WTRJOBNAME - 書き出しプログラムのジョブ名
- ATTR_WTRJOBSTS - 書き出しプログラムのジョブ状況
- ATTR_WTRSTRTD - 開始済みの書き出しプログラム
- ATTR_WRTNGSTS - 書き出し中状況

属性の設定

プリンターの属性を設定することはできません。

プリンター・ファイルの属性:

このトピックでは、IBM Toolbox for Java で使用するプリンター・ファイル属性のリストを示します。

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 - 1 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 - 計測単位
- 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 - 1 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 - 計測単位
- 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:

このトピックでは、スプール・ファイルの検索および設定可能な属性をリストしています。

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日付
- 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

書き出しプログラム・ジョブ属性:

このトピックでは、書き出しプログラム・ジョブの属性をリストしています。

属性の検索

次の属性は、書き出しプログラム・ジョブについて調べるために、適切な getIntegerAttribute()、getStringAttribute()、または getFloatAttribute() メソッドを使用して検索されます。

- ATTR_WTRJOBNAME - 書き出しプログラムのジョブ名
- ATTR_WTRJOBNUM - 書き出しプログラムのジョブ番号
- ATTR_WTRJOBSTS - 書き出しプログラムのジョブ状況
- ATTR_WTRJOBUSER - 書き出しプログラムのジョブ・ユーザー名

属性の設定

書き出しプログラム用に属性を設定することはできません。

印刷オブジェクトの属性:

このトピックでは、印刷オブジェクトで利用できる属性をリストしています。

- Advanced Function Printing
- AFP リソース
- 用紙の位置合わせ
- ページの位置合わせ
- 直接印刷可能
- 権限
- 検査権限
- 書き出しプログラムの自動終了
- 補助記憶域
- バック・マージン横方向オフセット
- バック・マージン下方向オフセット
- 後部オーバーレイ
- 背面オーバーレイ横方向オフセット
- 背面オーバーレイ下方向オフセット
- コピー間状況
- ファイル間状況
- 変更
- 1 インチ当たりの文字数
- コード・ページ
- コード化フォント名
- コード化フォント・ライブラリー名
- 制御文字
- 行データの変換
- コピー枚数
- 作成されていない残りのコピー
- コーナー・ステープルとじ
- 現在のページ
- データ形式
- データ待ち行列
- ファイルがオープンされた日付
- スプール・ファイル作成ジョブが終了した日付
- 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
- 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
- ユーザー指定の DBCS データ

- DBCS 拡張文字
- DBCS の回転
- インチ当たりの DBCS 数
- DBCS の SO/SI のスペース
- 据え置き書き出し
- ページ回転の角度
- 送信後のファイルの削除
- 宛先オプション
- 宛先タイプ
- 装置クラス
- 装置型式
- 装置状況
- 装置タイプ
- ファイルの表示
- 区切りページの用紙入れ
- 平とじステープル数
- 平とじ参照
- 平とじ参照オフセット
- 終了保留中状況
- 終了ページ
- エンベロープ・ソース
- ファイル区切り
- レコードの折り返し
- フォント ID
- 用紙定義
- 用紙送り
- 用紙タイプ
- 用紙タイプ・メッセージ・オプション
- フロント・マージン横方向オフセット
- フロント・マージン下方向オフセット
- フロント・オーバーレイ
- フロント・オーバーレイ横方向オフセット
- フロント・オーバーレイ下方向オフセット
- 図形文字セット
- ハードウェア行末調整
- 保留中状況
- スプール・ファイル保留
- 保留保留中状況
- イメージ構成
- 書き出しプログラムの初期設定

- インターネット・アドレス
- IPP 属性文字セット
- IPP ジョブ ID
- IPP ジョブ名
- IPP ジョブ名 NL
- IPP ジョブ発信ユーザー名
- IPP ジョブ発信ユーザー名 NL
- IPP プリンター名
- ジョブ名
- ジョブ番号
- ジョブ区切り
- ジョブ・システム
- ジョブのユーザー
- 印刷する最終ページ
- ページの長さ
- ライブラリー名
- 1 インチ当たりの行数
- 行の間隔
- メーカー、機種、および型式
- クライアントあたり最大ジョブ数リスト
- 最大スプール出力レコード数
- 測定方法
- メッセージ・ヘルプ
- メッセージ ID
- メッセージ待ち行列
- メッセージ応答
- メッセージ・テキスト
- メッセージ・タイプ
- メッセージ重大度
- Multi_Item 応答機能
- ネットワーク ID
- ネットワーク印刷サーバー・オブジェクトの属性
- スプール・ファイル内のバイト数
- 読み取り/書き込みのバイト数
- ファイル数
- 待ち行列への書き出しを開始するプログラムの数
- オブジェクト拡張属性
- ジョブ待ち行列上状況
- オープン時刻コマンド
- オペレーター制御

- 待ち行列上のファイルの順序
- 出力ビン
- 出力優先順位
- 出力待ち行列
- 出力待ち行列の状況
- 全体的な状況
- オーバーフロー行番号
- 一度に 1 ページ
- 推定ページ・カウント
- ページ定義
- ページ番号
- 面当たりページ数
- 用紙ソース 1
- 用紙ソース 2
- ペル密度
- ポイント・サイズ
- 印刷精度
- 両面印刷
- 印刷品質
- 印刷順序
- 印刷テキスト
- プリンター
- 割り当てプリンター
- 印刷装置タイプ
- プリンター・ファイル
- プリンター待ち行列
- パブリッシュ情報: カラー・サポート
- パブリッシュ情報: 分当たりページ数 (カラー)
- パブリッシュ情報: 分当たりページ数 (モノクローム)
- パブリッシュ情報: 両面印刷サポート
- パブリッシュ情報: ロケーション
- リモート・ロケーション名
- レコード長
- 出力の削減
- リモート・システム
- 印刷不能文字の置き換え
- 置き換え文字
- 印刷の再始動
- 中とじステープル数
- 中とじ参照

- スプール・ファイルの保管
- シークのオフセット
- シークの起点
- 送信優先順位
- 区切りページ
- ソース用紙入れ
- スプール SCS
- データのスプール
- スプール・ファイル作成の認証メソッド
- スプール・ファイル作成のセキュリティー・メソッド
- スプール・ファイル名
- スプール・ファイル番号
- スプール・ファイル状況
- スプール出力スケジュール
- 開始したユーザー
- 開始ページ
- 作成したシステム
- テキスト記述
- ファイルがオープンされた時刻
- スプール・ファイル作成ジョブが終了した時刻
- 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
- 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
- 合計ページ数
- SCS から ASCII への変換
- 測定単位
- ユーザー注記
- ユーザー・データ
- ユーザー定義データ
- ユーザー定義ファイル
- ユーザー定義オブジェクト
- ユーザー定義オプション
- ユーザー・ドライバー・プログラム・データ
- ユーザー・ドライバー・プログラム
- ユーザー ID
- ユーザー ID アドレス
- ユーザー変換プログラム
- 精度の表示
- VM/MVS クラス
- データ待ち中状況
- 装置待ち中状況

- メッセージ待ち中状況
- 書き出しプログラムの自動終了時期
- 書き出しプログラムの終了時期
- ファイルの保留時期
- ページ幅
- ワークステーション・カスタマイズ・オブジェクト
- 書き出しプログラムのジョブ名
- 書き出しプログラムのジョブ番号
- 書き出しプログラムのジョブ状況
- 書き出しプログラムのジョブ・ユーザー名
- 開始済みの書き出しプログラム
- 書き出しプログラム開始ページ
- 書き出し中状況
- NPS CCSID
- NPS レベル

Advanced Function Printing

ID ATTR_AFP

型 String

説明 このスプール・ファイル外の AFP リソースをこのスプール・ファイルで使用するかどうかを示します。有効な値は *YES と *NO です。

AFP リソース

ID ATTR_AFP_RESOURCE

型 String

説明 外部の AFP (高機能印刷) リソースの統合ファイル・システム・パス。統合ファイル・システム・パスの形式は /QSYS.LIB/library.LIB/resource.type です。ただし *library* は、リソースを収容しているライブラリー、 *resource* はそのリソースの名前、*type* はリソース・タイプです。 *type* の有効な値には、"FNTRSC"、"FORMDF"、"OVL"、"PAGSEG"、および "PAGDFN" が含まれます。

用紙の位置合わせ

ID ATTR_ALIGNFORMS

型 String

説明 用紙位置決めメッセージを送る時刻。有効な値は *WTR、*FILE、*FIRST です。

ページの位置合わせ

ID ATTR_ALIGN

型 String

説明 このスプール・ファイルの印刷の前に用紙位置決めメッセージを送るかどうかを指示します。有効な値は *YES、*NO です。

直接印刷可能

ID ATTR_ALWDRTPRT

型 String

説明 プリンターに直接印刷するジョブに、プリンターを印刷装置書き出しプログラムから割り振ってもよいかどうかを指示します。有効な値は *YES または *NO です。

権限

ID ATTR_AUT

型 String

説明 出力待ち行列に対する特定権限をもたないユーザーに与える権限を指定します。有効な値は *USE、*ALL、*CHANGE、*EXCLUDE、*LIBCRTAUT です。

検査権限

ID ATTR_AUTCHK

型 String

説明 出力待ち行列に対するどのような種類の権限を使って、出力待ち行列のすべてのファイルをユーザーが制御できるかを指示します。有効な値は *OWNER、*DTAAUT です。

書き出しプログラムの自動終了

ID ATTR_AUTOEND

型 String

説明 書き出しプログラムを自動終了しなければならないかどうかを指定します。有効な値は *NO、*YES です。

補助記憶域

ID ATTR_AUX_POOL

型 Integer

説明 スプール・ファイルが保管される補助記憶域プール (ASP) の番号を指定します。使用できる値は、次の通りです。

- 1: システム ASP
- 2から32: ユーザー ASP のいずれか

バック・マージン横方向オフセット

ID ATTR_BACKMGN_ACR

型 Float

説明 用紙の裏面において、ページの左端からどのくらい内側の位置から印刷を開始するかを指定します。特殊値 *FRONTMGN は -1 にエンコードされます。

バック・マージン下方向オフセット

ID ATTR_BACKMGN_DWN

型 Float

説明 用紙の裏面において、ページの上端からどのくらい下の位置から印刷を開始するかを指定します。特殊値 *FRONTMGN は -1 にエンコードされます。

背面オーバーレイ

ID ATTR_BACK_OVERLAY

型 String

説明 背面オーバーレイまたは特殊値の統合ファイル・システム・パス。値が統合ファイル・システム・パスであると、その形式は /QSYS.LIB/library.LIB/overlay.OVL となります。ただし *library* はリソースのライブラリー、*overlay* はオーバーレイの名前です。有効な特殊値には *FRONTOVL などがあります。

背面オーバーレイ横方向オフセット

ID ATTR_BKOVL_ACR

型 Float

説明 オーバーレイを印刷するための、起点からの横方向のオフセット。

背面オーバーレイ下方向オフセット

ID ATTR_BKOVL_DWN

型 Float

説明 オーバーレイを印刷するための、起点からの下方向のオフセット。

コピー間状況

ID ATTR_BTWNCPYSTS

型 String

説明 複数コピーのスプール・ファイルのそれぞれのコピー間に書き出しプログラムがあるかどうか。戻り値は *YES または *NO です。

ファイル間状況

ID ATTR_BTWNFILESTS

型 String

説明 書き出しプログラムがファイル間にあるかどうか。戻り値は *YES または *NO です。

変更

ID ATTR_CHANGES

型 String

説明 保留変更中がいつ有効化されるか。有効な値は *NORDYF、*FILEEND、またはブランクですが、ブランクの場合、書き出しプログラムに対する保留中の変更はないという意味です。

1 インチ当たりの文字数

ID ATTR_CPI

型 Float

説明 横方向の 1 インチあたりの文字数。

コード・ページ

ID ATTR_CODEPAGE

型 String

説明 このスプール・ファイル用のコード点に対するグラフィック文字のマッピング。グラフィック文字セット・フィールドに特殊値が入っている場合、このフィールドにゼロ (0) を入れてもかまいません。

コード化フォント名

ID ATTR_CODEDFNT

型 String

説明 コード化フォントの名前。コード化フォントとは、文字セットとコード・ページから成る AFP リソースです。特殊値には *FNTCHRSET などがあります。

コード化フォント・ライブラリー名

ID ATTR_CODEDFNTLIB

型 String

説明 コード化フォントの入ったライブラリーの名前。コード化フォント・フィールドに特殊値が入っていれば、このフィールドにブランクを入れてもかまいません。

制御文字

ID ATTR_CONTROLCHAR

型 String

説明 このファイルで米国標準規格の印刷制御文字を使用するかどうかを指定します。指定できる値は、印刷対象のデータに印刷制御文字を渡さない場合は *NONE、各レコードの先頭文字を米国標準規格印刷制御文字にする場合は *FCFC です。

行データの変換

ID ATTR_CONVERT_LINEDATA

型 String

説明 スプールに書き込む前に行データを AFPDS に変換するかどうか。指定できる値は *NO と *YES です。

コピー枚数

ID ATTR_COPIES

型 Integer

説明 作成するこのスプール・ファイルのコピーの合計数。

作成されていない残りのコピー

ID ATTR_COPIESLEFT

型 Integer

説明 作成するこのスプール・ファイル・コピーの残りのコピー部数。

コーナー・ステープルとじ

ID ATTR_CORNER_STAPLE

型 String

説明 コーナー・ステープルとじに使用する基準コーナー。ステープルは、基準コーナーでメディアに装着されます。有効な値は *NONE、*DEVD、*BOTRIGHT、*TOPRIGHT、*TOPLEFT、および *BOTLEFT です。

現在のページ

ID ATTR_CURPAGE

型 Integer

説明 書き出しプログラム・ジョブで現在書き込まれているページ。

データ形式

ID ATTR_DATAFORMAT

型 String

説明 データ形式。有効な値は *RCDDATA、*ALLDATA です。

データ待ち行列

ID ATTR_DATA_QUEUE

型 String

説明 データ待ち行列が出力待ち行列に関連付けられていない場合に、出力待ち行列または *NONE に関連付けられているデータ待ち行列の統合ファイル・システム・パスを指定します。統合ファイル・システム・パスの形式は /QSYS.LIB/library.LIB/dataqueue.DTAQ ですが、 *library* はデータ待ち行列の入ったライブラリー、 *dataqueue* はそのデータ待ち行列の名前です。

ファイルがオープンされた日付

ID ATTR_DATE

型 String

説明 スプール・ファイルの場合、これはそのスプール・ファイルがオープンされた日付です。 AFP リソースの場合、これはそのオブジェクトが最後に変更された日付です。日付は、C YY MM DD という形式の文字ストリングにエンコードされます。

スプール・ファイル作成ジョブが終了した日付

ID ATTR_DATE_END

型 String

説明 システム上でスプール・ファイルを作成したジョブが終了した日付。「スプール・ファイル作成開始日付」フィールドが *ALL に設定されている場合、このフィールドをブランクに設定する必要があります。「スプール・ファイル作成開始日付」フィールドに日付が設定されている場合には、このフィールドには有効な日付を設定しなければなりません。日付は CYYMMDD 形式、または以下の特殊値のいずれかであることが必要です。

- *LAST: 作成日が、スプール・ファイル作成開始日付以降のすべてのスプール・ファイルが戻されます。

- 日付: 作成日時が、スプール・ファイル作成開始日付以降で、なおかつスプール・ファイル作成終了日付以前のすべてのスプール・ファイルが戻されます。

日付形式 CYYMMDD は以下のように定義されています。

- C は世紀で、0 は 19xx 年を、1 は 20xx 年を表す
- YY は年
- MM は月
- DD は日

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

ID ATTR_DATE_WTR_BEGAN_FILE

型 String

説明 書き出しプログラムがこのスプール・ファイルの処理を開始した日付を示します。日付は、C YY MM DD という形式の文字ストリングにエンコードされます。

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

ID ATTR_DATE_WTR_CMPL_FILE

型 String

説明 書き出しプログラムがこのスプール・ファイルの処理完了を開始した日付を示します。日付は、C YY MM DD という形式の文字ストリングにエンコードされます。

ユーザー指定の DBCS データ

ID ATTR_DBCSDATA

型 String

説明 2 バイト文字セット (DBCS) データがスプール・ファイルに入っているかどうか。有効な値は *NO と *YES です。

DBCS 拡張文字

ID ATTR_DBCSEXTENSN

型 String

説明 DBCS 拡張文字をシステムで処理するかどうかを指定します。有効な値は *NO と *YES です。

DBCS の回転

ID ATTR_DBCAROTATE

型 String

説明 印刷前に DBCS 文字を左回りに 90 度回転させるかどうか。有効な値は *NO と *YES です。

インチ当たりの DBCS 数

ID ATTR_DBCSCPI

型 Integer

説明 1 インチあたりに印刷する 2 バイト文字数。有効な値は -1、-2、5、6、および 10 です。値 *CPI は -1 にエンコードされます。値 *CONDENSED は -2 にエンコードされます。

DBCS の SO/SI のスペース

ID ATTR_DBCSSISO

型 String

説明 印刷時のシフトアウト文字とシフトイン文字の表示を指定します。有効な値は *NO、*YES、および *RIGHT です。

据え置き書き出し

ID ATTR_DFR_WRITE

型 String

説明 印刷データを事前にシステム・バッファーに入れるかどうか。

ページ回転の角度

ID ATTR_PAGRTT

型 Integer

説明 プリンターへの用紙のロード時のページ上のテキストの回転の角度。有効な値は -1、 -2、 -3、 0、 90、 180、 270 です。値 *AUTO は -1 にエンコードされ、値 *DEVD は -2 にエンコードされ、値 *COR は -3 にエンコードされます。

送信後のファイルの削除

ID ATTR_DELETESPLF

型 String

説明 送信後にスプール・ファイルを削除するかどうか。有効な値は *NO または *YES です。

宛先オプション

ID ATTR_DESTOPTION

型 String

説明 宛先オプション。受信側システムにオプションを渡すのにユーザーが使用できるテキスト・ストリング。

宛先タイプ

ID ATTR_DESTINATION

型 String

説明 宛先タイプ。有効な値は *OTHER、*AS400、*PSF2 です。

装置クラス

ID ATTR_DEVCLASS

型 String

説明 装置クラス。

装置型式

ID ATTR_DEVMODEL

型 String

説明 装置の型式番号。

装置状況

ID ATTR_DEVSTATUS

型 Integer

説明 印刷装置の状況。有効な値は 0 (オフに構成変更)、10 (オフに構成変更保留中)、20 (オンに構成変更保留中)、30 (オンに構成変更) 、40 (接続保留中)、60 (活動状態)、66 (書き出しプログラムの活動中)、70 (保留中)、75 (電源オフ)、80 (回復保留)、90 (回復取り消し状態)、100 (失敗)、106 (書き出しプログラム障害)、110 (保守中)、111 (損傷)、112 (ロック)、113 (不明) です。

装置タイプ

ID ATTR_DEVTYPE

型 String

説明 装置タイプ。

ファイルの表示

ID ATTR_DISPLAYANY

型 String

説明 この出力待ち行列を読み取る権限をもったユーザーが、この待ち行列上のどの出力ファイルのデータ出力でも表示できるか、それともユーザー自身のファイル内でしか表示できないか。有効な値は *YES、*NO、*OWNER です。

区切りページの用紙入れ

ID ATTR_DRWRSEP

型 Integer

説明 取り出すジョブおよびファイル区切りのページが入っている用紙入れを指定します。有効な値は -1、-2、1、2、3 です。値 *FILE は -1 にエンコードされ、値 *DEVD は -2 にエンコードされます。

平とじステープル数

ID ATTR_EDGESTITCH_NUMSTAPLES

型 Integer

説明 終了操作軸に沿って装着されるステープル数。

平とじ参照

ID ATTR_EDGESTITCH_REF

型 String

説明 終了操作軸に沿ってメディアに装着される 1 つ以上のステープル。有効な値は *NONE、*DEVD、 *BOTTOM、 *RIGHT、 *TOP、および *LEFT です。

平とじ参照オフセット

ID ATTR_EDGESTITCH_REFOFF

型 Float

説明 メディアの中央方向への基準縁からの平とじオフセット。

終了保留中状況

ID ATTR_ENDPNDSTS

型 String

説明 この書き出しプログラムに対して書き出しプログラム終了 (ENDWTR) コマンドを出すかどうか。指定できる値は、*NO (ENDWTR コマンドを出さない)、*IMMED (出力バッファーが空になりしだい書き出しプログラムは終了する)、 *CTRLD (スプール・ファイルの現在のコピーの印刷が完了したら書き出しプログラムは終了する)、 *PAGEEND (ページの末尾で書き出しプログラムは終了する) です。

終了ページ

ID ATTR_ENDPAGE

型 Integer

説明 スプール・ファイルの印刷を終了するページ番号。有効な値は 0 または終了ページ番号です。値 *END は 0 にエンコードされます。

エンベロープ・ソース

ID ATTR_ENVLP_SOURCE

型 String

説明 エンベロープ・ソース内のエンベロープのサイズ。このフィールドを指定しないか、または値が有効でないと、特殊値 *MFRTYPMDL が使われます。有効な値は *NONE (エンベロープ・ソースはなし)、*MFRTYPMDL (メーカー、機種、および型式に指示されているエンベロープ・サイズが使用される)、*MONARCH (3.875 x 7.5 インチ)、 *NUMBER9 (3.875 x 8.875 インチ)、*NUMBER10 (4.125 x 9.5 インチ)、*B5 (176mm x 250mm)、*C5 (162mm x 229mm)、*DL (110mm x 220mm) です。

ファイル区切り

ID ATTR_FILESEP

型 Integer

説明 スプール・ファイルの各コピーの先頭に置かれるファイル区切りページ数。有効な値は -1、または区切りの数です。値 *FILE は -1 とエンコードされます。

レコードの折り返し

ID ATTR_FOLDREC

型 String

説明 プリンターの用紙の幅を超えるレコードを次行に折り返すかどうか。有効な値は *YES、*NO です。

フォント ID

ID ATTR_FONTID

型 String

説明 使用するプリンター・フォント。有効な特殊値には *CPI および *DEVD などがあります。

用紙定義

ID ATTR_FORM_DEFINITION

型 String

説明 用紙定義または特殊値の統合ファイル・システム・パス名。統合ファイル・システム・パスを指定する場合、その形式は /QSYS.LIB/library.LIB/formdef.FORMDF です。ただし、*library* は用紙定義のライブラリー、 *formdef* は用紙定義の名前です。有効な特殊値には *NONE、 *INLINE、*INLINED、 *DEVD などがあります。

用紙送り

ID ATTR_FORMFEED

型 String

説明 用紙をプリンターに送り込む方式。有効な値は *CONT、*CUT、*AUTOCUT、*DEVD です。

用紙タイプ

ID ATTR_FORMTYPE

型 String

説明 このスプール・ファイルの印刷のためにプリンターにロードする用紙のタイプ。

用紙タイプ・メッセージ・オプション

ID ATTR_FORMTYPEMSG

型 String

説明 現行用紙タイプが完了したら書き出しプログラムのメッセージ待ち行列にメッセージを送るためのメッセージ・オプション。有効な値は *MSG、*NOMSG、*INFOMSG、*INQMSG です。

フロント・マージン横方向オフセット

ID ATTR_FTMGN_ACR

型 Float

説明 用紙のおもて面において、ページの左端からどのくらい内側の位置から印刷を開始するかを指定します。特殊値 *DEVD は -2 にエンコードされます。

フロント・マージン下方向オフセット

ID ATTR_FTMGN_DWN

型 Float

説明 用紙のおもて面において、ページの上端からどのくらい下の位置から印刷を開始するかを指定します。特殊値 *DEVD は -2 にエンコードされます。

フロント・オーバーレイ

ID ATTR_FRONT_OVERLAY

型 String

説明 フロント・オーバーレイの統合ファイル・システム・パス。統合ファイル・システム・パスの形式

は /QSYS.LIB/library.LIB/overlay.OVL となります。ただし *library* はリソースのライブラリー、*overlay* はオーバーレイの名前です。フロント・オーバーレイを指定しないことを指示するには文字列 "*NONE" を使います。

フロント・オーバーレイ横方向オフセット

ID ATTR_FTOVL_ACR

型 Float

説明 オーバーレイを印刷するための、起点からの横方向のオフセット。

フロント・オーバーレイ下方向オフセット

ID ATTR_FTOVL_DWN

型 Float

説明 オーバーレイを印刷するための、起点からの下方向のオフセット。

図形文字セット

ID ATTR_CHAR_ID

型 String

説明 このファイルの印刷時に使用する一連のグラフィック文字セット。有効な特殊値には *DEVD、*SYSVAL、*JOBCCSID などがあります。

ハードウェア行末調整

ID ATTR_JUSTIFY

型 Integer

説明 出力を右そろえするパーセント。有効な値は 0、50、100 です。

保留中状況

ID ATTR_HELDSTS

型 String

説明 書き出しプログラムを保留にするかどうか。有効な値は *YES、*NO です。

スプール・ファイル保留

ID ATTR_HOLD

型 String

説明 スプール・ファイルを保留にするかどうか。有効な値は *YES、*NO です。

保留保留中状況

ID ATTR_HOLDPNDSTS

型 String

説明 この書き出しプログラムに対して書き出しプログラム保留 (HLDWTR) コマンドを出したかどうか。指定できる値は、*NO (HLDWTR コマンドを出していない)、*IMMED (出力バッファーが空になったら書き出しプログラムは保留になる)、 *CTRLD (スプール・ファイルの現在のコピーの印刷が完了したら書き出しプログラムは保留になる)、 *PAGEEND (ページの末尾で書き出しプログラムは保留になる) です。

イメージ構成

ID ATTR_IMGCFG

型 String

説明 さまざまなイメージおよび印刷のデータ・ストリーム形式のための変換サービス。

書き出しプログラムの初期設定

ID ATTR_WTRINIT

型 String

説明 ユーザーは、印刷装置をいつ初期設定するかを指定することができます。有効な値は *WTR、*FIRST、 *ALL です。

インターネット・アドレス

ID ATTR_INTERNETADDR

型 String

説明 受信側システムのインターネット・アドレス。

IPP 属性文字セット

ID ATTR_IPP_ATTR_CHARSET

型 String

説明 IPP 指定のスプール・ファイル属性の charset (コード化文字セットとエンコード方式) を示します。

IPP ジョブ ID

ID ATTR_IPP_JOB_ID

型 Integer

説明 ジョブを作成した IPP プリンターに関連した IPP ジョブ ID。

IPP ジョブ名

ID ATTR_IPP_ATR_CHARSET

型 String

説明 ユーザーにとって分かりやすいジョブの名前。

IPP ジョブ名 NL

ID ATTR_IPP_JOB_NAME_NL

型 String

説明 ジョブ名の自然言語。

IPP ジョブ発信ユーザー名

ID ATTR_IPP_JOB_ORIGUSER

型 String

説明 この IPP ジョブを実行依頼したエンド・ユーザーを識別します。

IPP ジョブ発信ユーザー名 NL

ID ATTR_IPP_JOB_ORIGUSER_NL

型 String

説明 ジョブから発信されたユーザー名の自然言語を識別します。

IPP プリンター名

ID ATTR_IPP_PRINTER_NAME

型 String

説明 この IPP ジョブを作成した IPP プリンターを識別します。

ジョブ名

ID ATTR_JOBNAME

型 String

説明 スプール・ファイルを作成したジョブの名前。

ジョブ番号

ID ATTR_JOBNUMBER

型 String

説明 スプール・ファイルを作成したジョブの番号。

ジョブ区切り

ID ATTR_JOBSEPRATR

型 Integer

説明 この出力待ち行列上にスプール・ファイルを保有している各ジョブの先頭に置かれるジョブ区切り記号の数。有効な値は -2、0 から 9 です。値 *MSG は -2 にエンコードされます。出力待ち行列の作成時にジョブ区切りが指定されます。

ジョブ・システム

ID ATTR_JOBSYSTEM

型 String

説明 スプール・ファイルを作成したジョブが実行されていたシステム。

ジョブのユーザー

ID ATTR_JOBUSER

型 String

説明 スプール・ファイルを作成したユーザーの名前。

印刷する最終ページ

ID ATTR_LASTPAGE

型 Integer

説明 ジョブの処理が完了する前に印刷が終了した場合、最後に印刷されたページの番号はファイルです。

ページの長さ

ID ATTR_PAGELEN

型 Float

説明 ページの長さ。測定単位は測定メソッド属性に指定します。

ライブラリー名

ID ATTR_LIBRARY

型 String

説明 ライブラリーの名前。

1 インチ当たりの行数

ID ATTR_LPI

型 Float

説明 スプール・ファイルでの縦方向の 1 インチあたりの行数。

行の間隔

ID ATTR_LINESPACING

型 String

説明 印刷時のファイルの行データ・レコードのスペースの取り方。情報が戻されるのは、*LINE と *AFPDSLINE の印刷装置タイプ・ファイルの場合だけです。有効な値は *SINGLE、 *DOUBLE、*TRIPLE、 *CTLCHAR です。

メーカー、機種、および型式

ID ATTR_MFGTYPE

型 String

説明 印刷データを SCS から ASCII に変換するときのメーカー、機種、および型式を指定します。

クライアントあたり最大ジョブ数リスト

ID ATTR_MAX_JOBS_PER_CLIENT

型 Integer

説明 プリンター待ち行列のサイズの上限を示すためにクライアントから指示されます。

最大スプール出力レコード数

ID ATTR_MAXRECORDS

型 Integer

説明 このファイルのオープン時にこのファイルに収容できるレコードの最大数。値 *NOMAX は 0 とエンコードされます。

測定方法

ID ATTR_MEASMETHOD

型 String

説明 ページの長さとページ幅の属性に使う測定方法。有効な値は *ROWCOL、*UOM です。

メッセージ・ヘルプ

ID ATTR_MSGHELP

型 char(*)

説明 2 次レベル・テキストと呼ばれることもあるメッセージ・ヘルプは、「メッセージの検索」要求で戻すことができます。システムしきい値は 3000 文字までです (英語版で変換するにはこれより 30 % 少なくしなければなりません)。

メッセージ ID

ID ATTR_MESSAGEID

型 String

説明 メッセージ ID。

メッセージ待ち行列

ID ATTR_MESSAGE_QUEUE

型 String

説明 書き出しプログラムが操作可能メッセージ用に使用するメッセージ待ち行列の統合ファイル・システム・パス。統合ファイル・システム・パスの形式は /QSYS.LIB/library.LIB/messageque.MSGQ ですが、 *library* はメッセージ待ち行列の入ったライブラリー、 *messageque* はそのメッセージ待ち行列の名前です。

メッセージ応答

ID ATTR_MSGREPLY

型 String

説明 メッセージ応答。タイプが「inquiry」のメッセージに応答するクライアントが指定するテキスト・ストリング。メッセージの検索の場合、属性値はサーバーから戻され、そこにはクライアントが使用できるデフォルトの応答が入っています。システムしきい値では、長さは 132 文字までです。可変長が原因で、NULL 文字で終了しなければなりません。

メッセージ・テキスト

ID ATTR_MSGTEXT

型 String

説明 1 次レベル・テキストと呼ばれることもあるメッセージ・テキストは、「メッセージの検索」要求で戻すことができます。システムしきい値では、長さは 132 文字までです。

メッセージ・タイプ

ID ATTR_MSGTYPE

型 String

説明 EBCDIC エンコードかつ 2 桁のメッセージ・タイプ。以下の 2 種類のメッセージは、「検索した」メッセージに対して「応答」できるかどうかを示します。つまり '04' 通知メッセージは、応答の必要のない (ただし訂正処置が必要なことはあります) 情報を伝え、 '05' 照会メッセージは、応答を必要とする情報を伝えます。

メッセージ重大度

ID ATTR_MSGSEV

型 Integer

説明 メッセージ重大度。値の範囲は 00 から 99 です。この値が高ければ高いほど、条件はより厳重または重大になります。

Multi_Item 応答機能

ID ATTR_MULTI_ITEM_REPLY

型 String

説明 クライアントがこの属性値を *YES に設定すると、リストのスプール・ファイルの操作のパフォーマンスを大きく改善することができます。デフォルト値は *NO です。

ネットワーク ID

ID ATTR_NETWORK

型 String

説明 ファイルが作成されたシステムのネットワーク ID。

スプール・ファイル内のバイト数

ID ATTR_NUMBYTES_SPLF

型 Integer

説明 ストリームまたはスプール・ファイル内で使うことのできるバイトの合計数。この値は、データ変換を行う前のバイト数を示します。 2**31 - 1 バイトより大きいサイズのファイルを収容するためにこの値はスケール済みになっています。実際のバイト数を得るにはこの値に 10K を掛ける必要があります。一度に 1 ページ・モードで表示されるスプール・ファイルに対してはこの属性は有効ではありません。

読み取り/書き込みのバイト数

ID ATTR_NUMBYTES

型 Integer

説明 読み取り操作で読み取るバイト数、または書き込み操作で書き込むバイト数。オブジェクト・アクションによってこの属性の解釈法が決まります。

ファイル数

ID ATTR_NUMFILES

型 Integer

説明 出力待ち行列上に存在するスプール・ファイルの数。

待ち行列への書き出しを開始するプログラムの数

ID ATTR_NUMWRITERS

型 Integer

説明 出力待ち行列を対象に開始された書き出しプログラム・ジョブの数。

オブジェクト拡張属性

ID ATTR_OBJEXTATTR

型 String

説明 フォント・リソースなどの特定のオブジェクトによって使われる「拡張」属性。この値は、サーバー上で WRKOBJ コマンドと DSPOBJD コマンドを使って表示します。 サーバーの画面上の表題は、「Attribute (属性)」としか示されないことがあります。たとえばフォント・リソースといったオブジェクト・タイプの場合、共通値は CDEPAG、CDEFNT、FNTCHRSET などになります。

ジョブ待ち行列上状況

ID ATTR_ONJOBQSTS

型 String

説明 書き出しプログラムがジョブ待ち行列上にあるために現在は稼働していないかどうか。指定できる値は *YES、*NO です。

オープン時刻コマンド

ID ATTR_OPENCMDS

型 String

説明 データ・ストリーム内のスプール・ファイル・データの前に SCS オープン時刻コマンドをユーザーが挿入したいかどうかを指定します。有効な値は *YES、*NO です。

オペレーター制御

ID ATTR_OPCNTRL

型 String

説明 ジョブ制御権限をもったユーザーが、この待ち行列上のスプール・ファイルを管理または制御できるかどうか。有効な値は *YES、*NO です。

待ち行列上のファイルの順序

ID ATTR_ORDER

型 String

説明 この出力待ち行列上のスプール・ファイルの順序。有効な値は *FIFO、*JOBNBR です。

出力ビン

ID ATTR_OUTPUTBIN

型 Integer

説明 印刷出力でプリンターが使用する出力ビン。値の範囲は 1 から 65535 です。値 *DEVD は 0 とエンコードされます。

出力優先順位

ID ATTR_OUTPTY

型 String

説明 スプール・ファイルの優先順位。優先順位の範囲は 1 (最高) から 9 (最低) です。有効な値は 0 から 9 ですが、0 は *JOB を表します。

出力待ち行列

ID ATTR_OUTPUT_QUEUE

型 String

説明 出力待ち行列の統合ファイル・システム・パス。統合ファイル・システム・パスの形式は /QSYS.LIB/library.LIB/queue.OUTQ ですが、 *library* は出力待ち行列の入ったライブラリー、*queue* はその出力待ち行列の名前です。

出力待ち行列の状況

ID ATTR_OUTQSTS

型 String

説明 出力待ち行列の状況。有効な値は RELEASED、HELD です。

全体的な状況

ID ATTR_OVERALLSTS

型 Integer

説明 「論理プリンター」の全体的な状況。「論理プリンター」とは、印刷装置、出力待ち行列、および書き出しプログラム・ジョブを指します。有効な値は 1 (使用不能)、2 (電源オフまたはまだ使用可能ではない)、3 (停止)、4 (メッセージ待機中)、 5 (保留中)、6 (停止保留中)、7 (保留保留中)、8 (プリンターの待機中)、9 (始動待機中)、10 (印刷中)、 11 (出力待ち行列待機中)、12 (接続保留中)、13 (電源オフ)、14 (利用不能)、15 (保守中)、999 (不明) です。

オーバーフロー行番号

ID ATTR_OVERFLOW

型 Integer

説明 次ページにオーバーフローされる前に最後に印刷データが印刷される行。

一度に 1 ページ

ID ATTR_PAGE_AT_A_TIME

型 String

説明 スプール・ファイルを一度に 1 ページ・モードでオープンするかどうかを指定します。有効な値は *YES と *NO です。

推定ページ・カウント

ID ATTR_PAGES_EST

型 String

説明 ページ・カウントを実際の数ではなく見積もりにするかどうかを指定します。有効な値は *YES と *NO です。

ページ定義

ID ATTR_PAGE_DEFINITION

型 String

説明 ページ定義または特殊値の統合ファイル・システム・パス名。統合ファイル・システム・パスを指定する場合、その形式は "/QSYS.LIB/library.LIB/pagedef.PAGDFN" です。ただし、*library* はページ定義のライブラリー、 *pagedef* はそのページ定義の名前です。有効な特殊値には *NONE などがあります。

ページ番号

ID ATTR_PAGENUMBER

型 Integer

説明 一度に 1 ページ・モードでオープンされたスプール・ファイルから読み取るページ番号。

面当たりページ数

ID ATTR_MULTIUP

型 Integer

説明 ファイルの印刷時に各物理ページのそれぞれの面に印刷する論理ページの数。有効な値は 1、2、4 です。

用紙ソース 1

ID ATTR_PAPER_SOURCE_1

型 String

説明 用紙ソース 1 内の用紙のサイズ。このフィールドを指定しないか、または値が有効でないと、特殊値 *MFRTYPMDL が使われます。有効な値は、*NONE (用紙ソース 1 がないか、または手動でプリンターに給紙する)、 *MFRTYPMDL (メーカー、機種、および型式で指示された用紙サイズを使用する)、 *LETTER (8.5 x 11.0 インチ)、*LEGAL (8.5 x 14.0 インチ)、 *EXECUTIVE (7.25 x 10.5 インチ)、*LEDGER (17.0 x 11.0 インチ)、 *A3 (297mm x 420mm)、*A4 (210mm x 297mm)、*A5 (148mm x 210mm)、*B4 (257mm x 364mm)、*B5 (182mm x 257mm)、 *CONT80 (8.0 インチ幅の連続給紙)、*CONT132 (13.2 インチ幅の連続給紙) です。

用紙ソース 2

ID ATTR_PAPER_SOURCE_2

型 String

説明 用紙ソース 2 内の用紙のサイズ。このフィールドを指定しないか、または値が有効でないと、特殊値 *MFRTYPMDL が使われます。有効な値は、*NONE (用紙ソース 2 がないか、または手動でプリンターに給紙する)、 *MFRTYPMDL (メーカー、機種、および型式で指示された用紙サイズを使用する)、 *LETTER (8.5 x 11.0 インチ)、*LEGAL (8.5 x 14.0 インチ)、 *EXECUTIVE (7.25 x 10.5 インチ)、*LEDGER (17.0 x 11.0 インチ)、 *A3 (297mm x 420mm)、*A4 (210mm x 297mm)、*A5 (148mm x 210mm)、*B4 (257mm x 364mm)、*B5 (182mm x 257mm)、 *CONT80 (8.0 インチ幅の連続給紙)、*CONT132 (13.2 インチ幅の連続給紙) です。

ペル密度

ID ATTR_PELDENSITY

型 String

説明 フォント・リソースの場合のみこの値は、ペル数にエンコードされます ("1" は 240 のペル・サイズを表し、 "2" は 320 のペル・サイズを表します)。サーバーで定義すれば、これら以外の値も有効にすることができます。

ポイント・サイズ

ID ATTR_POINTSIZE

型 Float

説明 スプール・ファイルのテキストを印刷するポイント・サイズ。特殊値 *NONE は 0 にエンコードされます。

印刷精度

ID ATTR_FIDELITY

型 String

説明 印刷時に実行されるある種のエラー処理。有効な値は *ABSOLUTE、*CONTENT です。

両面印刷

ID ATTR_DUPLEX

型 String

説明 情報の印刷の仕方。有効な値は *FORMDF、*NO、*YES、*TUMBLE です。

印刷品質

ID ATTR_PRTQUALITY

型 String

説明 このスプール・ファイルを印刷するのに適用する印刷品質。有効な値は *STD、 *DRAFT、*NLQ、 *FASTDRAFT です。

印刷順序

ID ATTR_PRTSEQUENCE

型 String

説明 印刷順序。有効な値は *NEXT です。

印刷テキスト

ID ATTR_PRTTEXT

型 String

説明 印刷出力の各ページの下端と区切りページに印刷するテキスト。有効な特殊値には *BLANK および *JOB などがあります。

プリンター

ID ATTR_PRINTER

型 String

説明 印刷装置の名前。

割り当てプリンター

ID ATTR_PRTASSIGNED

型 String

説明 プリンターが割り当て済みかどうかを示します。有効な値は 1 (特定のプリンターに割り当て済み)、2 (複数のプリンターに割り当て済み)、3 (未割り当て) です。

印刷装置タイプ

ID ATTR_PRTDEVTYPE

型 String

説明 プリンターのデータ・ストリームのタイプ。有効な値は *SCS、 *IPDS、 *USERASCII、 *AFPDS、 *LINE です。

プリンター・ファイル

ID ATTR_PRINTER_FILE

型 String

説明 プリンター・ファイルの統合ファイル・システム・パス。統合ファイル・システム・パスの形式は /QSYS.LIB/library.LIB/printerfile.FILE ですが、 *library* はプリンター・ファイルの入ったライブラリー、 *printerfile* はそのプリンター・ファイルの名前です。

プリンター待ち行列

ID ATTR_RMTPRTQ

型 String

説明 SNDTCPSPLF (LPR) を使ってスプール・ファイルを送るときの宛先プリンター待ち行列の名前。

パブリッシュ情報: カラー・サポート

ID ATTR_PUBINF_COLOR_SUP

型 String

説明 このパブリッシュ・リスト項目でカラーがサポートされることを示します。

パブリッシュ情報: 分当たりページ数 (カラー)

ID ATTR_PUBINF_PPM_COLOR

型 Integer

説明 このパブリッシュ・リスト項目のカラー・モードでサポートされる分当たりのページ数。

パブリッシュ情報: 分当たりページ数 (モノクローム)

ID ATTR_PUBINF_PPM

型 Integer

説明 このパブリッシュ・リスト項目のモノクロームでサポートされる分当たりのページ数。

パブリッシュ情報: 両面印刷サポート

ID ATTR_PUBINF_DUPLEX_SUP

型 String

説明 このパブリッシュ・リスト項目での両面印刷サポート標識。

パブリッシュ情報: ロケーション

ID ATTR_PUBINF_LOCATION

型 String

説明 このパブリッシュ・リスト項目のロケーションの説明。

リモート・ロケーション名

ID ATTR_RMTLOCNAME

型 String

説明 印刷装置のロケーション名。

レコード長

ID ATTR_RECLENGTH

型 Integer

説明 レコード長。

出力の削減

ID ATTR_REDUCE

型 String

説明 物理ページの両面に複数の論理ページを印刷する方法。有効な値は *TEXT または ???? です。

リモート・システム

ID ATTR_RMTSYSTEM

型 String

説明 リモート・システム名。有効な特殊値には *INTNETADR などがあります。

印刷不能文字の置き換え

ID ATTR_RPLUNPRT

型 String

説明 印刷不能文字を別の文字に置き換えるかどうか。有効な値は *YES または *NO です。

置き換え文字

ID ATTR_RPLCHAR

型 String

説明 任意の印刷不能文字に置き換える文字。

印刷の再始動

ID ATTR_RESTART

型 Integer

説明 印刷を再始動します。有効な値は -1、-2、-3、または再始動するページ番号 です。値 *STRPAGE は -1 に、値 *ENDPAGE は -2 に、値 *NEXT は -3 にエンコードされます。

中とじステープル数

ID ATTR_SADDLESTITCH_NUMSTAPLES

型 Integer

説明 終了操作軸に沿って装着されるステープル数。

中とじ参照

ID ATTR_SADDLESTITCH_REF

型 String

説明 終了操作軸に沿って 1 つ以上のステープルがメディアに装着されます。なおこの軸は、メディアの中央に、しかも基準縁に対して並行に置かれます。有効な値は *NONE、*DEVD、*TOP、および *LEFT です。

スプール・ファイルの保管

ID ATTR_SAVESPLF

型 String

説明 書き込み後にスプール・ファイルを保管するかどうか。有効な値は *YES、*NO です。

シークのオフセット

ID ATTR_SEEKOFF

型 Integer

説明 シークのオフセット。シークの起点に対して相対的な正および負のどちらの値でも指定できます。

シークの起点

ID ATTR_SEEKORG

型 Integer

説明 有効な値には 1 (先頭または最上部)、2 (現在位置)、および 3 (末尾または最下部) などがあります。

送信優先順位

ID ATTR_SENDPTY

型 String

説明 送信優先順位。有効な値は *NORMAL、*HIGH です。

区切りページ

ID ATTR_SEPPAGE

型 String

説明 バナー・ページを印刷するかどうかのオプションをユーザーが指定できます。有効な値は *YES または *NO です。

ソース用紙入れ

ID ATTR_SRCDRWR

型 Integer

説明 自動用紙裁断給紙オプションを選択したときに使用する用紙入れ。有効な値は -1、 -2、 1 から 255 です。値 *E1 は -1 に、値 *FORMDF は -2 にエンコードされます。

スプール SCS

ID ATTR_SPLSCS

型 Long

説明 スプール・ファイルの作成中の SCS データの使用法を指定します。

データのスプール

ID ATTR_SPOOL

型 String

説明 印刷装置用の出力データをスプール化するかどうか。有効な値は *YES、*NO です。

スプール・ファイル作成の認証メソッド

ID ATTR_SPLF_AUTH_METHOD

型 Integer

説明 このスプール・ファイルの作成で使用されるクライアント認証メソッドを指示します。有効な値には x'00'(*NONE)、x'01'(*REQUESTER)、x'02'(*BASIC)、x'03'(*CERTIFICATE)、および 'x'04'(*DIGEST) などがあります。

スプール・ファイル作成のセキュリティー・メソッド

ID ATTR_SPLF_SECURITY_METHOD

型 String

説明 このスプール・ファイルの作成で使用されるセキュリティー・メソッドを指示します。有効な値は x'00'(*NONE)、x'01'(*SSL3)、および x'02'(*TLS) です。

スプール・ファイル名

ID ATTR_SPOOLFILE

型 String

説明 スプール・ファイルの名前。

スプール・ファイル番号

ID ATTR_SPLFNUM

型 Integer

説明 スプール・ファイル番号。許可される特殊値は、-1 および 0 です。値 *LAST は -1 に、値 *ONLY は 0 にエンコードされます。

スプール・ファイル状況

ID ATTR_SPLFSTATUS

型 String

説明 スプール・ファイルの状況。有効な値は *CLOSED、*HELD、 *MESSAGE、 *OPEN、 *PENDING、 *PRINTER、 *READY、 *SAVED、 *WRITING です。

スプール出力スケジュール

ID ATTR_SCHEDULE

型 String

説明 スプール・ファイルの場合のみ、書き出しプログラムがスプール・ファイルをいつ使えるかを指定します。有効な値は *IMMED、*FILEEND、*JOBEND です。

開始したユーザー

ID ATTR_STARTEDBY

型 String

説明 書き出しプログラムを開始したユーザーの名前。

開始ページ

ID ATTR_STARTPAGE

型 Integer

説明 スプール・ファイルの印刷を開始するページ番号。有効な値は -1、0、1、またはページ番号です。値 *ENDPAGE は -1 とエンコードされます。値 0 の場合、印刷は 1 ページから開始されます。値 1 の場合、ファイル全体が印刷されます。

作成したシステム

ID ATTR_SYSTEM

型 String

説明 スプール・ファイルが作成されたシステムの名前。このスプール・ファイルが作成されたシステムの名前を特定できなければ、受信側システムの名前が使われます。

テキスト記述

ID ATTR_DESCRIPTION

型 String

説明 AS400 オブジェクトのインスタンスを記述するテキスト。

ファイルがオープンされた時刻

ID ATTR_TIMEOPEN

型 String

説明 スプール・ファイルの場合、これはそのスプール・ファイルがオープンされた時刻です。 AFP リソースの場合、これはそのオブジェクトが最後に変更された時刻です。時刻は、HH MM SS という形式の文字ストリングにエンコードされます。

スプール・ファイル作成ジョブが終了した時刻

ID ATTR_TIME_END

型 String

説明 システム上でスプール・ファイルを作成したジョブが終了した時刻。「スプール・ファイル作成開始日付」フィールドに特殊値 *ALL を使用する場合、または「スプール・ファイル作成終了日付」フィールドに特殊値 *LAST を使用する場合には、このフィールドをブランクに設定する必要があります。「スプール・ファイルの作成終了日付」フィールドに日付を指定する場合には、このフィールドには値が設定される必要があります。時刻は、以下に定義されているように HHMMSS 形式にしなければなりません。

- HH - 時刻
- MM - 分
- SS - 秒

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

ID ATTR_TIME_WTR_BEGAN_FILE

型 String

説明 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を示します。時刻は、HH MM SS という形式の文字ストリングにエンコードされます。

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

ID ATTR_TIME_WTR_CMPL_FILE

型 String

説明 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を示します。時刻は、HH MM SS という形式の文字ストリングにエンコードされます。

合計ページ数

ID ATTR_PAGES

型 Integer

説明 スプール・ファイルに入っているページ数。

SCS から ASCII への変換

ID ATTR_SCS2ASCII

型 String

説明 印刷データを SCS から ASCII に変換するかどうか。有効な値は *YES、*NO です。

測定単位

ID ATTR_UNITOFMEAS

型 String

説明 距離を指定するのに使う測定単位。有効な値は *CM、*INCH です。

ユーザー注記

ID ATTR_USERCMT

型 String

説明 スプール・ファイルを記述する 100 文字のユーザー指定の注記。

ユーザー・データ

ID ATTR_USERDATA

型 String

説明 スプール・ファイルを記述する 10 文字のユーザー指定のデータ。有効な特殊値には *SOURCE などがあります。

ユーザー定義データ

ID ATTR_USRDFNDTA

型 String

説明 スプール・ファイルを処理するユーザー・アプリケーションまたはユーザー指定プログラムで使用するユーザー定義データ。どの文字でも使用することができます。 最大サイズは 255 です。

ユーザー定義ファイル

ID ATTR_USRDEFFILE

型 String

説明 スプール・ファイルが API を使って作成されたかどうか。有効な値は *YES または *NO です。

ユーザー定義オブジェクト

ID ATTR_USER_DEFINED_OBJECT

型 String

説明 スプール・ファイルを処理するユーザー・アプリケーションで使用するユーザー定義オブジェクトの統合ファイル・システム・パス。統合ファイル・システム・パスの場合、その統合ファイル・システム・パスの形式は /QSYS.LIB/library.LIB/object.type です。ただし *library* は、オブジェクトが入っているかまたは特殊値 %LIBL% または %CURLIB% のどちらかが入っているライブラリーの名前です。 *object* はオブジェクトの名前、*type* はオブジェクトのタイプです。 *type* の有効な値には、"DTAARA"、"DTAQ"、"FILE"、"PSFCFG"、"USRIDX"、"USRQ" および "USRSPC" が含まれます。ユーザー定義オブジェクトを使用しないよう指示するには文字列 "*NONE" を使います。

ユーザー定義オプション

ID ATTR_USEDFNOPTS

型 String

説明 スプール・ファイルを処理するユーザー・アプリケーションで使用するユーザー定義オプション。それぞれの値が長さ char(10) である 4 個までのオプションを指定することができます。どの文字でも使用することができます。

ユーザー・ドライバー・プログラム・データ

ID ATTR_USRDRVPGMDTA

型 String

説明 ユーザー・ドライバー・プログラムで使用するユーザー・データ。どの文字でも使用することができます。 最大サイズは 5000 文字です。

ユーザー・ドライバー・プログラム

ID ATTR_USER_DRIVER_PROG

型 String

説明 スプール・ファイルを処理するユーザー定義ドライバー・プログラムの統合ファイル・システム・パス。統合ファイル・システム・パスの形式は /QSYS.LIB/library.LIB/program.PGM ですが、*library* はプログラムの入ったライブラリー、*program* はそのプログラムの名前です。 *library* は、特殊値 %LIBL% または %CURLIB% のどちらか、または特定のライブラリーの名前です。ドライバー・プログラムを定義しないよう指示するには文字列 "*NONE" を使います。

ユーザー ID

ID ATTR_TOUSERID

型 String

説明 スプール・ファイルの送信先のユーザー ID。

ユーザー ID アドレス

ID ATTR_TOADDRESS

型 String

説明 スプール・ファイルの送信先のユーザーのアドレス。

ユーザー変換プログラム

ID ATTR_USER_TRANSFORM_PROG

型 String

説明 ドライバー・プログラムでの処理の前にスプール・ファイルのデータを変換するユーザー定義変換プログラムの統合ファイル・システム・パス。統合ファイル・システム・パスの形式は /QSYS.LIB/library.LIB/program.PGM ですが、 *library* はプログラムの入ったライブラリー、*program* はそのプログラムの名前です。 *library* は、特殊値 %LIBL% または %CURLIB% のどちらか、または特定のライブラリーの名前です。変換プログラムを定義しないよう指示するには文字列 "*NONE" を使います。

精度の表示

ID ATTR_VIEWING_FIDELITY

型 String

説明 スプール・ファイル・データを表示する (一度に 1 ページ・モードで) ときに行われる処理。有効な値は *ABSOLUTE と *CONTENT(デフォルト) です。現在のページより前の非ラスター・データ (コマンド) を処理する場合はすべて、*ABSOLUTE を使います。 SCS ファイルの場合、オープン時刻コマンドに加えて現在のページを処理するには *CONTENT を使います。 AFPDS ファイルの場合、データの 1 ページ目に加えて現在のページを処理するには *CONTENT を使います。

VM/MVS クラス

ID ATTR_VMMVSCLASS

型 String

説明 VM/MVS クラス。有効な値は A から Z と 0 から 9 です。

データ待ち中状況

ID ATTR_WTNGDATASTS

型 String

説明 書き出しプログラムは、現在スプール・ファイル内にあるデータをすべて書き出してから、さらに別のデータ待ちになっているかどうか。指定できる値は、*NO (書き出しプログラムはさらに別のデータ待ちになっていない)、または *YES (書き出しプログラムは、現在スプール・ファイル内にあるデータをすべて書き出してから、さらに別のデータ待ちになっている) です。この条件が成立するのは、SCHEDULE(*IMMED) を指定して書き出しプログラムでオープン・スプール・ファイルを作成する場合です。

装置待ち中状況

ID ATTR_WTNGDEVSTS

型 String

説明 プリンターに直接印刷を行うジョブから装置を取得するために書き出しプログラムが待機中かどうか。値は *NO (書き出しプログラムは装置待ちになっていない)、または *YES (書き出しプログラムは装置待ちになっている) です。

メッセージ待ち中状況

ID ATTR_WTNGMSGSTS

型 String

説明 書き出しプログラムは照会メッセージに対する応答待ちになっているかどうか。値は *NO と *YES です。

書き出しプログラムの自動終了時期

ID ATTR_WTRAUTOEND

型 String

説明 書き出しプログラムを自動終了する場合に、このプログラムを終了させる時期。有効な値は *NORDYF、*FILEEND です。書き出しプログラムの自動終了属性は *YES に設定しなければなりません。

書き出しプログラムの終了時期

ID ATTR_WTREND

型 String

説明 書き出しプログラムをいつ終了するか。有効な値は *CNTRLD、*IMMED、および *PAGEEND です。これは、書き出しプログラムの自動終了とは異なります。

ファイルの保留時期

ID ATTR_HOLDTYPE

型 String

説明 スプール・ファイルをいつ保留にするか。有効な値は *IMMED、および *PAGEEND です。

ページ幅

ID ATTR_PAGEWIDTH

型 Float

説明 ページ幅。測定単位は測定メソッド属性に指定します。

ワークステーション・カスタマイズ・オブジェクト

ID ATTR_WORKSTATION_CUST_OBJECT

型 String

説明 ワークステーション・カスタマイズ・オブジェクトの統合ファイル・システム・パス。統合ファイル・システム・パスの形式は /QSYS.LIB/library.LIB/custobj.WSCST ですが、 *library* はカスタマイズ・オブジェクトの入ったライブラリー、 *custobj* はワークステーション・カスタマイズ・オブジェクトの名前です。

書き出しプログラムのジョブ名

ID ATTR_WRITER

型 String

説明 書き出しプログラム・ジョブの名前。

書き出しプログラムのジョブ番号

ID ATTR_WTRJOBNUM

型 String

説明 書き出しプログラム・ジョブ番号。

書き出しプログラムのジョブ状況

ID ATTR_WTRJOBSTS

型 String

説明 書き出しプログラム・ジョブの状況。有効な値は STR、END、JOBQ、HLD、MSGW です。

書き出しプログラムのジョブ・ユーザー名

ID ATTR_WTRJOBUSER

型 String

説明 書き出しプログラム・ジョブを開始したユーザーの名前。

開始済みの書き出しプログラム

ID ATTR_WTRSTRTD

型 String

説明 書き出しプログラムをこのプリンターで開始するかどうかを示します。値は 1 (はい。書き出しプログラムを開始します) または 0 (いいえ。書き出しプログラムを開始しません) です。

書き出しプログラム開始ページ

ID ATTR_WTRSTRPAGE

型 Integer

説明 書き出しプログラム・ジョブの開始時に最初のスプール・ファイルから印刷する最初のページのページ番号を指定します。これが有効なのは、書き出しプログラムの開始時にスプール・ファイル名も指定した場合だけです。

書き出し中状況

ID ATTR_WRTNGSTS

型 String

説明 印刷書き出しプログラムが書き出し中状況になっているかどうかを示します。値は *YES (書き出しプログラムは書き出し中状況になっている)、*NO (書き出しプログラムは書き出し中状況になっていない)、または *FILE (書き出しプログラムはファイル区切りを書き出している) です。

ネットワーク印刷サーバー・オブジェクトの属性

NPS CCSID

ID ATTR_NPSCCSID

型 Integer

説明 ネットワーク印刷サーバーにおいてすべての文字列でエンコードされることになっているCCSID。

NPS レベル

ID ATTR_NPSLEVEL

説明 ネットワーク印刷サーバーのバージョン、リリース、およびモディフィケーション・レベル。この属性は、VXRYMY (つまり "V3R1M0") とエンコードされる文字ストリングです。詳細は以下のとおりです。

```
X は 0 から 9 までのいずれか
Y は (0 から 9、A から Z) のいずれか
```

スプール・ファイルのコピー:

SpooledFile クラスのコピー・メソッドを使用すると、 SpooledFile オブジェクトが表すスプール・ファイルのコピーを作成できます。

SpooledFile.copy() を使用すると、以下の操作を行います。

- 新規のスプール・ファイルを、元のスプール・ファイルと同じ出力キュー、また同じシステム上に作成する
- リファレンスを新規のスプール・ファイルに戻す

SpooledFile.copy() は、JTOpen 3.2 以降をダウンロードするか、あるいは IBM i フィックスを適用した場合にのみ使用可能な新規のメソッドです。JTOpen をダウンロードして使用することを、お勧めします。詳細は、以下を参照してください。

- IBM Toolbox for Java and JTOpen: Downloads 
- IBM Toolbox for Java and JTOpen: Service Packs 

このコピー・メソッドは、ネットワーク印刷サーバー・ジョブ内でスプール・ファイルの作成 (QSPCRTSP) API を使用してスプール・ファイルの正確なレプリカを作成します。新しく作成されたスプール・ファイルのコピーを識別するために必要なのは、固有の作成日時だけです。

コピー・メソッドに出力キューをパラメーターとして指定すると、スプール・ファイルのコピーが指定の出力キューの最初の位置に作成されます。出力キューおよび元のスプール・ファイルの両方が同じシステムにある必要があります。

例: SpooledFile.copy() を使用して、スプール・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、 コードに関するライセンス情報および特記事項をお読みください。

この例では、 SpooledFile.copy() を使用して、コピーするファイルを含む同じ待ち行列にスプール・ファイルをコピーする方法を示します。新しくコピーされたスプール・ファイル・コピーを特定の出力キューに経路指定するには、出力キューをパラメーターとしてコピー・メソッドに渡してください。

```
SpooledFile newSplf = new sourceSpooledFile.copy(<outqname>);
```

<outqname> は、OutputQueue オブジェクトです。

```
public static void main(String args[]) {
   // Create the system object
   AS400 as400 = new AS400(<systemname>,<username>, <password>);
   // Identify the output queue that contains the spooled file you want to copy.
   OutputQueue outputQueue =
     new OutputQueue(as400, "/QSYS.LIB/QUSRSYS.LIB/<outqname>.OUTQ");

   // Create an array that contains all the elements required to
   // uniquely identify a spooled file on the server.
   String[][] splfTags = { {
      <spoolfilename>,
      <spoolfilenum>,
      <jobname>,
      <username>,
      <jobnumber>,
      // Note that <systemname>,<date>, and <time> are optional.
      // If you do not include them, remove the corresponding
      // splfTags[i],[j], where j has the value of 5,6, or 7.
      <systemname>,
      <date>,
      <time>},
   };

   // Print the information that identifies the spooled file to System.out
   for ( int i=0; i<splfTags.length; i++) {
      System.out.println("Copying -> " + splfTags[i][0] + ","
                                       + splfTags[i][1] + ","
                                       + splfTags[i][2] + ","
                                       + splfTags[i][3] + ","
                                       + splfTags[i][4] + ","
                                       + splfTags[i][5] + ","
                                       + splfTags[i][6] + ","
                                       + splfTags[i][7] );

   // Create the SpooledFile object for the source spooled file.
      SpooledFile sourceSpooledFile =
         new SpooledFile(as400,
                         splfTags[i][0],
                         Integer.parseInt(splfTags[i][1]),
                         splfTags[i][2],
                         splfTags[i][3],
                         splfTags[i][5],
                         splfTags[i][6],
                         splfTags[i][7] );
   }

   // Copy the spooled file, which creates a new SpooledFile object.
   // To route the copy of the spooled file to a specific output queue,
   // use the following code:
```

```
// SpooledFile newSplf = new sourceSpooledFile.copy(<outqname>);
// where <outqname> is an OutputQueue object. Specify the output
// queue in the following way:
// OutputQueue outputQueue =
//    new OutputQueue(as400, "/QSYS.LIB/QUSRSYS.LIB/<outqname>.OUTQ");
try { SpooledFile newSplf = new sourceSpooledFile.copy();
}

catch ( Exception e){
}
```

SpooledFile Javadoc

Create Spooled File (QSPCRTSP) API

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、Javadoc の 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型を判別するための自動データ型指定が可能になる (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・データ型)」属性を使用してください。呼び出し元がプリンター・データ型を指定しない場合、デフォルトは自動データ型指定の使用です。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。自動データ型指定は、ほとんどの場合うまくいきます。呼び出し元では、自動データ型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、この自動データ型指定を使用します。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型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型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・ファイルにあるデータ・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。

  SpooledFileOutputStream Javadoc

  574 ページの『例: スプール・ファイルを作成する』
  以下の例には、 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。

  575 ページの『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。

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・ストリームを作成しなければなりません。 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します。

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 writer 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 書き込み機能を拡張し、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。

例

575 ページの『例: SCS スプール・ファイルを作成する』 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Advanced Function Printing (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。

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、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

以下の例では、PrintObjectInputStream を使用して既存のサーバー・スプール・ファイルを読み取る方法を示します。

例: スプール・ファイルを読み取る

PrintObjectInputStream Javadoc

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。

例

以下の例では、スプール・ファイル・データを読み取る際に、異なる変換結果を得るための PrintObjectPageInputStream および PrintObjectTransformedInputStream の使用法を示します。

578 ページの『例: スプール・ファイルを読み取り変換する』

関連情報

PrintObjectPageInputStream Javadoc

PrintObjectTransformedInputStream Javadoc

ProductLicense クラス

ProductLicense クラスを使用すると、システム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 IBM i 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、IBM i 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・ポリシーを実施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実施するのに十分な情報を戻します。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べきかを認識しないからです。

IBM i 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実施します。

- 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 IBM i 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、IBM i 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。

関連情報

ProductLicense Javadoc

ProgramCall クラス

IBM Toolbox for Java ProgramCall クラスでは、Java プログラムが IBM i 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 IBM i プログラムから戻されたデータが入ります。 IBM i 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 IBM i 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サーバーを表す 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、 ProgramCall setProgram() メソッドを介してコンストラクターで設定するか、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 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。

例: ProgramCall を使用する

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

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。

デフォルトの動作では、 Java プログラムとIBM i プログラムが同じサーバー上にあるときでも、IBM i プログラムは別のサーバー・ジョブで稼働します。しかし、このデフォルトの動作をオーバーライドし、ProgramCall setThreadSafe() メソッドを使用して IBM i 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 IBM i 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 IBM i 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。

例: ProgramParameter を使用する

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

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 system 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 data type
   AS400Bin4 bin4Converter = new AS400Bin4();

   // Convert from system 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);
```

ProgramCall Javadoc

ProgramParameter Javadoc

AS400Message Javadoc

AS400 Javadoc

CommandCall Javadoc

ProgramParameter Javadoc

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。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

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: オブジェクトの名前が一度だけ使用される場合、Java プログラムは、toPath() メソッドを使用して名前を作成することができます。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();
```

QSYSObjectPathName Javadoc

レコード・レベルでのアクセス

レコード・レベルの access クラスは、IBM i の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。

- 次のいずれかを指定して IBM i 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
- 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 IBM i 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、IBM i ファイル内のレコードに順次にアクセスする。
- システム・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、システム・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、システム・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
- システム・ファイルを削除する。
- システム・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、キーでアクセスできるシステム・ファイルを表します。
- SequentialFile クラスは、レコード番号でアクセスできるシステム・ファイルを表します。

- AS400FileRecordDescription クラスは、システム・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 IBM i システムに接続されます。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、システム・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・クラスを使用してシステム・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、システム・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、次のようないくつかのアクション用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

関連情報

AS400File Javadoc

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のレコードから始まります。
- 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。現行カーソル位置の前のレコードから始まります。

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 before 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);
```

関連情報

KeyedFile Javadoc

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 before 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);
```

関連情報

SequentialFile Javadoc

AS400FileRecordDescription:

IBM Toolbox for Java 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

AS400FileRecordDescription 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上書きします。

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

AS400FileRecordDescription 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 as I am done using it
myFile.close();

// Disconnect since I am done using record-level access
sys.disconnectService(AS400.RECORDACCESS);
```

AS400FileRecordDescription Javadoc

RecordFormat Javadoc

AS400File Javadoc

ファイルとメンバーの作成および削除:

サーバー上の物理ファイルは、レコード長、既存のサーバー・データ記述仕様 (DDS) ソース・ファイル、または RecordFormat オブジェクトを指定することによって作成されます。

レコード長を指定してファイルを作成すると、データ・ファイルまたはソース・ファイルを作成することができます。オブジェクトのレコード様式はメソッドによって設定されます。オブジェクトのために setRecordFormat() メソッドを呼び出さないでください。

データ・ファイルには、1 つのフィールドがあります。フィールド名はファイルの名前で、フィールド・タイプは文字で、フィールド長は作成メソッドで指定された長さです。

ソース・ファイルには、3 つのフィールドがあります。

- フィールド SRCSEQ は ZONED DECIMAL (6,2) です。
- フィールド SRCDAT は ZONED DECIMAL (6,0) です。
- SRCDTA は、作成メソッドで指定された長さから 12 を引いた長さの文字フィールドです。

以下の例は、ファイルとメンバーを作成する方法を示しています。

例 1: 128 バイト・レコードが入るデータ・ファイルを作成する場合:

```
// Create an AS400 object, the file
// will be created on this server.
AS400 sys = new AS400("mySystem.myCompany.com");

// Create a file object that represents the file
SequentialFile newFile = new SequentialFile(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE/%FILE%.MBR");
// Create the file
newFile.create(128, "*DATA", "Data file with a 128 byte record");
```

```
// Open the file for writing only.
// Note: The record format for the file
// has already been set by create()
newFile.open(AS400File.WRITE_ONLY, 0, AS400File.COMMIT_LOCK_LEVEL_NONE);

// Write a record to the file.  Because the record
// format was set on the create(), getRecordFormat()
// can be called to get a record properly formatted
// for this file.
Record writeRec = newFile.getRecordFormat().getNewRecord();
writeRec.setField(0, "Record one");
newFile.write(writeRec);

// ....

// Close the file since I am done using it
newFile.close();

// Disconnect since I am done using record-level access
sys.disconnectService(AS400.RECORDACCESS);
```

例 2: 既存の DDS ソース・ファイルを指定してファイルを作成する場合、 DDS ソース・ファイルは create() メソッドで指定します。ファイルのレコード様式は、ファイルがオープンされる前に、 setRecordFormat() メソッドを使用して設定しなければなりません。たとえば、次のようにします。

```
// Create an AS400 object, the
// file will be created on this server.
AS400 sys = new AS400("mySystem.myCompany.com");

// Create QSYSObjectPathName objects for
// both the new file and the DDS file.
QSYSObjectPathName file    = new QSYSObjectPathName("MYLIB", "MYFILE", "FILE", "MBR");
QSYSObjectPathName ddsFile = new QSYSObjectPathName("MYLIB", "DDSFILE", "FILE", "MBR");

// Create a file object that represents the file
SequentialFile newFile = new SequentialFile(sys, file);

// Create the file
newFile.create(ddsFile,  "File created using DDSFile description");

// Set the record format for the file
// by retrieving it from the server.
newFile.setRecordFormat(new AS400FileRecordDescription(sys,
newFile.getPath()).retrieveRecordFormat()[0]);

// Open the file for writing
newFile.open(AS400File.WRITE_ONLY, 0, AS400File.COMMIT_LOCK_LEVEL_NONE);

// Write a record to the file.  The getRecordFormat()
// method followed by the getNewRecord() method is used to get
// a default record for the file.
Record writeRec = newFile.getRecordFormat().getNewRecord();
newFile.write(writeRec);

// ....

// Close the file since I am done using it
newFile.close();

// Disconnect since I am done using
// record-level access
sys.disconnectService(AS400.RECORDACCESS);
```

例 3: RecordFormat オブジェクトを指定してファイルを作成する場合、 RecordFormat オブジェクトは create() メソッドで指定します。オブジェクトのレコード様式はメソッドによって設定されます。オブジェクトのために setRecordFormat() メソッドを呼び出してはなりません。

```
     // Create an AS400 object, the file will be created
     // on this server.
     AS400 sys = new AS400("mySystem.myCompany.com");

     // Create a file object that represents the file
     SequentialFile newFile = new SequentialFile(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE/%FILE%.MBR");
// Retrieve the record format from an existing file
     RecordFormat recordFormat = new AS400FileRecordDescription(sys,
     "/QSYS.LIB/MYLIB.LIB/EXISTING.FILE/MBR1.MBR").retrieveRecordFormat()[0];

     // Create the file
     newFile.create(recordFormat,  "File created using record format object");

     // Open the file for writing only.
     // Note: The record format for the file
     // has already been set by create()
     newFile.open(AS400File.WRITE_ONLY, 0, AS400File.COMMIT_LOCK_LEVEL_NONE);

     // Write a record to the file.  The recordFormat
     // object is used to get a default record
     // properly formatted for the file.
     Record writeRec = recordFormat.getNewRecord();
     newFile.write(writeRec);

     // ....

     // Close the file since I am done using it
     newFile.close();
     // Disconnect since I am done using
     // record-level access
     sys.disconnectService(AS400.RECORDACCESS);
```

ファイルおよびメンバーを削除するときには、以下のメソッドを使用してください。

- サーバー・ファイルとそのすべてのメンバーを削除するには、 delete() を使用します。
- ファイルの 1 つのメンバーだけを削除するには、 deleteMember() メソッドを使用します。

ファイルにメンバーを追加するには、 addPhysicalFileMember() メソッドを使用します。

関連情報

AS400File Javadoc

レコードの読み取りおよび書き込み:

AS400File クラスを使用すると、サーバー上のファイル内のレコードを読み取り、書き込み、更新、および削除することができます。

レコードは Record クラスを介してアクセスされ、このクラスは RecordFormat クラスによって記述されます。レコード様式は、ファイルがオープンされる前に、 setRecordFormat() メソッドによって設定しなければなりません。ただし、ファイルが、オブジェクトのレコード様式を設定する create() メソッドの 1 つによって作成されたばかりである (close() が介在しない) 場合は除きます。

ファイルからレコードを読み取るには、read() メソッドを使用します。提供されているメソッドとその機能は、以下のとおりです。

- read() - 現行カーソル位置のレコードを読み取ります。
- readFirst() - ファイルの最初のレコードを読み取ります。

- readLast() - ファイルの最後のレコードを読み取ります。
- readNext() - ファイル内の次のレコードを読み取ります。
- readPrevious() - ファイル内の前のレコードを読み取ります。

以下の例は、readNext() メソッドを使用する方法を示しています。

```
// 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 before invoking open()
myFile.setRecordFormat(recordFormat);

// Open the file.
myFile.open(AS400File.READ_ONLY, 0, AS400File.COMMIT_LOCK_LEVEL_NONE);

// Read each record in the file writing field
// CUSTNAME to System.out
System.out.println("             CUSTOMER LIST");
System.out.println("_____________________________________________");

Record record = myFile.readNext();
while(record != null)
{
  System.out.println(record.getField("CUSTNAME"));
  record = myFile.readNext();
}

                ....

// Close the file since I am done using it
myFile.close();

// Disconnect since I am done using
// record-level access.
sys.disconnectService(AS400.RECORDACCESS);
```

カーソル位置にあるレコードを更新するには、 update() メソッドを使用します。

たとえば、次のようにします。

```
// 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 for updating
myFile.open(AS400File.READ_WRITE, 0, AS400File.COMMIT_LOCK_LEVEL_NONE);

// Update the first record in the file.  Assume
// that newName is a String with the new name for
// CUSTNAME
Record updateRec = myFile.readFirst();
updateRec.setField("CUSTNAME", newName);
myFile.update(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);
```

ファイルの終わりにレコードを追加するには、 write() メソッドを使用します。単一のレコードまたはレコードの配列をファイルに追加できます。

カーソル位置にあるレコードを削除するには、 deleteCurrentRecord() メソッドを使用します。

関連情報

AS400File Javadoc

ファイルのロック:

Java プログラムでは、最初の Java プログラムがファイルを使用しているときに他のユーザーがそのファイルにアクセスできないように、ファイルをロックすることができます。

ロック・タイプは、以下のとおりです。ロック・タイプに関する詳細は、AS400File Javadoc に示されています。

- 読み取り/排他ロック - 現行の Java プログラムはレコードを読み取り、他のプログラムはファイルにアクセスすることができません。
- 読み取り/共用読み取り許可ロック - 現行の Java プログラムはレコードを読み取り、他のプログラムはファイルからレコードを読み取ることができます。
- 読み取り/共用書き込み許可ロック - 現行の Java プログラムはレコードを読み取り、他のプログラムはファイルを変更することができます。
- 書き込み/排他ロック - 現行の Java プログラムはファイルを変更し、他のプログラムはファイルにアクセスすることができません。
- 書き込み/共用読み取り許可ロック - 現行の Java プログラムはファイルを変更し、他のプログラムはファイルからレコードを読み取ることができます。
- 書き込み/共用書き込み許可ロック - 現行の Java プログラムはファイルを変更し、他のプログラムはファイルを変更することができます。

lock() メソッドによって取得したロックを解除するには、Java プログラムは releaseExplicitLocks() メソッドを開始します。

関連情報

AS400File Javadoc

レコード・ブロック化を使用する:

IBM Toolbox forJava AS400File クラスは、レコード・ブロックを使用してパフォーマンスを改善します。

- ファイルが読み取り専用アクセスのためにオープンされる場合、 Java プログラムがレコードを読み取るときにレコードのブロックが読み取られます。ブロック化により、後続の読み取り要求がサーバーへのアクセスなしで処理される可能性があるため、パフォーマンスが向上します。 1 つのレコードの読み取りと複数のレコードの読み取りとでは、パフォーマンスの違いはほとんどありません。クライアントにキャッシュされているレコードのブロックからレコードが取り出される場合は、パフォーマンスが大幅に向上します。

  ファイルのオープン時に、各ブロックで読み取るレコードの数を設定することができます。たとえば、次のようにします。

```
// 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 before invoking open()
myFile.setRecordFormat(recordFormat);

// Open the file.  Specify a blocking factor of 50.
int blockingFactor = 50;
myFile.open(AS400File.READ_ONLY, blockingFactor, AS400File.COMMIT_LOCK_LEVEL_NONE);

// Read the first record of the file.  Because
// a blocking factor was specified, 50 records
// are retrieved during this read() invocation.
Record record = myFile.readFirst();
for (int i = 1; i < 50 && record != null; i++)
{

  // The records read in this loop will be served out of the block of
  // records cached on the client.
  record = myFile.readNext();
}
                ....

// Close the file since I am done using it
myFile.close();

// Disconnect since I am done using
// record-level access
sys.disconnectService(AS400.RECORDACCESS);
```

- ファイルが書き込み専用アクセスのためにオープンされる場合、ブロック化因数は、 write(Record[]) メソッドの呼び出し時にファイルに一度に書き込まれるレコードの数を示します。

たとえば、次のようにします。

```
// 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.  Specify a blocking factor of 50.
int blockingFactor = 50;
myFile.open(AS400File.WRITE_ONLY, blockingFactor, AS400File.COMMIT_LOCK_LEVEL_NONE);

// Create an array of records to write to the file
Record[] records = new Record[100];
for (int i = 0; i < 100; i++)
{
  // Assume the file has two fields,
  // CUSTNAME and CUSTNUM
  records[i] = recordFormat.getNewRecord();
  records[i].setField("CUSTNAME", "Customer " + String.valueOf(i));
  records[i].setField("CUSTNUM", new Integer(i));
}
// Write the records to the file.  Because the
// blocking factor is 50, only two trips to the
// server are made with each trip writing 50 records
myFile.write(records);

                ....

// Close the file since I am done using it
myFile.close();

// Disconnect since I am done using
// record-level access
sys.disconnectService(AS400.RECORDACCESS);
```

- ファイルが読み取り/書き込みアクセスのためにオープンされる場合、ブロック化は行われません。 open() に指定されたブロック化因数はすべて無視されます。

  AS400File Javadoc

カーソル位置の設定:

オープン・ファイルには、カーソルがあります。カーソルは、読み取り、更新、または削除されるレコードを指します。ファイルを初めてオープンするときには、カーソルはファイルの先頭を指します。ファイルの先頭とは、最初のレコードの前になります。

カーソル位置を設定するには、以下のメソッドを使用してください。

- positionCursorAfterLast() - カーソルを最後のレコードの後ろに設定します。このメソッドは、 Java プログラムが readPrevious() メソッドを使用してファイル内のレコードにアクセスできるようにするために存在します。

- positionCursorBeforeFirst() - カーソルを最初のレコードの前に設定します。このメソッドは、 Java プログラムが readNext() メソッドを使用してファイル内のレコードにアクセスできるようにするために存在します。
- positionCursorToFirst() - カーソルを最初のレコードに設定します。
- positionCursorToLast() - カーソルを最後のレコードに設定します。
- positionCursorToNext() - カーソルを次のレコードに移動します。
- positionCursorToPrevious() - カーソルを前のレコードに移動します。

以下の例は、positionCursorToFirst() メソッドを使用してカーソルを位置付ける方法を示しています。

```
// 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 before invoking open()
myFile.setRecordFormat(recordFormat);

// Open the file.
myFile.open(AS400File.READ_WRITE, 1, AS400File.COMMIT_LOCK_LEVEL_NONE);

// I want to delete the first record of the file.
myFile.positionCursorToFirst();
myFile.deleteCurrentRecord();

                ....

// Close the file since I am done using it
myFile.close();

// Disconnect since I am done using
// record-level access
sys.disconnectService(AS400.RECORDACCESS);
```

関連情報

AS400File Javadoc

コミットメント制御:

コミットメント制御を使用すると、 Java プログラムではファイルの変更を別のレベルで制御することができます。コミットメント制御をオンにすると、ファイルへのトランザクションは、コミットまたはロールバックされるまで保留になります。コミットされると、すべての変更がファイルに入れられます。ロールバックされると、すべての変更が廃棄されます。トランザクションは、open() で指定されたコミットメント制御ロック・レベルに応じて、既存レコードの変更、レコードの追加、レコードの削除、またはレコードの読み取りさえ可能です。

コミットメント制御のレベルは、以下のとおりです。各レベルの詳細は、AS400File Javadoc を参照してください。

- 全 (All) - ファイル内のアクセスされたすべてのレコードは、トランザクションがコミットまたはロールバックされるまでロックされます。
- 変更 (Change) - ファイル内の更新、追加、および削除されたレコードは、トランザクションがコミットまたはロールバックされるまでロックされます。
- カーソル固定 (Cursor Stability) - ファイル内の更新、追加、および削除されたレコードは、トランザクションがコミットまたはロールバックされるまでロックされます。アクセスされたが変更されないレコードは、別のレコードがアクセスされるまでの間だけロックされます。
- なし (None) - ファイルに対するコミットメント制御はありません。変更はただちにファイルに適用され、ロールバックできません。

AS400File startCommitmentControl() メソッドを使用すると、コミットメント制御を開始することができます。コミットメント制御は、AS400 **接続**に適用されます。接続に対してコミットメント制御が開始されると、コミットメント制御が開始された時間以降にその接続のもとでオープンされたすべてのファイルにコミットメント制御が適用されます。コミットメント制御の開始前にオープンされたファイルは、コミットメント制御を受けません。個々のファイルに対するコミットメント制御のレベルは、 open() メソッドで指定されます。 startCommitmentControl() メソッドで指定されたのと同じレベルのコミットメント制御を使用するには、 COMMIT_LOCK_LEVEL_DEFAULT を指定してください。

たとえば、次のようにします。

```
// Create an AS400 object, the files exist on this
// server.
AS400 sys = new AS400("mySystem.myCompany.com");

// Create three file objects
SequentialFile myFile = new SequentialFile(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE/%FILE%.MBR");
SequentialFile yourFile = new SequentialFile(sys, "/QSYS.LIB/YOURLIB.LIB/YOURFILE.FILE/%FILE%.MBR");
SequentialFile ourFile = new SequentialFile(sys, "/QSYS.LIB/OURLIB.LIB/OURFILE.FILE/%FILE%.MBR");

// Open yourFile before starting commitment control
// No commitment control applies to this file.  The
// commit lock level parameter is ignored because
// commitment control is not started for the connection.
yourFile.setRecordFormat(new YOURFILEFormat());
yourFile.open(AS400File.READ_WRITE, 0, AS400File.COMMIT_LOCK_LEVEL_DEFAULT);

// Start commitment control for the connection.
// Note: Any of the three files might be used for
// this call to startCommitmentControl().
myFile.startCommitmentControl(AS400File.COMMIT_LOCK_LEVEL_CHANGE);

// Open myFile and ourFile
myFile.setRecordFormat(new MYFILEFormat());

// Use the same commit lock level as specified
// when commitment control was started
myFile.open(AS400File.WRITE_ONLY, 0, COMMIT_LOCK_LEVEL_DEFAULT);

ourFile.setRecordFormat(new OURFILEFormat());
// Specify a different commit lock level than
// when commitment control was started
ourFile.open(AS400File.READ_WRITE, 0, COMMIT_LOCK_LEVEL_CURSOR_STABILITY);

// write and update records in all three files
                ....

// Commit the changes for files myFile and ourFile.
// Note that the commit commits all changes for the connection
// even though it is invoked on only one AS400File object.
myFile.commit();
```

```
// Close the files
myFile.close();
yourFile.close();
ourFile.close();

// End commitment control
// This ends commitment control for the connection.
ourFile.endCommitmentControl();

// Disconnect since I am done using record-level access
sys.disconnectService(AS400.RECORDACCESS);
```

commit() メソッドは、**接続**に対する最後のコミット境界以後のすべてのトランザクションをコミットします。 rollback() メソッドは、**接続**に対する最後のコミット境界以後のすべてのトランザクションを廃棄します。接続に対するコミットメント制御は、 endCommitmentControl() メソッドによって終了されます。 commit() または rollback() メソッドの呼び出し前にファイルがクローズされると、コミットされていないトランザクションはすべてロールバックされます。コミットメント制御下でオープンされたすべてのファイルは、 endCommitmentControl() メソッドの呼び出し前にクローズしておかなければなりません。

以下の例は、コミットメント制御を開始し、機能をコミットまたはロールバックした後、コミットメント制御を終了する方法を示しています。

```
// ... assume the AS400 object and file have been
// instantiated.

// Start commitment control for *CHANGE
aFile.startCommitmentControl(AS400File.COMMIT_LOCK_LEVEL_CHANGE);

// ... open the file and do several changes.  For
// example, update, add or delete records.

// Based on a flag either save or discard the
// transactions.
if (saveChanges)
   aFile.commit();
else
   aFile.rollback();

// Close the file
aFile.close();

// End commitment control for the connection.
aFile.endCommitmentControl();
```

関連情報

AS400File Javadoc

SaveFile クラス

SaveFile クラスは、サーバー上の保管ファイルを表します。

関連情報

SaveFile Javadoc

ServiceProgramCall クラス

IBM Toolbox for Java ServiceProgramCall クラスでは、IBM i サービス・プログラムを呼び出すことができます。 ServiceProgramCall は、 IBM i 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。 IBM i 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、 IBM i 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サービス・プログラムにデータを渡し、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、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。

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。 しかし、このデフォルトの動作を指定変更し、ProgramCall を継承元とする setThreadSafe() メソッドを使用して、サービス・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。

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・プログラムはサーバー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、IBM i 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。データがサーバーに渡されると、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、IBM i 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。

ServiceProgramCall Javadoc

ProgramCall Javadoc

ProgramParameter Javadoc

Subsystem クラス

Subsystem クラスは、サーバー上のサブシステムを表します。

関連情報

Subsystem Javadoc

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

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

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();
```

SystemStatus Javadoc

SystemPool クラス:

SystemPool クラスを使うと、システム・プール情報を取り出して変更することができます。

SystemPool クラスには、以下のメソッドが含まれます。

- getPoolSize() メソッドは、プールのサイズを戻し、 setPoolSize() メソッドは、プールのサイズを設定します。
- getPoolName() メソッドは、プールの名前を取り出し、 setPoolName() メソッドは、プールの名前を設定します。

- getReservedSize() メソッドは、システム使用のために予約されている、プール内の記憶域の量を戻します。
- getDescription() メソッドは、システム・プールの説明を戻します。
- getMaximumActiveThreads() メソッドは、プール内で同時に活動状態になることができるスレッドの最大数を戻します。
- setMaximumFaults() メソッドは、このシステム・プールについて使用する、1 秒当たりの最大障害数の指針を設定します。
- setPriority() メソッドは、このシステム・プールの、他のシステム・プールとの相対的な優先順位を設定します。

例

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
//Create AS400 object.
AS400 as400 = new AS400("system name");

//Construct a system pool object.
SystemPool systemPool = new SystemPool(as400,"*SPOOL");

//Get system pool paging option
System.out.println("Paging option : "+systemPool.getPagingOption());
```

関連情報

SystemPool Javadoc

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
- 説明
- 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、サーバーから値が検索され、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、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、指定されたサーバー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 プログラムから一度に 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

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

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 System i servers.
//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();
```

SystemValue Javadoc

SystemValueList Javadoc

SystemValueGroup Javadoc

Trace クラス

Trace クラスを使用すると、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	システムと 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。

情報カテゴリー	説明
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 プログラムはロギングをオンにするための手段を提供します。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。

例

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

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: setTraceOn() を使用し、 log メソッドを使ってデータをログに書き込む

```
// 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: Trace を使用する

以下のコードで、トレースを使用する場合、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: コンポーネント・トレースを使用する

```
// 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 IBM Toolbox for Java and the component trace data each to separate files.
// The 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 IBM Toolbox for Java
// trace data.

Trace.setFileName("c:¥¥bit.bucket");
Trace.setFileName(myComponent1, "c:¥¥Component1.log");
```

関連情報

Trace Javadoc

ユーザー・クラスとグループ・クラス

IBM Toolbox for Java の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、サーバー上のユーザーとユーザー・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。

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() メソッドを使ってシステム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() の UserList メソッド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。

User Javadoc

UserGroup Javadoc

UserList Javadoc

AS400 Javadoc

597 ページの『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』
このソースは、IBM Toolbox for Java UserList の例です。 このプログラムは、特定のグループ内のすべてのユーザーをリストします。

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設定できます。

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。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
// 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 10KB
          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);
```

UserSpace Javadoc

AS400 Javadoc

Commtrace クラス

IBM Toolbox for Java の Commtrace クラスを使用すると、 Java プログラムが、指定された LAN (イーサネットまたはトークンリング) 回線記述に関する通信トレース・データを処理できます。 Commtrace パッケージには、スタンドアロン型ユーティリティー・プログラムとして実行し、通信トレース・データをフォーマットできるクラスが含まれています。

サーバーの通信トレースをストリーム・ファイルにダンプすると、情報はバイナリー形式で保管されます。Commtrace クラスを使用すると、ストリーム・ファイルのさまざまなコンポーネントを処理できます。

注: 通信トレース・ファイルには、暗号化されていないパスワードなどの機密情報が含まれる場合もあります。通信トレース・ファイルがサーバー上にある場合は、*SERVICE 特殊権限のあるユーザーのみがトレース・データにアクセスできます。このファイルをクライアントに移動する場合には、該当する方法でファイルを保護していることを確認してください。通信トレースについて詳しくは、このページの下部のリンクを参照してください。

Commtrace クラスを使用して、以下のタスクを実行してください。

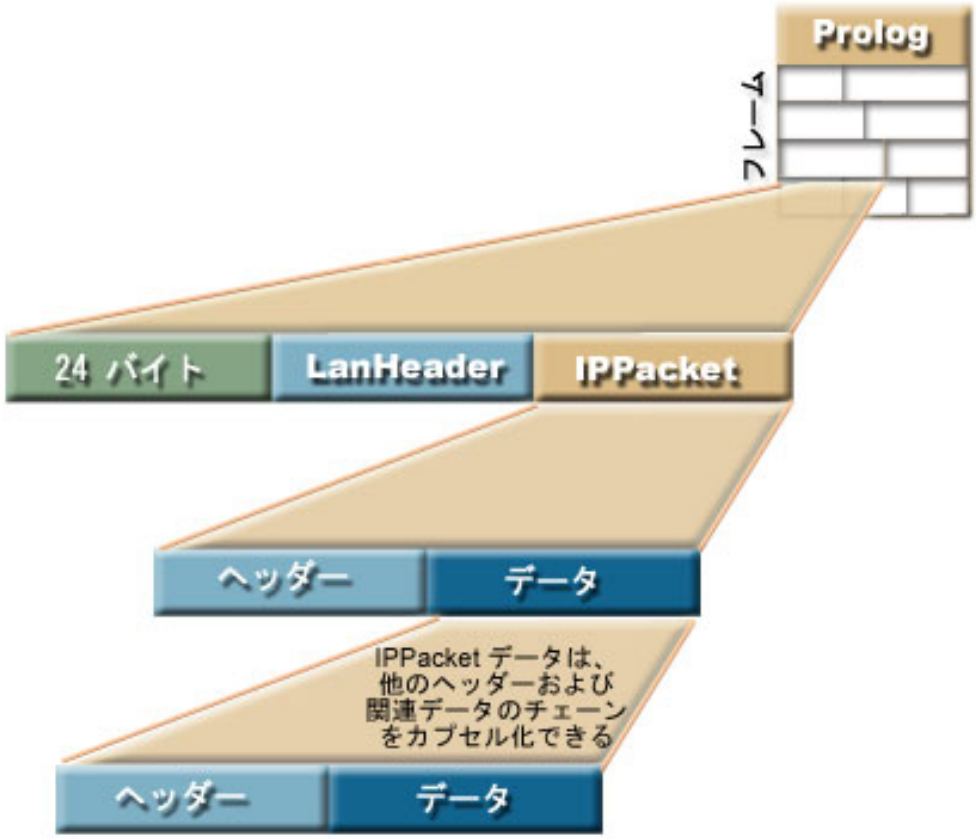
- 生のトレース・データをフォーマットする。
- データを解析して必要な情報を抽出する。 Commtrace クラスを使用してデータをフォーマットすると、生のデータとフォーマット済みデータの両方とも解析できます。

ここにリストされていない com.ibm.as400.commtrace パッケージ中の他のクラスは、処理するトレース・データのタイプごとに固有のクラスです。通信トレースおよびすべての Commtrace クラスについて詳しくは、通信トレースを参照してください。

Commtrace モデル

以下の図は、 Commtrace クラスと通信トレース・ファイルの対応を図解しています。また、 Commtrace クラスが使用する通信トレース中のコンポーネントの命名規則も示しています。

図 1: Commtrace モデル



トレース・ファイル中の個々のフレームには、2 つの初期セクション (フレームの内容に関する一般情報を提供する) と、サーバーがネットワーク上の別のポイントとの間で伝送したパケットが入っています。

データの一番初めの 24 バイト・セクションには、フレーム番号やデータ長などの、フレームの内容に関する一般情報が含まれています。この情報を処理するには、Frame クラスを使用してください。

Format および FormatProperties クラス

IBM Toolbox for Java Format クラスは、通信トレースから生のデータとフォーマット済みデータの両方を読み取ります。 FormatProperties は、開始時刻と終了時刻、IP アドレス、およびポートなどの Format オブジェクトのプロパティーを設定します。

Format クラスは、呼び出し側プログラムとトレースのフレームとの間のインターフェースの働きをします。 FormatProperties クラスを使用すると、 Format オブジェクトがトレースのフレーム中で情報を検出した場合に作動する方法を決定するプロパティーの設定および検索を行えます。

Format クラス

生のトレース・データと、 Commtrace クラスを使用してすでにフォーマットしたトレース・データの両方を読み取るには、 Format クラスを使用してください。

注: Commtrace クラスを使用して、 Print Communications Trace (PRTCMNTRC) 制御言語コマンドを使用してフォーマットした通信トレースを読み取ることはできません。

Format クラスを使用して、トレース中の情報を解析してフォーマットしてから、フォーマット済みの情報をファイルまたは印刷装置に送信してください。さらに、スタンドアロン型アプリケーションまたはブラウザー中に情報を表示するグラフィカル・フロントエンドを作成することもできます。特定のデータのみ選択する場合は、 Format クラスを使用して、Java プログラムにその情報を指定してください。例えば、Format クラスを使用して、トレースから IP アドレスを読み取ってから、このデータをプログラム中で使用できます。

Format コンストラクターは、IFSFileInputStream オブジェクト、ローカル・ファイル、バイナリー・トレース・ファイルなどのフォーマットされていないデータを表す引数を受け入れます。すでにフォーマット済みのトレースを表示するには、デフォルトの Format コンストラクターを使用してから、Format.openIFSFile() または Format.openLclFile() を使用して、表示するフォーマット済みファイルを指定してください。

例

以下の例は、保管済みのトレースの表示またはバイナリー・トレースのフォーマットを行う方法を示します。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

例: 保管済みのトレースを表示する

```
Format fmt = new Format();
fmt.openLclFile("/path/to/file");

// Read the Prolog
System.out.println(fmt.getRecFromFile());
// The total number of records in the trace TCP and non-TCP
System.out.println("Total Records:" + fmt.getIntFromFile());
String rec;
// Read in records until we reach the end.
while((rec = fmt.getRecFromFile())!=null) {
System.out.println(rec);
```

例: バイナリー・トレースをフォーマットする

```
// Create a FormatProperties. By default display everything.
FormatProperties fmtprop = new FormatProperties();


Format fmt = new Format("/path/to/file");
// Sets the filtering properties for this format
fmt.setFilterProperties(fmtprop);
fmt.setOutFile("/path/to/output/file");
// Format the prolog
fmt.formatProlog();
// Format the trace and send data to the specified file
fmt.toLclBinFile();
```

スタンドアロン型ユーティリティーとしての Format の実行

Format クラスをスタンドアロン型ユーティリティー・プログラムとして実行することもできます。詳細は、以下のトピックを参照してください。

スタンドアロン型プログラムとしての Format の実行

FormatProperties クラス

Format オブジェクトのプロパティーの指定および検索を行うには、 FormatProperties クラスを使用してください。つまり、Format クラスを使用して情報をファイルに送信する際には、 FormatProperties クラスを使用して、送信する情報をフィルター操作してください。

これらのプロパティーは、 Format オブジェクトが通信トレースのフレーム中で検出した情報を処理する方法を指定します。デフォルトの動作は、Format オブジェクトが、特定の値が指定されていないプロパティーを無視することです。

FormatProperties クラスは、プロパティーの設定時に使用する定数を指定します。プロパティーを設定すると、Format オブジェクトが、どのフィルターを使用するか検査できます。例えば、以下のコードは、Format オブジェクトが進行状況ダイアログを表示してブロードキャスト・フレームを表示しないよう設定します。

```
FormatProperties prop = new FormatProperties();
prop.setProgress(FormatProperties.TRUE);
prop.setBroadcast(FormatProperties.NO);
```

ほとんどのプロパティーは、特定のデータを明示的に組み込むよう設定するフィルターとして、 Format オブジェクトで使用できます。フィルターを設定すると、 Format オブジェクトはそれらのフィルターと合致するデータのみ表示します。例えば、以下のコードは、特定の開始時刻と終了時刻の間にあるフレームを表示するフィルターを設定します。

```
FormatProperties prop = new FormatProperties();
  // Set the filter to start and end times of 22 July, 2002,
  // 2:30 p.m. and 2:45 p.m. GMT.
  // The time is expressed as a Unix(TM) timestamp, which is
  // based on the standard epoch of 01/01/1970 at 00:00:00 GMT.
  prop.setStartTime("1027348200");
  prop.setEndTime("1027349100");
```

例

以下の例には、 Format および FormatProperties クラスを含む多数の Commtrace クラスを使用して、モニターにトレース情報を表示する方法が示されています。

203 ページの『例: Commtrace クラスを使用する』

Format Javadoc

FormatProperties Javadoc

スタンドアロン型プログラムとしての Format の実行:

Java プログラム中で Format クラスを使用することに加えて、スタンドアロン型のコマンド行ユーティリティーとして実行し、通信トレースをフォーマットできます。プログラムは指定された出力ファイルに IFSFileOutputStream を接続し、このファイルにデータを書き込みます。

スタンドアロン型ユーティリティーとして Format を実行すると、サーバーの処理能力とストレージ・スペースを使用して、ファイルをフォーマットできます。

コマンド行からの Format の実行

コマンド行プロンプトから Format ユーティリティーを実行するには、以下のコマンドを使用してください。

```
java com.ibm.as400.commtrace.Format [options]
```

[options] は、1 つ以上の使用可能なオプションです。オプションには、次のものが含まれます。

- 接続先のシステム
- システムのユーザー ID およびパスワード
- 解析する通信トレース
- 結果を保管するファイル

利用できるオプションの完全リストは、Format クラスの Javadoc 参照資料を参照してください。

リモート側での Format の実行

このクラスをリモート側で実行するには、 JavaApplicationCall クラスを使用してください。

```
// Construct a JavaApplicationCall object.
jaCall = new JavaApplicationCall(sys);

// Set the Java application you want to run.
jaCall.setJavaApplication("com.ibm.as400.util.commtrace.Format");

// Set the classpath environment variable used by the JVM on
// the server, so it can find the class to run.
jaCall.setClassPath("/QIBM/ProdData/OS400/JT400/lib/JT400Native.jar");

String[] args2 =
{ "-c", "true", "-t", "/path/to/trace", "-o", "/path/to/trace.extension"};

jaCall.setParameters(args2);

if (jaCall.run() != true) {
  // Call Failed
}
```

関連情報

Format Javadoc

Prolog クラス

IBM Toolbox for Java Prolog クラスは、 LAN 回線記述の通信トレースの初期 256 バイト・セクションを表します。 Prolog には、開始時刻および終了時刻や収集されるバイト数などの、トレースに関する一般情報が含まれています。トレースのこのセクションから情報を取り出すには、 Prolog クラスを使用してください。検索後、印刷、表示、フィルター操作、または他の方法の処理を行えます。

Prolog クラスは、以下を含むさまざまな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。

- Prolog のフィールドから、トレース記述、イーサネット・タイプ、データの方向、 IP アドレスなどの値を取り出す。
- Prolog のすべてのフィールドを含むフォーマット済みストリングを戻す。
- 無効データに関して Prolog フィールドをテストする。

例

以下の例には、Prolog クラスを含む多数の Commtrace クラスを使用して、モニターにトレース情報を表示する方法が示されています。

203 ページの『例: Commtrace クラスを使用する』

関連情報

Prolog Javadoc

Frame クラス

Frame クラスは、LAN 回線記述に関する通信トレース中の 1 つのレコード、またはフレーム中のすべてのデータを表します。

個々のフレームにはデータの 3 つのメイン・セクションが含まれ、以下の順序で表示されます。

1. フレームに関する一般情報を含む初期 24 バイト・セクション
2. フレームに関する一般情報 (LanHeader クラスによって表される)
3. パケット・データ (IPacket 抽象クラスのサブクラスによって表される)

Frame クラスを使用して、フレーム中のデータを解析して印刷可能な表示を作成してください。 Frame クラスは、特定の形式を使用して、リンク・リストに似た構造で、パケット・データを保守します。フレーム中のパケット・データの有効な形式に関する特定の情報と、フレームの構造に関する一般情報については、196 ページの『Commtrace モデル』を参照してください。

Frame クラスは、以下を含むさまざまな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。

- データ・パケットを検索する。
- フレームの番号、状況、およびタイプを検索する。
- 特定のデータをフォーマット済みストリングとしてフレームから戻す。

以下のプロセスを使用して、パケット中のデータにアクセスできます。

1. Frame.getPacket() を使用してパケットを検索します。
2. Packet.getHeader() を呼び出してヘッダー中のデータにアクセスします。
3. ヘッダーの検索後に、Header.getType() を呼び出してタイプを検索します。
4. 特定の Header サブクラスを使用して、そのヘッダーに関連したデータ (ペイロード) および追加のヘッダーにアクセスします。

例

以下の例には、 Format および FormatProperties クラスを含む多数の Commtrace クラスを使用して、モニターにトレース情報を表示する方法が示されています。

203 ページの『例: Commtrace クラスを使用する』

LanHeader クラス

LanHeader クラスは、フレームの先頭近くの最初の 24 バイト・セクション内に 1 つのみあるデータのセクションから、情報を取り出します。通常このセクションには、ハードウェア固有の情報が含まれます。この情報には、フレーム番号やデータ長などの、フレームに関する一般情報が含まれます。

LanHeader 中の情報の解析や印刷を行うには、LanHeader クラスを使用してください。 LanHeader に含まれる情報には、以下の種類があります。

- このパケット中の最初のヘッダーの先頭を識別するバイト
- メディア・アクセス制御 (MAC) アドレス
- トークンリング・アドレスおよび経路指定情報

LanHeader は、以下を含むフォーマット済みストリングを戻す 2 つのメソッドも備えています。

- トークンリング経路指定データ
- ソース MAC アドレス、宛先 MAC アドレス、フレーム形式、およびフレーム・タイプ

関連情報

LanHeader Javadoc

IPPacket クラス

IBM Toolbox for Java IPPacket クラスは、通信トレース時にネットワークで伝送された、このフレームのすべてのデータ・パケットを表します。 IPPacket は抽象クラスなので、さまざまな具象サブクラスを使用して、パケット中のヘッダーおよびデータを処理できます。

IPPacket のサブクラスには、次のものが含まれます。

- ARPPacket
- IP4Packet
- IP6Packet
- UnknownPacket

Packet クラスを使用すると、パケットのタイプを取り出し、パケットに含まれる生のデータ (ヘッダーおよびペイロード) にアクセスできます。すべてのサブクラスで同じコンストラクターが使用され、印刷用のパケット内容をストリングとして戻す追加のメソッドが 1 つ組み込まれています。

すべての Packet クラスのコンストラクターで引数としてパケット・データのバイト配列が使用されますが、 ARPPacket にはフレームのタイプを指定する整数も必要です。 Packet クラスのインスタンスを作成すると、該当する Header オブジェクトが自動的に作成されます。

Packet クラスは、以下を含むさまざまな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。

- パケットの名前およびタイプを取り出す。
- パケットのタイプを設定する。
- パケットに関連した最上位の Header オブジェクトを戻す。
- すべてのパケット・データをフォーマットされていないストリングとして戻す。
- 特定のデータをフォーマット済みストリングとしてパケットから戻す。

IPPacket Javadoc

ARPPacket Javadoc

IP4Packet Javadoc

IP6Packet Javadoc

UnknownPacket Javadoc

Header クラス

Header クラスは、特定の種類のパケット・ヘッダーを表すクラスを作成する抽象スーパークラスです。パケット・ヘッダーには関連データ (つまりペイロード) が組み込まれていますが、このデータは他のヘッダーやペイロードである場合もあります。

Header のサブクラスには、次のものが含まれます。

- ARPHeader
- ExtHeader
- ICMP4Header

- ICMP6Header
- IP4Header
- IP6Header
- TCPHeader
- UDPHeader
- UnknownHeader

Header クラスを使用すると、ヘッダーおよびペイロードに関するデータを検索できます。あるヘッダーが他のヘッダーとそれらのペイロードをカプセル化できます。

Packet クラスのインスタンスを作成すると、該当する Header オブジェクトが自動的に作成されます。Header クラスは、以下を含むさまざまな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。

- ヘッダーの長さ、名前、およびタイプを戻す。
- ヘッダー中のデータをバイト配列として取り出す。
- パケット中の次のヘッダーを取り出す。
- ペイロードをバイト配列、ASCII ストリング、および 16 進数ストリングとして取り出す。
- すべてのヘッダー・データをフォーマットされていないストリングとして戻す。
- 特定のデータをフォーマット済みストリングとしてヘッダーから戻す。

Header Javadoc

ARPHeader Javadoc

ExtHeader Javadoc

ICMP4Header Javadoc

ICMP6Header Javadoc

IP4Header Javadoc

IP6Header Javadoc

TCPHeader Javadoc

UDPHeader Javadoc

UnknownHeader Javadoc

例: Commtrace クラスを使用する

この例は、IBM Toolbox for Java commtrace クラスを使用して、通信トレース・データをモニターに出力します。そのために、通信トレース・バイナリー・ファイルをデータのソースとして使用します。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
///////////////////////////////////////////////////////////////////////
//
// Example using the commtrace classes to print communications trace
// data to a monitor by using a communications trace binary file as
// the source for the data.
//
// Command syntax:
//    java CommTraceExample
//
///////////////////////////////////////////////////////////////////////

import com.ibm.as400.util.commtrace.*;
```

```
public class CommTraceExample {


  public CommTraceExample() {
    // Create a FormatProperties. By default display everything.
    FormatProperties fmtprop = new FormatProperties();

    Format fmt = new Format("/path/to/file");
    // Sets the filtering properties for this format
    fmt.setFilterProperties(fmtprop);
    fmt.formatProlog(); // Format the prolog

    Prolog pro = fmt.getProlog();
    System.out.println(pro.toString());

    // If this is not a valid trace
    if (!pro.invalidData()) {
      Frame rec;

      // Get the records
      while ((rec = fmt.getNextRecord()) != null) {

        // Print out the Frame Number
        System.out.print("Record:" + rec.getRecNum());
        // Print out the time
        System.out.println(" Time:" + rec.getTime());
        // Get this records packet
        IPPacket p = rec.getPacket();
        // Get the first header
        Header h = p.getHeader();

        // If IP6 IPPacket
        if (p.getType() == IPPacket.IP6) {

          // If IP6 Header
          if (h.getType() == Header.IP6) {

            // Cast to IP6 so we can access methods
            IP6Header ip6 = (IP6Header) h;

            System.out.println(h.getName() + " src:" + ip6.getSrcAddr() + " dst:" + ip6.getDstAddr());
            // Print the header as hex
            System.out.println(ip6.printHexHeader());
            // Print a string representation of the header.
            System.out.println("Complete " + h.getName() + ":¥n" + ip6.toString(fmtprop));

            // Get the rest of the headers
            while ((h = h.getNextHeader()) != null) {

              // If it is a TCP header
              if (h.getType() == Header.TCP) {
                // Cast so we can access methods
                TCPHeader tcp = (TCPHeader) h;
                System.out.println(h.getName() +" src:" + tcp.getSrcPort() +" dst:" + tcp.getDstPort());
                System.out.println("Complete " + h.getName() + ":¥n" + tcp.toString(fmtprop));

                // If it is a UDP header
              } else if (h.getType() == Header.UDP) {
                // Cast so we can access methods
                UDPHeader udp = (UDPHeader) h;
                System.out.println(h.getName() +" src:" + udp.getSrcPort() + " dst:" + udp.getDstPort());
                System.out.println("Complete " + h.getName() + ":¥n" + udp.toString(fmtprop));
              }
            }
          }
        }
      }
```

```
    }
  }

    public static void main(String[] args) {
        CommTraceExample e = new CommTraceExample();
    }
}
```

HTML クラス

IBM Toolbox for Java HTML クラスは、多数の共通する HTML タグ・エレメントの表示の手段になります。

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
- Extensible Stylesheet Language (XSL) formatting object (FO) ソース・データの作成
- 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、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。

さらに、HTMLDocument クラスを使用して、簡単に HTML ページまたは XSL FO ソース・データを作成できます。 XSL FO データを Portable Document Format (PDF) 文書に変換できます。PDF 形式を使用することによって、文書を印刷する場合のグラフィカル表示を、文書を電子的に表示する場合と同じように保つことができます。

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。

関連情報

HTMLTagElement Javadoc

DirFilter Javadoc - File オブジェクトがディレクトリーかどうかの判別で使用します。

HTMLFileFilter Javadoc - File オブジェクトがファイルかどうかの判別で使用します。

URLEncoder Javadoc - 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 Javadoc - 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。

BidiOrdering クラス

IBM Toolbox for Java 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

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。
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、以下のタグが生成されます。

```
<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

関連情報

BidiOrdering Javadoc

HTMLAlign クラス

IBM Toolbox for Java 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への項目の追加またはリストからの除去
- 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

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。
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、以下のタグが作成されます。

```
<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。

- Centered unordered list
- Another item

関連情報

HTMLAlign Javadoc

HTMLDocument クラス

HTMLDocument クラスによって、既存の IBM Toolbox for Java HTML クラスを使用して HTML ページと Portable Document Format (PDF) 文書のいずれかを作成することがより容易になります。

HTMLDocument を作成する場合、それに HTML タグと Extensible Stylesheet Language (XSL) Formatting Object (FO) タグのどちらを含めるかを指定します。

- HTML ページを作成する場合、HTMLDocument クラスによって、必要なすべての HTML タグのグループ化がより簡単になります。ただし、HTML ページを印刷する場合と Web ブラウザーで表示する場合とでは表示が異なります。
- PDF 文書を作成する場合、HTMLDocument クラスによって、PDF 文書の作成に必要なすべての情報を含む XSL FO ソースを作成することができます。 PDF 文書では、文書を印刷する場合のグラフィカル表示を、文書を電子的に表示する場合と同じように保つことができます。

HTMLDocument を使用するには、XML パーサーと XSLT プロセッサーを CLASSPATH 環境変数に含める必要があります。詳細は、以下のページを参照してください。

- 11 ページの『JAR ファイル』
- 454 ページの『XML パーサーおよび XSLT プロセッサー』

生成される HTML または XSL ソース・データは、HTML を表示したり、XSL をファイルに保存したり、あるいは Java プログラムの別の部分でストリーム・データを使用したりするなど、希望する方法で処理できます。

HTML ページおよび XSL FO ソース・データの作成についての詳細は、以下のページを参照してください。

- 『HTMLDocument を使用して HTML データを作成する』
- 209 ページの『HTMLDocument を使用して XSL FO データを作成する』
- 211 ページの『例: HTMLDocument を使用する』

HTMLDocument Javadoc

HTMLDocument を使用して HTML データを作成する:

HTMLDocument は、HTML または Extensible Stylesheet Language (XSL) Formatting Object (FO) ソース・データのいずれかを作成するのに必要な情報を保持するラッパーとして機能します。 HTML ページを作成する場合、HTMLDocument クラスによって、必要なすべての HTML タグのグループ化がより簡単になります。

HTML ソース・データの生成

HTML ソースを作成する場合、HTMLDocument は作成された HTML オブジェクトから HTML タグを検索します。 HTMLDocument.getTag() を使用してユーザーが定義したすべての要素をストリームするか、それぞれの HTML オブジェクトごとに getTag() を使用することができます。

HTMLDocument は、ご使用の Java プログラムで定義したとおりに HTML データを生成します。それで、生成された HTML が完全であり正確であるかどうか確かめてください。

HTMLDocument.getTag() を呼び出すと、HTMLDocument オブジェクトは以下のアクションを実行します。

- 開始 <HTML> タグを生成します。データの最後に、終了 </HTML> タグを生成します。
- HTMLHead および HTMLMeta オブジェクトを HTML タグに変換します。
- <HEAD> タグの直後に開始 <BODY> タグを生成します。データの最後に、終了 </HTML> タグの直前に終了 </BODY> タグを生成します。

  注: <HEAD> タグを指定しなかった場合、HTMLDocument は <HTML> タグの後ろに <BODY> タグを生成します。
- 残りの HTML オブジェクトをプログラムが指図するとおりに HTML タグに変換します。

注: HTMLDocument は、ご使用の Java プログラムが指図するとおりに HTML タグをストリームして、タグが適切な順序で呼び出されるようにします。

例: HTMLDocument を使用する

以下の例では、HTMLDocument を使用して HTML ソース・データ (および XSL FO ソース) を生成する方法を示します。

214 ページの『例: HTMLDocument を使用して HTML ソースと XSL FO ソースの両方を生成する』

Javadoc 参照資料

HTMLDocument クラスについての詳細は、以下の Javadoc 参照資料を参照してください。

HTMLDocument

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。

コードに関するライセンス情報および特記事項

IBM は、お客様に、すべてのプログラム・コードのサンプルを使用することができる非独占的な著作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。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

こ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、いかなる保証も適用されません。商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任の保証の適用も一切ありません。

HTMLDocument を使用して XSL FO データを作成する:

HTMLDocument は、Extensible Stylesheet Language (XSL) Formatting Object (FO) または HTML ソース・データのいずれかを作成するのに必要な情報を保持するラッパーとして機能します。

生成された XSL FO ソースは XSL FO フォーマット・モデルに従います。このモデルは、エリアと呼ばれる四角形の要素を使用して、イメージ、テキスト、他の XSL FO といった個々のコンテンツ要素を保持することもあれば、何も保持しない場合もあります。以下のリストは、4 つの基本的なタイプのエリアを示しています。

- Regions は最高水準のコンテナーとして機能します。
- Block エリアは、パラグラフまたはリスト項目といったブロック・レベル要素を表します。
- Line エリアは、ブロック内のテキストの行を表します。
- Inline エリアは、単一文字、脚注、あるいは数式といった行の一部分を表します。

IBM Toolbox for Java によって作成された XSL FO タグは、W3C 勧告で説明されている XSL 規格に従います。 XSL、XSL FO、および W3C 勧告についての詳細は、以下を参照してください。

Extensible Stylesheet Language (XSL) バージョン 1.0

XSL FO ソース・データの生成

XSL FO ソースを作成する場合、HTMLDocument プロパティーは、ページ・サイズ、方向、およびマージンを指定する XSL FO タグを表します。さらに HTMLDocument は、多くの HTML クラスからそのコンテンツ要素に対応する XSL FO タグを検索します。

HTMLDocument を使用して XSL FO ソースを生成すると、XSL フォーマッター (例、XSLReportWriter クラス) を使用して、ある文書のページ上にコンテンツ要素を配置することができます。

HTMLDocument は、XSL FO ソース・データを以下の 2 つの主要なセクションに生成します。

- 最初のセクションには、ページの高さ、ページ幅、およびページの余白といった汎用ページ・レイアウト情報を保持する <fo:root> および <fo:layout-master-set> XSL FO タグが含まれます。レイアウト情報の値を指定するには、HTMLDocument 設定メソッドを使用して、関連したプロパティーの値を設定します。
- 2 番目のセクションには、それぞれのコンテンツ要素を保持する XSL FO <fo:page-sequence> タグが含まれます。 HTML クラスのインスタンスであるそれぞれのコンテンツ要素を指定するには、対応する XSL FO タグを HTML オブジェクトから検索します。コンテンツ要素には、getFoTag() メソッドを持つ HTML クラスのみを使用してください。

  注: getFoTag() メソッドを持たない HTML クラスから XSL FO タグを検索しようとすると、コメント・タグが現れます。

XSL FO タグを処理するためのメソッドを含む HTML クラスについての詳細は、以下の Javadoc 参照資料を参照してください。

211 ページの『XSL FO 対応クラス』

HTMLDocument のインスタンスを作成してレイアウト・プロパティーを設定した後、setUseFO()、getFoTag()、および getTag() メソッドを使用して、HTML オブジェクトから XSL FO タグを検索してください。

- HTMLDocument またはそれぞれの HTML オブジェクト上のいずれかで setUseFO() を使用できます。setUseFO() を使用する場合、HTMLDocument.getTag() を使用して XSL FO タグを検索できます。
- あるいは、HTMLDocument またはそれぞれの HTML オブジェクト上のいずれかで getFoTag() メソッドを使用できます。 HTMLDocument または HTML オブジェクトから XSL FO と HTML ソースの両方を生成する必要がある場合に、この代わりのメソッドを使用することもできます。

例: HTMLDocument を使用する

XSL FO ソース・データを作成した後、その XSL FO データをユーザーが表示および印刷できるフォームに変換する必要があります。以下の例では、XSL FO ソース・データ (および HTML ソース) を生成する方法、および XSL FO ソース・データを含むファイルを XSLReportWriter および Context クラスを使用して PDF 文書に変換する方法を示します。

214 ページの『例: HTMLDocument を使用して HTML ソースと XSL FO ソースの両方を生成する』

212 ページの『例: XSL FO ソース・データを PDF に変換する』

Javadoc 参照資料

HTMLDocument クラスについての詳細は、以下の Javadoc 参照資料を参照してください。

HTMLDocument

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。

コードに関するライセンス情報および特記事項

IBM は、お客様に、すべてのプログラム・コードのサンプルを使用することができる非独占的な著作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。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

こ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、いかなる保証も適用されません。商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任の保証の適用も一切ありません。

XSL FO 対応クラス:

このトピックでは、HTMLDocument クラスと互換性のある IBM Toolbox for Java クラスについて説明します。

多くの IBM Toolbox for Java HTML クラスは以下のメソッドをフィーチャーしており、これらのクラスのインスタンスで HTMLDocument を処理できるようにします。

- getFoTag()
- getTag()
- setUseFO()

XSL FO を処理するためのメソッドが含まれる HTMLDocument クラスおよび HTML クラスについての詳細は、以下の Javadoc 参照資料を参照してください。

- HTMLDocument
- BidiOrdering
- HTMLAlign
- HTMLHead
- HTMLHeading
- HTMLImage
- HTMLList
- HTMLListItem
- HTMLTable
- HTMLTableCaption
- HTMLTableCell
- HTMLTableHeader
- HTMLTableRow
- HTMLTagElement
- OrderedList
- UnorderedList

例: HTMLDocument を使用する:

以下の例では、HTMLDocument クラスを使用して、HTML および Extensible Stylesheet Language (XSL) Formatting Object (FO) ソース・データを生成する方法を示します。

例: HTMLDocument を使用して HTML ソースと XSL FO ソースの両方を生成する

以下の例では、HTML ソース・データと XSL FO ソース・データの両方を同時に生成する方法を示します。

214 ページの『例: HTMLDocument を使用して HTML ソースと XSL FO ソースの両方を生成する』

例: XSL FO ソース・データを PDF に変換する

XSL FO ソース・データを作成した後、その XSL FO データをユーザーが表示および印刷できるフォームに変換する必要があります。以下の例では、XSL FO ソース・データを含むファイルを XSLReportWriter および Context クラスを使用して PDF 文書に変換する方法を示します。

『例: XSL FO ソース・データを PDF に変換する』

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。

コードに関するライセンス情報および特記事項

IBM は、お客様に、すべてのプログラム・コードのサンプルを使用することができる非独占的な著作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。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

こ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、いかなる保証も適用されません。商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任の保証の適用も一切ありません。

例: XSL FO ソース・データを PDF に変換する:

今後、XSLReportProcessor クラスはサポートされないので、このプログラム例は、使用しないでください。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////////////////////
//
// Example: Converting XSL FO source to a PDF.
//
// This program uses the IBM Toolbox for Java ReportWriter classes to convert
// XSL FO source data (created by using HTMLDocument) to a PDF.
//
// This example requires the following.jars to be in the classpath.
//
//   composer.jar
//   outputwriters.jar
//   reportwriter.jar
//   x4j400.jar
//   xslparser.jar
//
// These JAR files are part of the IBM ToolBox for Java, and reside in directory
// /QIBM/ProdData/HTTP/Public/jt400/lib on your server.
//
// You will also need the class definition for
```

```
// org/apache/xerces/dom/NodeContainer, which resides
// in directory /QIBM/ProdData/OS400/xml/lib.
//
// Command syntax:
//    ProcessXslFo FOfilename PDFfilename
//
////////////////////////////////////////////////////////////////////////////


import java.io.FileInputStream;
import java.io.FileOutputStream;

import java.awt.print.Paper;
import java.awt.print.PageFormat;

import org.w3c.dom.Document;

import com.ibm.xsl.composer.framework.Context;

import com.ibm.as400.util.reportwriter.pdfwriter.PDFContext;
import com.ibm.as400.util.reportwriter.processor.XSLReportProcessor;

public class ProcessXslFo
{
    public static void main(String args[])
    {
        if (args.length != 2)
        {
            System.out.println("Usage:  java ProcessXslFo <fo file name> <pdf file name>");
            System.exit(0);
        }

        try
        {
            String inName = args[0];
            String outName = args[1];

            /* Input. File containing XML FO. */
            FileInputStream fin = null;

            /* Output. Which in this example will be PDF. */
            FileOutputStream fout = null;

            try
            {
                fin  = new FileInputStream(inName);
                fout = new FileOutputStream(outName);
            }
            catch (Exception e)
            {
                e.printStackTrace();
                System.exit(0);
            }

            /*
            * Setup Page format.
            */
            Paper paper = new Paper();
            paper.setSize(612, 792);
            paper.setImageableArea(0, 0, 756, 936);

            PageFormat pageFormat = new PageFormat();
            pageFormat.setPaper(paper);

            /*
            * Create a PDF context. Set output file name.
            */
```

```
            PDFContext pdfContext = new PDFContext(fout, pageFormat);

            /*
            * Create XSLReportProcessor instance.
            */
            XSLReportProcessor report = new XSLReportProcessor(pdfContext);

            /*
            * Open XML FO source.
            */
            try
            {
                report.setXSLFOSource(fin);
            }
            catch (Exception e)
            {
                e.printStackTrace();
                System.exit(0);
            }

            /*
            * Process the report.
            */
            try
            {
                report.processReport();
            }
            catch (Exception e)
            {
                e.printStackTrace();
                System.exit(0);
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();
            System.exit(0);
        }

        /* exit */
        System.exit(0);
    }
}
```

例: HTMLDocument を使用して HTML ソースと XSL FO ソースの両方を生成する:

この例は、HTMLDocument クラスを使用して、HTML および XSL FO ソース・データを生成します。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////////////////////
//
// Example: Using the Toolbox HTMLDocument Class
// to generate both HTML and XSL FO source data.
//
// This program uses the HTMLDocument class to
// generate two files: one that has HTML source and
// another than has XSL FO source.
//
// Command syntax:
//    HTMLDocumentExample
//
////////////////////////////////////////////////////////////////////////////


import com.ibm.as400.util.html.*;
```

```
import java.*;
import java.io.*;
import java.lang.*;
import java.beans.PropertyVetoException;

public class HTMLDocumentExample
{
    public static void main (String[] args)
    {
        //Create the HTMLDocument that holds necessary document properties
        HTMLDocument doc = new HTMLDocument();

        //Set page and margin properties.  Numbers are in inches.
        doc.setPageWidth(8.5);
        doc.setPageHeight(11);
        doc.setMarginTop(1);
        doc.setMarginBottom(1);
        doc.setMarginLeft(1);
        doc.setMarginRight(1);

        //Create a header for the page.
        HTMLHead head = new HTMLHead();
        //Set the title for the header
        head.setTitle("This is the page header.");

        //Create several headings
        HTMLHeading h1 = new HTMLHeading(1, "Heading 1");
        HTMLHeading h2 = new HTMLHeading(2, "Heading 2");
        HTMLHeading h3 = new HTMLHeading(3, "Heading 3");
        HTMLHeading h4 = new HTMLHeading(4, "Heading 4");
        HTMLHeading h5 = new HTMLHeading(5, "Heading 5");
        HTMLHeading h6 = new HTMLHeading(6, "Heading 6");

        //Create some text that is printed from right to left.
        //Create BidiOrdering object and set the direction
        BidiOrdering bdo = new BidiOrdering();
        bdo.setDirection(HTMLConstants.RTL);

        //Create some text
        HTMLText text = new HTMLText("This is Arabic text.");
        //Add the text to the bidi-ordering object
        bdo.addItem(text);

        // 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 olistItem1 = new OrderedListItem();
        OrderedListItem olistItem2 = new OrderedListItem();
        OrderedListItem olistItem3 = new OrderedListItem();
        // Set the data in the OrderedListItems.
        olistItem1.setItemData(new HTMLText("First item"));
        olistItem2.setItemData(new HTMLText("Second item"));
        olistItem3.setItemData(new HTMLText("Third item"));
        // Add the list items to the OrderedList.
        oList.addListItem(olistItem1);
        oList.addListItem(olistItem2);
```

```
// Add (nest) the unordered list to OrderedListItem2
oList.addList(uList);
// Add another OrderedListItem to the OrderedList
// after the nested UnorderedList.
oList.addListItem(olistItem3);

// Create a default HTMLTable object.
HTMLTable table = new HTMLTable();
try
{
    // Set the table attributes.
    table.setAlignment(HTMLTable.LEFT);
    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< 5; rowIndex++)
    {
        HTMLTableRow row = new HTMLTableRow();
        row.setHorizontalAlignment(HTMLTableRow.CENTER);

        HTMLText account = new HTMLText("000" + rowIndex);
        HTMLText name = new HTMLText("Customer" + rowIndex);
        HTMLText balance = new HTMLText("" + (rowIndex + 1)*200);

        row.addColumn(new HTMLTableCell(account));
        row.addColumn(new HTMLTableCell(name));
        row.addColumn(new HTMLTableCell(balance));

        // Add the row to the table.
        table.addRow(row);
    }
}
catch(Exception e)
{
    System.out.println("Problem creating table");
    System.exit(0);
}

//Add the items to the HTMLDocument
doc.addElement(head);
doc.addElement(h1);
doc.addElement(h2);
doc.addElement(h3);
doc.addElement(h4);
doc.addElement(h5);
doc.addElement(h6);
doc.addElement(oList);
doc.addElement(table);
doc.addElement(bdo);
```

```
        //Print the fo tags to a file.
        try
        {
            FileOutputStream fout = new FileOutputStream("FOFILE.fo");
            PrintStream pout = new PrintStream(fout);
            pout.println(doc.getFOTag());
        }
        catch (Exception e)
        {
            System.out.println("Unable to write fo tags to FOFILE.fo");
        }

        //Print the html tags to a file
        try
        {
            FileOutputStream htmlout = new FileOutputStream("HTMLFILE.html");
            PrintStream phtmlout = new PrintStream(htmlout);
            phtmlout.println(doc.getTag());
        }
        catch (Exception e)
        {
            System.out.println("Unable to write html tags to HTMLFILE.html");
        }

    }
}
```

HTML フォーム・クラス

IBM Toolbox for Java HTMLForm クラスは HTML フォームを表します。このクラスを使用すれば、CGI スクリプトを使った場合より簡単にフォームを作成することができます。

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。

もちろん、HTMLText、HTMLHyperlink、および HTMLTable も含め、他のタグ要素もフォームに追加できます。

HTMLForm Javadoc

235 ページの『HTML Text クラス』

IBM Toolbox for Java 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、多くのテキスト属性の状況を取得、設定、および検査できます。

227 ページの『HTMLHyperlink クラス』
IBM Toolbox for Java 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。

233 ページの『HTML Table クラス』
IBM Toolbox for Java 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。

645 ページの『例: HTML フォーム・クラスを使用する』
以下の IBM Toolbox for Java 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。

655 ページの『HTML クラスの出力例』
以下に示すのは、HTML クラスの例を実行すると生成されるサンプル出力です。

FormInput クラス:

IBM Toolbox for Java FormInput クラスは、 HTML フォームの入力要素を表します。

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。

FormInput Javadoc

ToggleFormInput Javadoc

RadioFormInputGroup Javadoc

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()" />
```

関連情報

ButtonFormInput Javadoc

FileFormInput クラス:

IBM Toolbox for Java 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" />
```

関連情報

FileFormInput Javadoc

HiddenFormInput クラス:

IBM Toolbox for Java 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ます。

関連情報

HiddenFormInput Javadoc

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

例: ImageFormInput オブジェクトを作成する

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" />
```

関連情報

ImageFormInput Javadoc

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" />
```

関連情報

ResetFormInput Javadoc

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" />
```

関連情報

SubmitFormInput Javadoc

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" />
```

関連情報

TextFormInput Javadoc

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" />
```

関連情報

PasswordFormInput Javadoc

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" />
```

RadioFormInput Javadoc

RadioFormInputGroup Javadoc

CheckboxFormInput クラス:

IBM Toolbox for Java 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:

IBM Toolbox for Java 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 によって提供され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することができます。

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

LayoutFormPanel Javadoc

『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。

223 ページの『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。

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>
```

関連情報

GridLayoutFormPanel Javadoc

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。

例: LineLayoutFormPanel を使用する

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/>
```

関連情報

LineLayoutFormPanel Javadoc

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・エリアのサイズを指定します。 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>
```

TextAreaFormElement Javadoc

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。

例: LabelFormElement を使用する

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
```

関連情報

LabelFormElement Javadoc

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>
```

関連資料

『SelectOption クラス』
SelectOption クラスは、HTML SelectFormElement 内のオプションを表します。 選択フォーム内ではオプション・フォーム要素を使用します。

関連情報

SelectFormElement Javadoc

SelectOption クラス:

SelectOption クラスは、HTML SelectFormElement 内のオプションを表します。選択フォーム内ではオプション・フォーム要素を使用します。

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>
```

関連資料

224 ページの『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。

関連情報

SelectOption Javadoc

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
```

関連情報

RadioFormInputGroup Javadoc

RadioFormInput Javadoc

HTMLHead クラス

IBM Toolbox for Java HTMLHead クラスは、HTML ヘッド・タグを表します。HTML ページのヘッド・セクションは、通常他のタグを含む開始および終了ヘッド・タグをフィーチャーします。一般に、ヘッド・タグにはタイトル・タグが含まれ、メタ・タグも含まれる場合があります。

HTMLHead のコンストラクターを使用すると、空になっていたり、タイトル・タグが含まれていたり、あるいはタイトル・タグとメタ・タグが含まれていたりするヘッド・タグを構成することができます。タイトルおよびメタ・タグを空の HTMLHead オブジェクトに簡単に追加できます。

HTMLHead クラスのメソッドには、ページのタイトルおよびメタ・タグの設定と取得が含まれます。HTMLMeta クラスを使用してメタ・タグの内容を定義します。

以下のコードは、HTMLHead タグを作成する 1 つの方法を示しています。

```
// Create an empty HTMLHead.
HTMLHead head = new HTMLHead("My Main Page");

// Add the title.
head.setTitle("My main page");

// Define your meta information and add it to HTMLHead.
HTMLMeta meta = new HTMLMeta("Content-Type", "text/html; charset=iso-8859-1");
head.addMetaInformation(meta);
```

これは、HTMLHead タグの出力例です。

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"/>
<title>My main page</title>
</head>
```

関連資料

231 ページの『HTMLMeta クラス』
IBM Toolbox for Java 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。

関連情報

HTMLHead Javadoc

HTMLHeading クラス

IBM Toolbox for Java HTMLHeading クラスは HTML ヘッディングを表します。各ヘッディングには、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。

- 見出しのテキストの取得および設定
- ヘッダーのレベルの取得および設定
- ヘッダーの位置合わせの取得および設定
- 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。
- HTMLHeading オブジェクトのストリング表記の取得 (Get a String representation)

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>
```

関連情報

HTMLHeading Javadoc

HTMLHyperlink クラス

IBM Toolbox for Java 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 を取得および設定する
- 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");
toolbox.setTarget(TARGET_BLANK);

// Display the toolbox link tag.
System.out.println(toolbox.toString());
```

上記のコードは、以下のタグを作成します。

```
<a href="http://www.ibm.com/as400/toolbox">IBM Toolbox for Java home page</a>
```

関連情報

HTMLHyperlink Javadoc

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" />
```

関連情報

HTMLImage Javadoc

HTMLList クラス

IBM Toolbox for Java 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: 順序なしリストを作成する

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: HTMLList クラスを使用する

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、および順序なし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>
```

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>
```

例: 順序なしリストを作成する

以下の例では、順序なし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>
```

HTMLList Javadoc

HTMLListItem Javadoc

HTMLMeta クラス

IBM Toolbox for Java 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">
```

関連情報

HTMLMeta Javadoc

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、以下のタグが作成されます。

```
<param name="age" value="21">
```

関連情報

HTMLParameter Javadoc

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>
```

HTMLServlet Javadoc

232 ページの『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。

HTML Table クラス

IBM Toolbox for Java 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

例: HTMLTable クラスを使用する

以下の例では、HTMLTable クラスを使用する方法を示します。

665 ページの『例: HTMLTable クラスを使用する』

関連情報

HTMLTable Javadoc

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、コンストラクター上で設定するか、または 2 つの setElement() メソッドのいずれかを介し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">IBM Home Page</td>

HTMLTableCell Javadoc

HTMLTagElement Javadoc

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。

HTMLTableRow クラスを使用して、以下を行うことができます。

- 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);
```

関連情報

HTMLTableRow Javadoc

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);
```

関連情報

HTMLTableHeader Javadoc

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);
```

関連情報

HTMLTableCaption Javadoc

HTML Text クラス

IBM Toolbox for Java 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、多くのテキスト属性の状況を取得、設定、および検査できます。

そのような属性には、次のものがあります。

- 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、以下のタグが生成されます。

```
<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

関連情報

HTMLText Javadoc

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 サーブレット要求を取得したり、設定したりする。
- 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。

例: HTMLTree クラスを使用する

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。

- 656 ページの『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』

関連情報

HTMLTree Javadoc

例: 走査可能な統合ファイル・システム・ツリーを作成する:

以下の例は 3 つのファイルで構成されており、それらを合わせることによって、走査可能な統合ファイル・システム・ツリーの作成方法を示しています。 この例では、HTMLTree および FileListElement をサーブレットで表示するためにフレームを使用します。

- 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。

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>
```

関連情報

HTMLTreeElement Javadoc

FileTreeElement クラス:

IBM Toolbox for Java 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、ユーザーは、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。

例: FileTreeElement を使用する

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。

関連情報

FileTreeElement Javadoc

FileListElement クラス:

IBM Toolbox for Java 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を取得および設定する。
- NetServer 共用ドライブのパスを取得および設定する。

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。

例: FileListElement を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

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。

236 ページの『例: 走査可能な統合ファイル・システム・ツリーを作成する』

関連情報

FileListElement Javadoc

FileTreeElement Javadoc

HTMLTreeElement Javadoc

FileListRenderer クラス:

IBM Toolbox for Java 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、レンダラーによって 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、メソッドをオーバーライド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、適切なメソッドをオーバーライドすることにより、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、行データへの列の追加や、列の再配置があります。 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。

関連資料

238 ページの『FileListElement クラス』
IBM Toolbox for Java 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。

関連情報

FileListRenderer Javadoc

ReportWriter クラス

com.ibm.as400.util.reportwriter パッケージでは、XML ソース・ファイルからのデータや、サーブレットまたは JavaServer Pages で作成されたデータに、簡単にアクセスしてフォーマットするするためのクラスが提供されます。

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ができます。 XML パーサーや XSLT プロセッサーを含む、必要な JAR ファイルが CLASSPATH の中にあることを確認してください。詳細は、以下のページを参照してください。

- JAR ファイル

(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 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。

Context クラス

IBM Toolbox for Java の 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 クラスとの組み合わせによって、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。例: PDFContext を指定した JSPReportProcessor を使用することにより、他の ReportWriter クラスと共に Context クラスを構成および使用してレポートを生成する方法を示します。

OutputQueue Javadoc

SpooledFileOutputStream Javadoc

240 ページの『ReportWriter クラス』
com.ibm.as400.util.reportwriter パッケージでは、XML ソース・ファイルからのデータや、サーブレットまたは JavaServer Pages で作成されたデータに、簡単にアクセスしてフォーマットするするためのクラスが提供されます。

681 ページの『例: PCLContext と共に XSLReportProcessor を使用する』
今後、XSLReportProcessor クラスはサポートされないので、以下の例は、使用しないでください。

677 ページの『例: PDFContext と共に JSPReportProcessor を使用する』
この例は JSPReportProcessor および PDFContext クラスを使用して、指定した URL からデータを取得し、そのデータを PDF フォーマットに変換します。 そのデータは次に、PDF 文書としてファイルにストリームされます。

JSPReportProcessor クラス

JSPReportProcessor クラスにより、 JavaServer Page (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 フォーマット・オブジェクト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 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。

- 677 ページの『例: PDFContext と共に JSPReportProcessor を使用する』
- 678 ページの『例: JSPReportProcessor サンプル JSP ファイル』

Java Server Pages technology

XSLReportProcessor クラス

今後、XSLReportProcessor クラスはサポートされないので、使用しないでください。

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

例

以下の例は、 XSLReportProcessor および PCLContext クラスを使用して報告書を生成する方法を示します。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:

- 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 Toolkit のトピックを参照してください。

リソース・クラス

リソース・パッケージとそのクラスは、推奨できなくなりました。代わって、アクセス・パッケージの使用をお勧めします。

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、システム・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、システム・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

リソース・パッケージとそのクラスは、推奨できなくなりました。代わって、アクセス・パッケージの使用をお勧めします。

com.ibm.as400.resource.Resource および com.ibm.as400.resource.ChangeableResource 抽象クラスは、IBM i 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 - 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 - サーバー上の Java プログラムを表します。
- RJob - IBM i ジョブを表します。
- RPrinter - IBM i プリンターを表します。
- RQueuedMessage - IBM i 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 - システム上のライセンス・プログラムを表します。
- RUser - IBM i 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、システム・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

リソース・パッケージとそのクラスは、推奨できなくなりました。代わって、アクセス・パッケージの使用をお勧めします。

com.ibm.as400.resource.ResourceList クラスは、システム・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 - 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 - システム・ジョブのリストを表します。
- RJobLog - システム・ジョブ・ログ内のメッセージのリストを表します。
- RMessageQueue - システム・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 - システム・プリンターのリストを表します。
- RUserList - システム・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の待機
- 完全なリソース・リストのロードの待機

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)

コードのサンプルに関する特記事項

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。

IBM は、お客様に、すべてのプログラム・コードのサンプルを使用することができる非独占的な著作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。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

こ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、いかなる保証も適用されません。商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任の保証の適用も一切ありません。

Presentation クラス

リソース・パッケージとそのクラスは、推奨できなくなりました。代わって、アクセス・パッケージの使用をお勧めします。

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 ID 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) を使用する通信インフラストラクチャーは、クライアントとサーバー間のセッション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。
- 認証サービスは、以下の機能を提供します。
  - ユーザーの ID およびパスワードを IBM i ユーザー・レジストリーと照合して認証する。
  - 現行の IBM i スレッドに ID を割り当てる。

Secure Sockets Layer

Secure Sockets Layer (SSL) は、クライアントとサーバー間のセッション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。

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セキュリティーの方がパフォーマンスより優先度が高い場合 (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。

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 クラスと IBM i サーバーの間で暗号化を使用する

SSL を使用して IBM Toolbox for Java と IBM i 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 IBM i 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・サイドでは、データの暗号化には JSSE を使用します。サーバー側では、 IBM i デ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 IBM i 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 IBM i 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. 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 IBM i のセットアップ:

IBM Toolbox for Java で SSL を使用するためにシステムをセットアップするには、次のようなステップを行います。

1. サーバー証明書を取得および構成します。
2. IBM Toolbox for Java によって使用される以下の システム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

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 (5770-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、システム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、システム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、 デジタル証明書マネージャーを使用します。
  1. システム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。

認証サービス

IBM Toolbox for Java では、 IBM i 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 ID (*プリンシパル* と呼ばれることもある) とパスワードを IBM i 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状を確立することができます。信任状を使用すると、現行 IBM i 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。

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、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。

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。

注: ProfileTokenCredential クラスを使用する場合は、このページの下部にある、トークンを設定するメソッドに関する説明を必ず検討してください。

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

信任状は、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。信任状を作成すると、呼び出しアプリケーションの要求に応じて、これを直列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、 IBM i 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代行処理作業を実行するために、信任状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 IBM i 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悪意ある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、信任状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。

ProfileTokenCredential 中のトークン設定メソッド

ProfileTokenCredential クラス中の、トークンを設定するためのメソッドでは、パスワードを指定するさまざまな方法を区別する必要があります。

- *NOPWD や *NOPWDCHK などの特殊値として、定義済みの特殊値の整数を使用する。
- ユーザー・プロファイルのパスワードとして、パスワードを表すストリングを使用する。

注: V5R3 では、パスワードの指定方法を区別する必要のない setToken メソッドを IBM Toolbox for Java で使用すべきではありません。

さらに setToken メソッドを使用すると、リモート・ユーザーがパスワードの特殊値を指定できますし、最大 128 文字の長いユーザー・プロファイル・パスワードを使用できます。

*NOPWD や *NOPWDCHK などのパスワード特殊値の整数を指定するには、以下のいずれかのメソッドを使用してください。

- setToken(AS400Principal principal, int passwordSpecialValue)
- setToken(String name, int passwordSpecialValue)

ProfileTokenCredential クラスは、パスワード特殊値の整数として以下の静的定数を組み込みます。

- ProfileTokenCredential.PW_NOPWD: *NOPWD を示す。
- ProfileTokenCredential.PW_NOPWDCHK: *NOPWDCHK を示す。

ユーザー・プロファイル・パスワードをストリングとして指定するには、以下のいずれかのメソッドを使用してください。

- setTokenExtended(AS400Principal principal, String password)
- setTokenExtended(String name, String password)

setTokenExended メソッドを使用して、パスワード特殊値のストリングをパスワード・パラメーターとして渡すことはできません。例えば、これらのメソッドでは *NOPWD のパスワード・ストリングを使用できません。

詳細は、ProfileTokenCredential Javadoc の参照情報を参照してください。

例

プロファイル・トークン信任状を使用して IBM i スレッドの識別を交換し、特定のユーザーに代わって代行処理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。

AS400 Javadoc

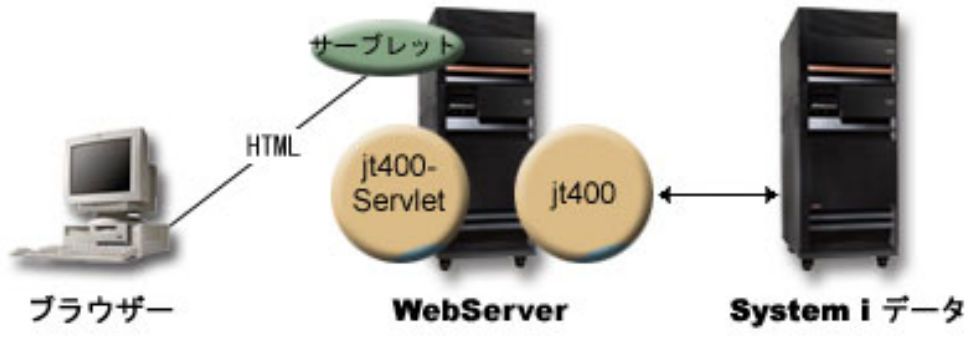
ProfileTokenCredential Javadoc

サーブレット・クラス

IBM Toolbox for Java に付属しているサーブレット・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 IBM i 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サーバーに接続されます。

図 1: サーブレットの仕組み



250 ページの『図 1 の詳細説明: サーブレットの働き (rzahh585.gif)』

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。

図 1 の詳細説明: サーブレットの働き (rzahh585.gif)

『IBM Toolbox for Java: Servlet クラス』にある図

この図は、サーブレットの一般的な働きを図解しています。

説明

この図は以下のもので構成されています。

- 左側にあるパーソナル・コンピューターのイメージには「ブラウザー」のラベルがあり、それはパーソナル・コンピューター上で実行中のブラウザーのインスタンスを表します。
- 右側にある IBM i サーバーのイメージには「System i データ」のラベルがあり、サーブレットにアクセスさせたいデータの場所を表します。
- 中央 (他の 2 つのイメージの間) にある IBM i サーバーのイメージには「Web サーバー」のラベルがあり、 Web サーバーを表します。 Web サーバーのイメージ上にある、ラベルが付いたいくつかの形状は、 Web サーバー上にあるファイルあるいはファンクションを示しています。
  - サーブレットのラベルが付いた緑の楕円はサーブレット・コードの場所を表しています。
  - jt400Servlet のラベルが付いた黄褐色の円は、jt400Servlet.jar ファイルの場所を示しています。
  - jt400 のラベルが付いた黄褐色の円は、jt400.jar ファイルの場所を示しています。

  注: Web サーバーは IBM i サーバー上にある必要はありませんが、そこに置くこともできます。さらに、System i データのイメージにより示されているサーバーと同じサーバーにすることさえ可能です。
- イメージを結ぶ線

HTML のラベルが付いている線は、ブラウザー (左のイメージ) と Web サーバー (中央のイメージ) 上のサーブレット (緑の楕円) を結んでいます。その線に HTML のラベルが付いている理由は、サーブレットはたいていの場合 HTML を使用してデータをブラウザーに 'サーブ (供給)' するからです。

Web サーバーは、IBM Toolbox for Java の 2 つの JAR ファイル (黄褐色の円) を実行しています。jt400Servlet.jar 内のクラスは jt400.jar 内のクラスと共に、 IBM i データ (右側のイメージ) を含むサーバーに容易に接続するサーブレットを、 Web サーバーが実行することを可能にします。 2 つのイメージを結んでいる、両端に矢印のある線は、この接続を表しています。

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 のサブクラスは、次のメソッドの 1 つまたは複数のものを変更します。

- 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持つことができるサーブレットへの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。

AuthenticationServlet Javadoc

AS400Servlet Javadoc

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()

設定メソッド		取得メソッド
beforeFirst()	relative()	isBeforeFirst()
first()		isFirst()
last()		isLast()

関連情報

RowData Javadoc

ListRowData クラス:

IBM Toolbox for Java ListRowData クラスは、テーブル・フォームのデータのリストを表します。このテーブルでは、各行は、ListMetaData オブジェクトによって判別された有限数の列から成り、1 行中の各列には、個々のデータ項目が入っています。データは、統合ファイル・システム内のディレクトリー、ジョブのリスト、プリンターのリスト、または他の各種データのいずれでもかまいません。

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

例

以下の例では、ListRowData および HTMLTableConverter クラスの動作方法を示します。また、この例では Java コード、HTML コード、および HTML の外観を示します。

700 ページの『例: ListRowData クラスを使用する』

ListRowData Javadoc

RecordListRowData クラス:

IBM Toolbox for Java RecordListRowData クラスを使用して、以下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・フォーマットの設定
- レコード・フォーマットの取得

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。

関連情報

RecordListRowData Javadoc

ResourceListRowData クラス:

IBM Toolbox for Java 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 の取得と設定
- リソース・リストの取得と設定
- リスト内の行数の取り出し
- 現在行の列データの取得
- データ・オブジェクトのプロパティー・リストの取得
- リスト用のメタデータの取得

例: サーブレットでリソース・リストを表示する

コードのサンプルに関する特記事項

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。

IBM は、お客様に、すべてのプログラム・コードのサンプルを使用することができる非独占的な著作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。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

こ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、いかなる保証も適用されません。商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任の保証の適用も一切ありません。

関連資料

244 ページの『リソース・リスト』
リソース・パッケージとそのクラスは、推奨できなくなりました。代わって、アクセス・パッケージの使用をお勧めします。

関連情報

ResourceListRowData Javadoc

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。

関連資料

67 ページの『JDBC クラス』
JDBC 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。

関連情報

SQLResultSetRowData Javadoc

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

関連情報

RowMetaData Javadoc

ListMetaData クラス:

IBM Toolbox for Java ListMetaData クラスを使用すれば、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。

例

以下の例では、 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。

700 ページの『例: ListRowData クラスを使用する』

ListMetaData Javadoc

252 ページの『ListRowData クラス』
IBM Toolbox for Java ListRowData クラスは、テーブル・フォームのデータのリストを表します。 このテーブルでは、各行は、ListMetaData オブジェクトによって判別された有限数の列から成り、1 行中の各列には、個々のデータ項目が入っています。 データは、統合ファイル・システム内のディレクトリー、ジョブのリスト、プリンターのリスト、または他の各種データのいずれでもかまいません。

RecordFormatMetaData クラス:

RecordFormatMetaData は、IBM Toolbox for Java 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・フォーマットにアクセスするときに、レコード・フォーマット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));
}
```

関連情報

RecordFormatMetaData Javadoc

RecordFormat Javadoc

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));
   }
}
```

SQLResultSetMetaData Javadoc

254 ページの『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。

コンバーター・クラス

IBM Toolbox for Java 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 フォーマットになって、 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 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。

関連情報

StringConverter Javadoc

HTMLFormConverter クラス:

IBM Toolbox for Java 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: HTMLFormConverter クラスを使用する

以下の例では、HTMLFormConverter の使用について説明します。(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うことができます。)

HTMLFormConverter クラスを使用する

関連資料

『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。

関連情報

HTMLFormConverter Javadoc

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 クラスを使用する
- 例: RecordListRowData クラスを使用する
- 例: SQLResultSetRowData クラスを使用する
- 例: サーブレットで ResourceList を表示する

コードのサンプルに関する特記事項

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。

HTMLTableConverter Javadoc

ユーティリティー・クラス

ユーティリティー・クラスは、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。

IBM Toolbox for Java は、次のユーティリティーを提供します。

クライアント・インストールおよびクラスの更新

インストールや更新を目的とする場合はたいてい、サーバーの統合ファイル・システム上のロケーションにある IBM Toolbox for Java クラスを参照することができます。

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。しかし、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、 IBM i 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。 IBM i Access for Windows は、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。

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 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 unzip 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、 Javadoc の中の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

AS400ToolboxJarMaker を使用して、以下の幾つかの方法で JAR ファイルを処理することができます。

- JAR ファイルに組み込まれているファイルの中から 1 つを解凍する
- 大きな JAR ファイルを小さい JAR ファイルに分割する
- アプリケーションが実行する必要のない IBM Toolbox for Java ファイルを除外する

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 300KB 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、 400KB 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。

JarMaker Javadoc

AS400ToolboxJarMaker Javadoc

IBM Toolbox for Java でサポートされているコンポーネント:

以下のテーブルには、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 がリストされます。

- コンポーネント列には、コンポーネントの共通名のリストが示されています。

- キーワード列には、-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。
- 定数列には、 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	メッセージ	AS400ToolboxJarMaker.MESSAGE
数値データ型	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

コンポーネント	キーワード	定数
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:

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 IBM i システムに (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、例外がスロー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 IBM i の CCSID をサポートします。

IBM i プラットフォームで認識されている CCSID の全リストを含め、 CCSID のその他の詳細は、グローバリゼーションを参照してください。

IBM Toolbox for Java でサポートされる CCSID

CCSID	フォーマット	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	ラテン語 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 (ラテン語 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	ラテン語 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

CCSID	フォーマット	説明
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)

CCSID	フォーマット	説明
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	ラテン語 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 ラテン語 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)

CCSID	フォーマット	説明
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	IBM i の Unicode (主に統合ファイル・システムで使用)
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

CommandHelpRetriever クラス

CommandHelpRetriever クラスは、 IBM i 制御言語 (CL) コマンドのヘルプ・テキストを検索し、そのテキストを HTML またはユーザー・インターフェース管理機能 (UIM) 形式で生成します。コマンド行から CommandHelpRetriever を実行するか、またはその機能を Java プログラムに組み込むことができます。

CommandHelpRetriever を使用するには、サーバーで CLASSPATH 環境変数に XML パーサーおよび XSL プロセッサーを指定していなければなりません。詳しくは、 454 ページの『XML パーサーおよび XSLT プロセッサー』を参照してください。

また、 Generate Command Documentation (GENCMDDOC) CL コマンドは CommandHelpRetriever クラスを使用します。したがって、GENCMDDOC コマンドを使用するだけで、 CommandHelpRetriever クラスによって提供される機能を利用できます。詳細は、CL Reference トピックのGenerate Command Documentation (GENCMDDOC) を参照してください。

コマンド行からの CommandHelpRetriever の実行

CommandHelpRetriever クラスは、スタンドアロン型コマンド行プログラムとして実行できます。コマンド行から CommandHelpRetriever を実行するには、以下の必須パラメーターを渡さなければなりません。

- サーバー上の CL コマンドを含むライブラリー。システム・コマンドは QSYS ライブラリー中にあります。
- CL コマンド。

サーバー、ユーザー ID、パスワード、および生成されるファイルの場所を含むオプション・パラメーターを CommandHelpRetriever に渡すこともできます。

詳細については、 CommandHelpRetriever の Javadoc 参照資料を参照してください。

例: コマンド行から CommandHelpRetriever を使用する

以下の例では、現行ディレクトリー中に CRTLIB.html という HTML ファイルを生成します。

注: この例のコマンドは、表示上の理由で 2 行になっています。実際には 1 行でコマンドを入力してください。

```
java com.ibm.as400.util.CommandHelpRetriever -library QSYS -command CRTLIB
     -system MySystem -userid MyUserID -password MyPassword
```

プログラムへの CommandHelpRetriever クラスの組み込み

CommandHelpRetriever クラスを Java アプリケーション中で使用して、指定された CL コマンドのヘルプ文書を表示することもできます。 CommandHelpRetriever オブジェクトの作成後に、 generateHTML および generateUIM メソッドを使用して、どちらかの形式でヘルプ文書を生成できます。

generateHTML() を使用する際には、生成された HTML を当該コマンドのパネル・グループ内で表示したり、別のパネル・グループを指定したりできます。

以下の例では、CommandHelpRetriever オブジェクトを作成し、 CRTLIB コマンドの HTML および UIM ヘルプ文書を表す String オブジェクトを生成します。

```
CommandHelpRetriever helpGenerator = new CommandHelpRetriever();
AS400 system = new AS400("MySystem", "MyUserID", "MyPassword");
Command crtlibCommand = new Command(system, "/QSYS.LIB/CRTLIB.CMD");
String html = helpGenerator.generateHTML(crtlibCommand);
String uim = helpGenerator.generateUIM(crtlibCommand);
```

関連情報

CommandHelpRetriever Javadoc

CommandPrompter クラス

IBM Toolbox for Java 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 クラスは、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、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
- jhall.jar

さらに、XML パーサーが CLASSPATH に組み込まれている必要があります。適切な XML パーサーの使用についての詳細は、以下のページを参照してください。

454 ページの『XML パーサーおよび XSLT プロセッサー』

jhall.jar 以外のすべての JAR ファイルは、IBM Toolbox for Java に組み込まれています。 IBM Toolbox for Java 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 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。CommandPrompter の使用方法を示すプログラミングの例についての詳細は、以下のページを参照してください。

767 ページの『例: CommandPrompter を使用する』

RunJavaApplication

RunJavaApplication クラスと VRunJavaApplication クラスは、IBM i 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パラメーターを指定してから、このプログラムを開始します。いったん開始したら、入力を Java プログラムに送ることができ、プログラムは標準入力を介してこれを受け取ります。 Java プログラムは、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。

RunJavaApplication Javadoc

VRunJavaApplication Javadoc

67 ページの『JavaApplicationCall クラス』
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。

284 ページの『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。

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 Javadocc 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 = 照会するシステム
- [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 *i_Service* **(-s** *i_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
```

関連情報

JPing Javadoc

AS400JPing Javadoc

Vaccess クラス

Vaccess パッケージとそのクラスは、推奨できなくなりました。代わりに、Access パッケージを Java Swing と組み合わせて使用することをお勧めします。

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 クラスを使用する Java プログラムでは、Java 2 Platform Standard Edition (J2SE) に付属している Swing パッケージが必要です。 Swing についての詳細は、Sun Java Foundation Classes  Web サイト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。

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、システム・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。

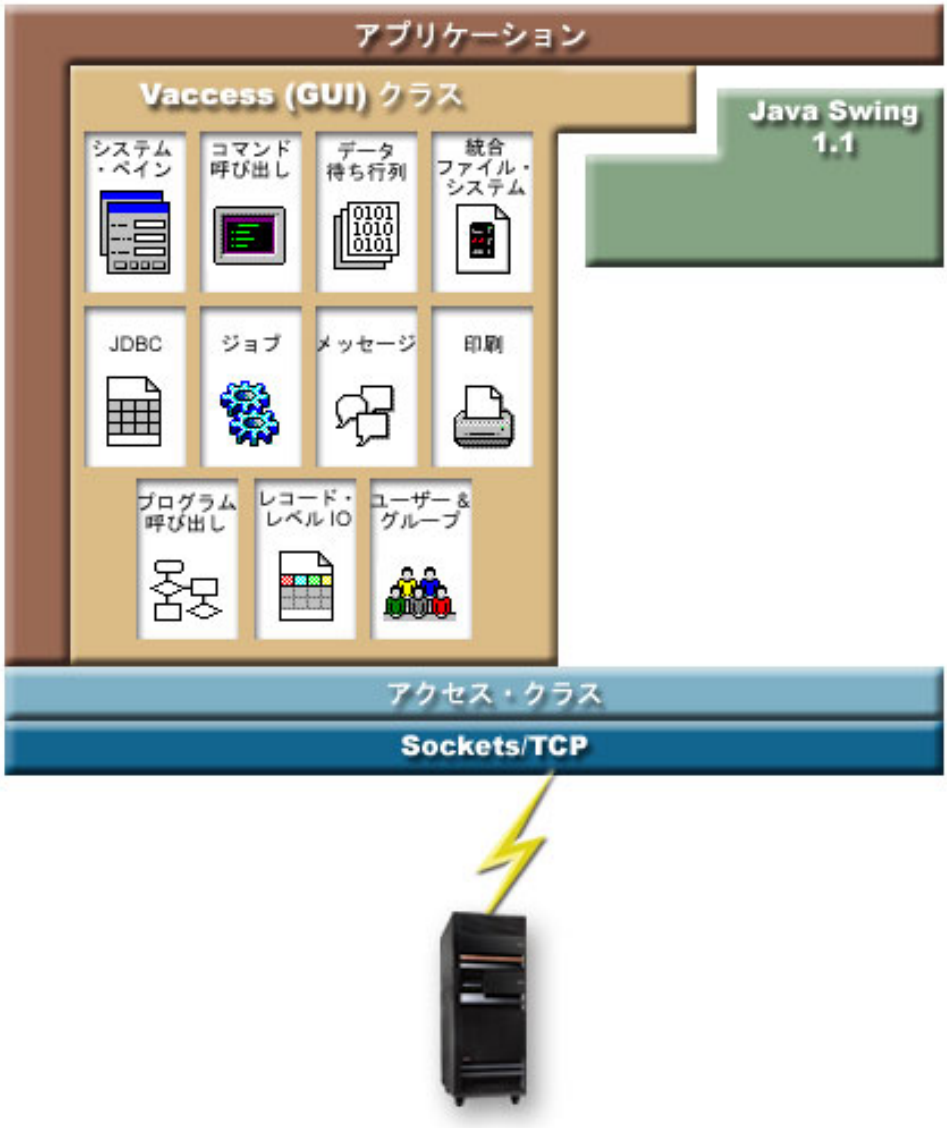
システム・データへのアクセスに関する詳細は、アクセス・クラスを参照してください。

Vaccess クラス

Vaccess パッケージとそのクラスは、推奨できなくなりました。代わりに、Access パッケージを Java Swing と組み合わせて使用することをお勧めします。

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



『図 1 の詳細説明: Vaccess クラス (rzahh508.gif)』

図 1 の詳細説明: Vaccess クラス (rzahh508.gif):

『IBM Toolbox for Java: Vaccess パッケージ』にある図

この図は、vaccess パッケージ、access パッケージ、および Java Swing クラスにあるクラス間の関係を示しています。

説明

この図は以下のもので構成されています。

- 最上部のイメージは、実際にはこげ茶色の枠であり (長方形の左辺と上辺のような形)、 Java アプリケーションを表す「アプリケーション」というラベルが付いています。 Java アプリケーションの枠が付いた長方形には、以下のような不規則な形状のイメージを含んでおり、それらは互いにパズルのピースのように組み合っています。
- IBM Toolbox for Java の Vaccess (GUI) クラスを表す黄褐色の多角形。この形状には、Vaccess クラスに含まれているファンクションを表す小さいイメージが含まれています。
- Java Swing クラスを表す緑の多角形。
- それらのイメージの下には、アクセス・クラスというラベルが付いた水色のバーがあり、それは IBM Toolbox for Java の access パッケージにあるクラスを表します。
- 水色のバーの下には、全く同様の形状をした濃い青のバーがあり、「ソケット/TCP」というラベルが付いています。
- 最下部のイメージは、IBM i サーバーの絵です。
- Java アプリケーションからサーバーへのソケット接続を表す稲妻の光は、 Sockets/TCP (濃い青のバー) から下に伸びており、サーバー (IBM i サーバーのイメージ) を結んでいます。

Java アプリケーション (こげ茶色の境界線の枠が付いているエリア) は、 Vaccess クラス (黄褐色の多角形) および Java Swing クラス (緑の多角形) を含んでいます。 Vaccess クラスは、Java アプリケーションがサーバー (黄褐色の多角形内の小さいイメージ) 上の次のデータおよびファンクションにアクセスすることを可能にします。

AS400Panes、CommandCall、DataQueues、統合ファイル・システム、JDBC、ジョブ、メッセージ、印刷、ProgramCall、レコード・レベルの入出力、およびユーザーおよびグループ

Java アプリケーションは、 IBM Toolbox for Java のアクセス・クラス (水色のバー) を使用して、1 つ以上のソケット接続 (濃い青のバー) を作成します。ソケット接続を使用すると、 Java アプリケーションがサーバー (最下部の IBM i サーバーのイメージ) と通信する (稲妻の光) ことができます。

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、ジョブおよびジョブのリストを表します
- VMessageList および VMessageQueue は、 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、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、IBM 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。

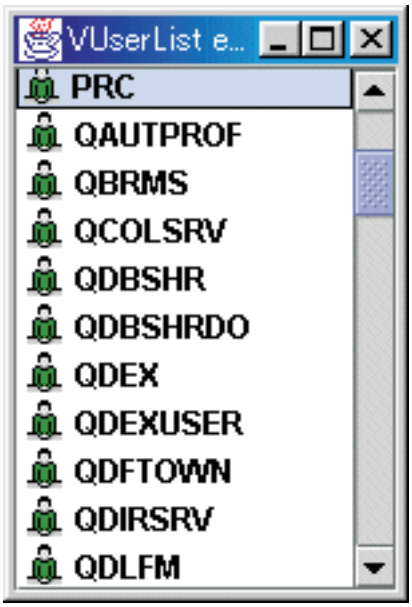
以下のモデルを使用できます。

- 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

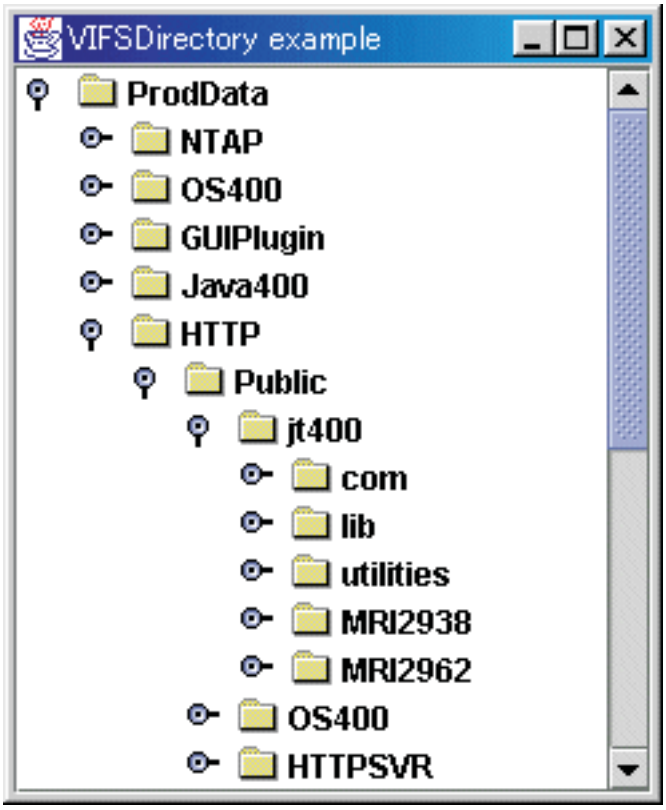


- AS400DetailsPane を VMessageList オブジェクトと一緒に使用することにより、コマンド呼び出しが生成するメッセージのリストを表します。図 2 は、完成後のプロダクトを示します。

図 2: AS400DetailsPane を VMessageList オブジェクトと共に使用する

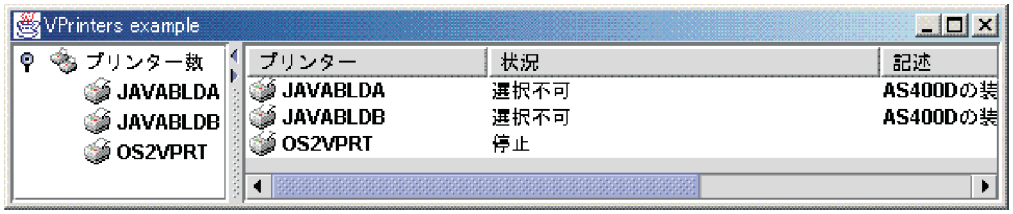
- AS400TreePane を VIFSDirectory オブジェクトと一緒に使用することにより、統合ファイル・システムのディレクトリー階層を表します。図 3 は、完成後のプロダクトを示します。

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



AS400DetailsPane Javadoc

AS400ExplorerPane Javadoc

AS400JDBCDataSourcePane Javadoc

AS400ListPane Javadoc

AS400TreePane Javadoc

VNode Javadoc

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。

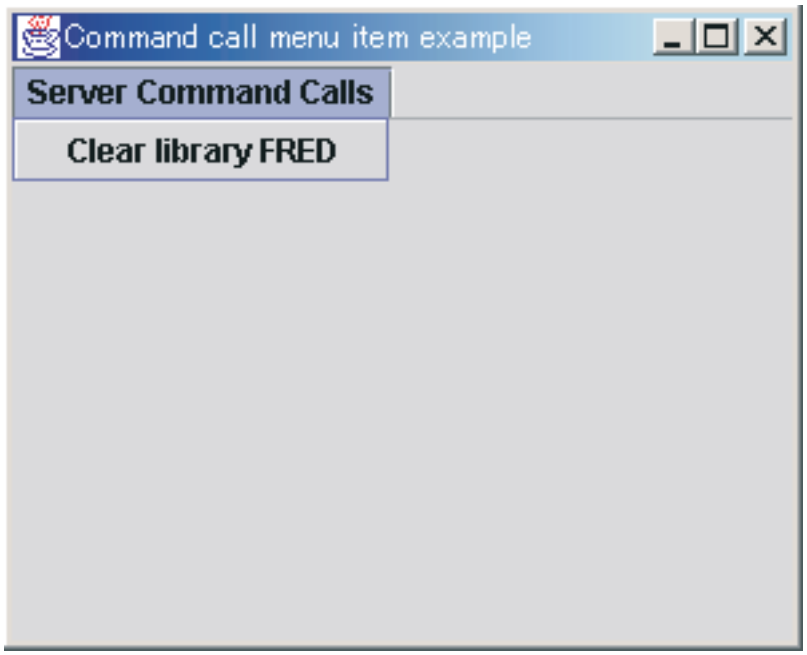
```
// Add an ActionCompletedListener that
// is implemented using an anonymous
// inner class. This is a convenient
// way to specify simple event
// listeners.
 button.addActionCompletedListener (new ActionCompletedListener ()
 {
      public void actionCompleted (ActionCompletedEvent event)
      {
           // 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

関連情報

DataQueueDocument Javadoc

KeyedDataQueueDocument Javadoc

エラー・イベント

多くの場合、IBM Toolbox for Java GUI コンポーネントは、例外をスローする代わりに、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スローされた例外を包むラッパーのことです。

特定のグラフィカル・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。例外がスロー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);
```

例: エラー・リスナーを定義する

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。

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、次のカスタマイズされた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);
```

関連資料

268 ページの『Vaccess クラス』
Vaccess パッケージとそのクラスは、推奨できなくなりました。代わりに、Access パッケージを Java Swing と組み合わせて使用することをお勧めします。

54 ページの『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。

関連情報

ErrorDialogAdapter Javadoc

IFS 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

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。

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。

以下のコンポーネントを使用できます。

"/QSYS.LIB" 以外のファイルへのパスを設定してください。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。

IFSFileSystemView:

このクラスは推奨できなくなり、com.ibm.as400.access.IFSSystemView で置き換えられました。

IFSFileSystemView は、javax.swing.JFileChooser オブジェクトの構成時に使用する、IBM i 統合ファイル・システムへのゲートウェイを提供します。

JFileChooser は、ファイルのナビゲートや選択を行うダイアログを構築するための Java の標準的な方法で、 IFSFileDialog に代えて使用することが推奨されています。

例: IFSFileSystemView を使用する

以下の例は、IFSFileSystemView の使用法を示します。

```
import com.ibm.as400.access.AS400;
import com.ibm.as400.access.IFSJavaFile;
import com.ibm.as400.vaccess.IFSFileSystemView;
import javax.swing.JFileChooser;
import java.awt.Frame;

// Work with directory /Dir on the system myAS400.
AS400 system = new AS400("myAS400");
IFSJavaFile dir = new IFSJavaFile(system, "/Dir");
JFileChooser chooser = new JFileChooser(dir, new IFSFileSystemView(system));
Frame parent = new Frame();
int returnVal = chooser.showOpenDialog(parent);
if (returnVal == JFileChooser.APPROVE_OPTION) {
   IFSJavaFile chosenFile = (IFSJavaFile)(chooser.getSelectedFile());
   System.out.println("You selected the file named " +
                      chosenFile.getName());
}
```

関連情報

IFSFileSystemView Javadoc

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);
```

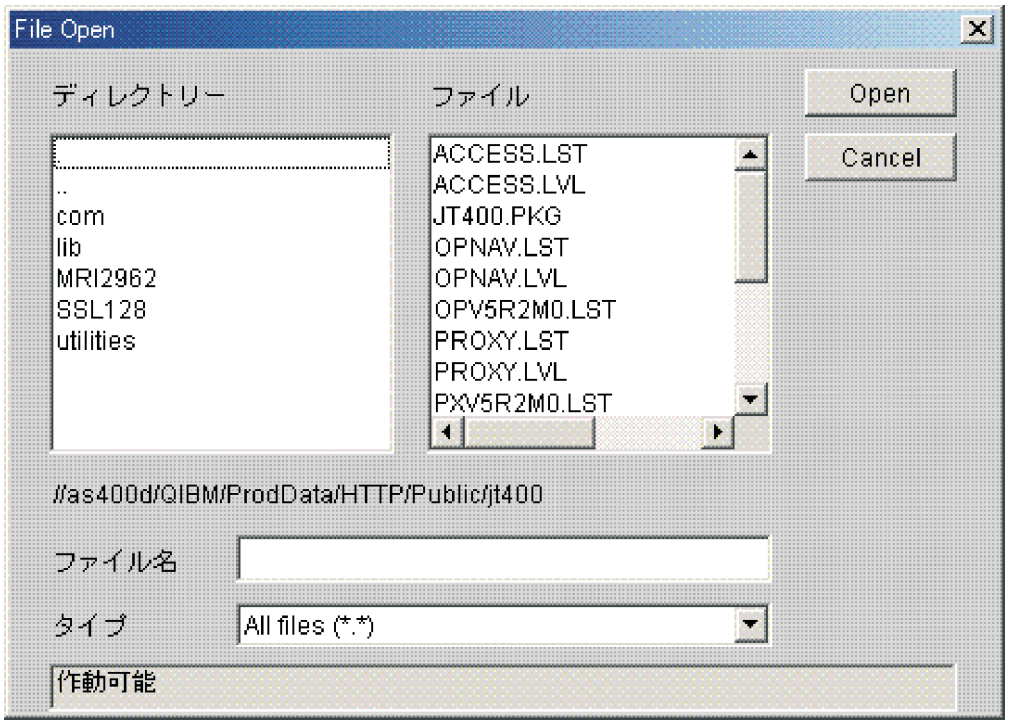
```
// Set a list of filters for the dialog.
// The first filter will be used
// when the dialog is first displayed.
FileFilter[] filterList = {new FileFilter ("All files (*.*)", "*.*"),
                           new FileFilter ("HTML files (*.HTML", "*.HTM")};

// 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



IFSFileDialog Javadoc

FileFilter Javadoc

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。

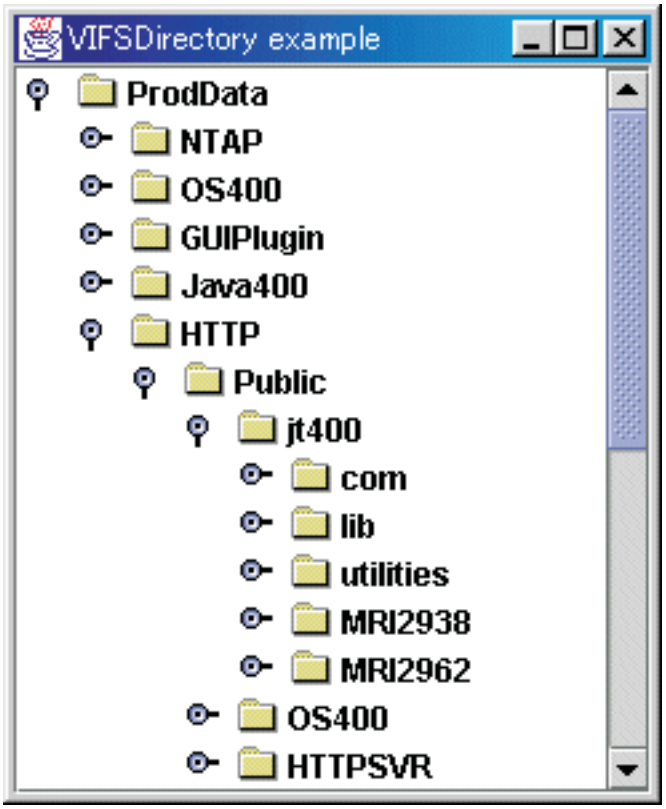
```
// Create the VIFSDirectory object.
// Assume that "system" in an AS400
// object created and initialized
// elsewhere.
VIFSDirectory root = new VIFSDirectory (system, "/DirectoryA/DirectoryB");

// 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



270 ページの『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。

VIFSDirectory Javadoc

IFSFileFilter Javadoc

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。

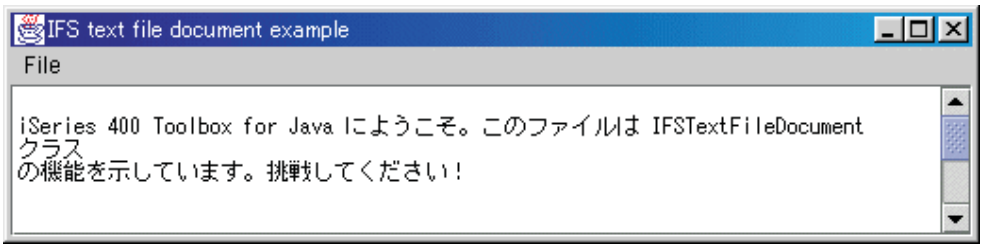
```
// Create and load the
// IFSTextFileDocument object. Assume
// that "system" is an AS400 object
// created and initialized elsewhere.
IFSTextFileDocument ifsDocument = new IFSTextFileDocument (system, "/DirectoryA/MyFile.txt");
ifsDocument.load ();

// 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: IFSTextFileDocument の例



IFSTextFileDocument Javadoc

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。

関連情報

VJavaApplicationCall Javadoc

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、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、IBM 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。

SQLConnection Javadoc

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。

SQLStatementButton Javadoc

SQLStatementMenuItem Javadoc

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() メソッドを使用すれば、文書内に入っている 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。

関連情報

SQLStatementDocument Javadoc

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);
```

関連情報

SQLResultSetFormPane Javadoc

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
```

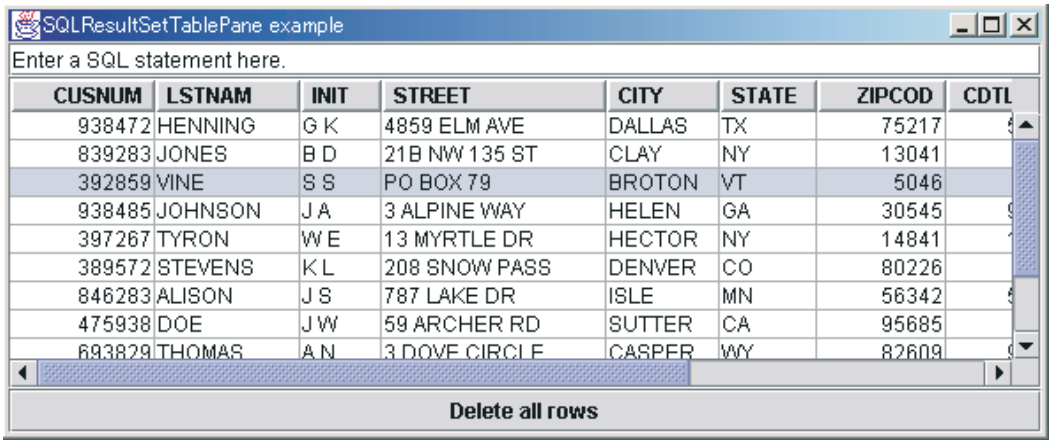
```
// created and initialized elsewhere.
SQLResultSetTablePane tablePane = new SQLResultSetTablePane (connection,
                                  "SELECT * FROM QIWS.QCUSTCDT");
// Load the results.
tablePane.load ();

// 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



関連情報

SQLResultSetTablePane Javadoc

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);
```

関連情報

SQLResultSetTableModel Javadoc

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。

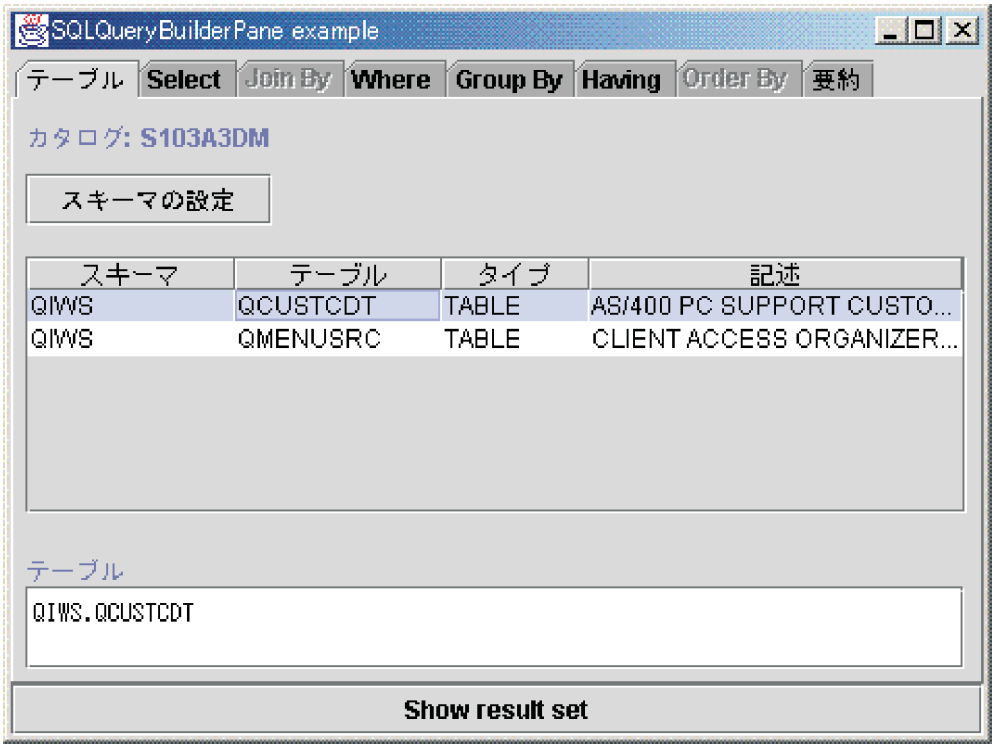
```
// Create an SQLQueryBuilderPane
// object. Assume that "connection"
// is an SQLConnection object that is
// created and initialized elsewhere.
SQLQueryBuilderPane queryBuilder = new SQLQueryBuilderPane (connection);

// 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



関連情報

SQLQueryBuilderPane Javadoc

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。

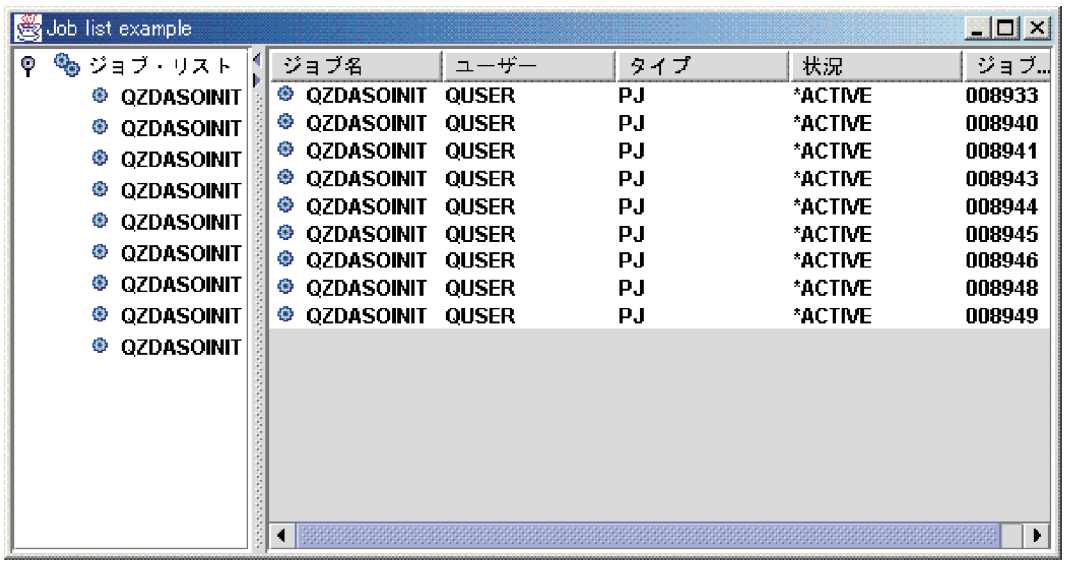
以下の例では、 VJobList を作成し、それを AS400ExplorerPane に表します。

```
// 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。



VJobList Javadoc

270 ページの『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。

VJob Javadoc

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、 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。

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

関連資料

270 ページの『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。

関連情報

VMessageList Javadoc

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() メソッドを使用したりします。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** - 重大度、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。

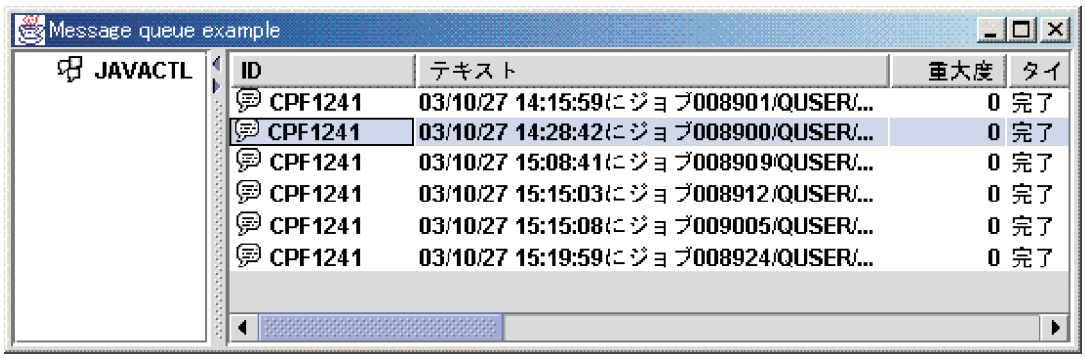
以下の例では、VMessageQueue を作成し、それを AS400ExplorerPane に表します。

```
                    // 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



関連情報

VMessageQueue Javadoc

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

ここにリストされている vaccess パッケージ・コンポーネントを使用すれば、Java プログラムは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・インターフェースで示すことができます。

- 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。

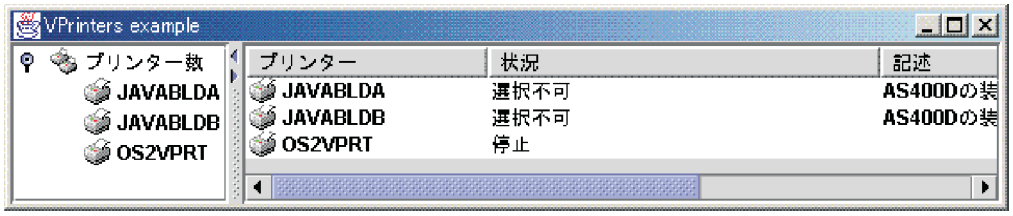
以下の例では、VPrinters オブジェクトを作成し、それを AS400TreePane に表示します。

```
                    // 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



関連情報

VPrinters Javadoc

VPrinter クラス:

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。

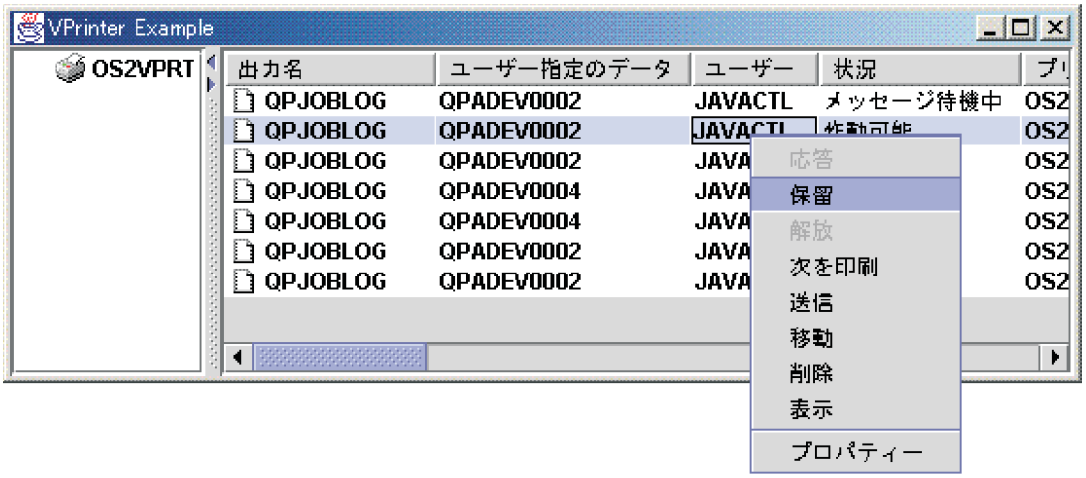
```
                    // Create the VPrinter object.
                    // Assume that "system" is an AS400
                    // object created and initialized
                    // elsewhere.
VPrinter root = new VPrinter (new Printer (system, "MYPRINTER"));

                    // 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



関連情報

VPrinter Javadoc

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。

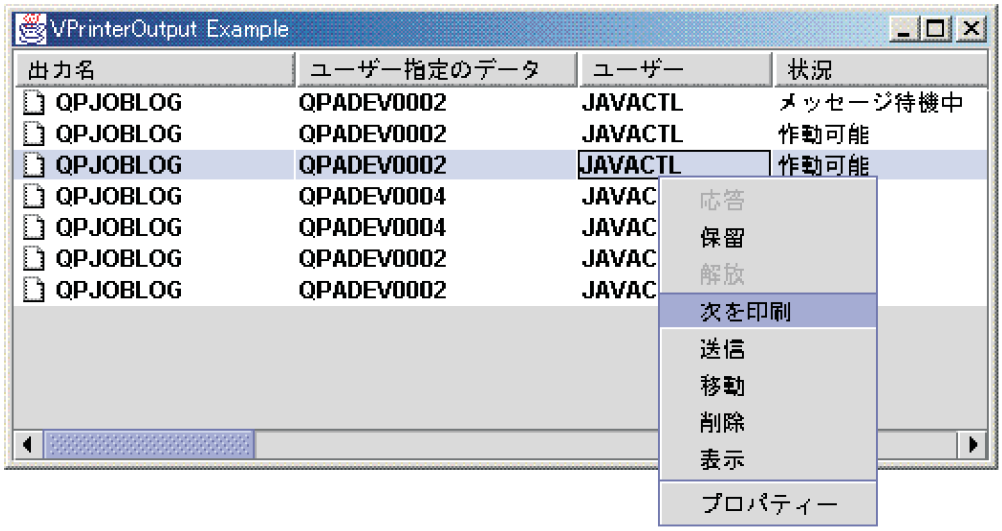
以下の例では、VPrinterOutput を作成し、それを AS400ListPane に表示します。

```
                    // 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



関連情報

VPrinterOutput Javadoc

SpooledFileViewer クラス:

IBM Toolbox for Java 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。

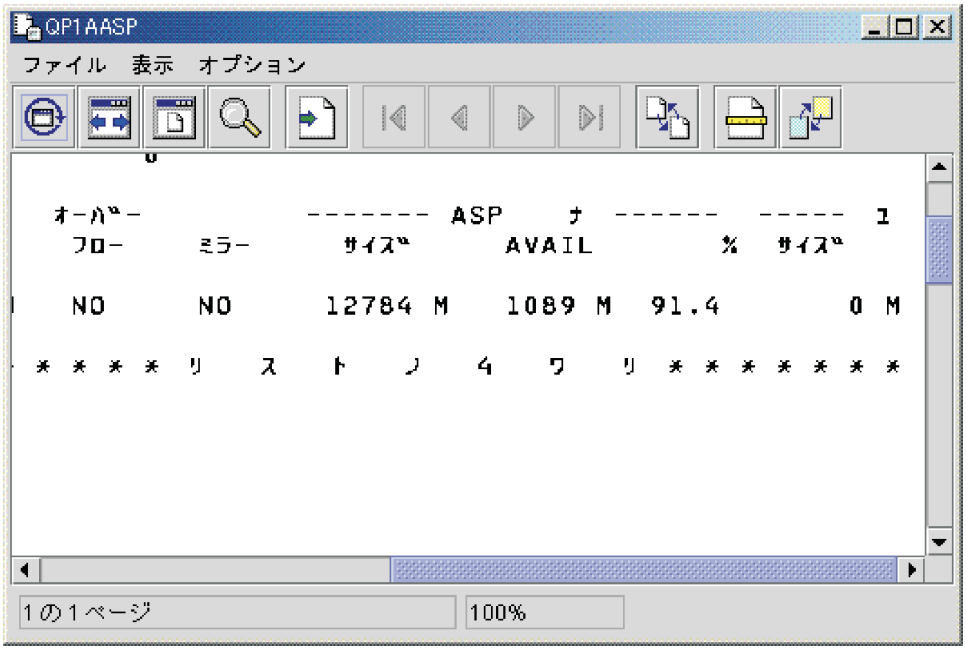
SpooledFileViewer の使用

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() 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。非活動化される際に最後のページが表示されることを示します。


「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。

SpooledFileViewer Javadoc

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パラメーター・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。

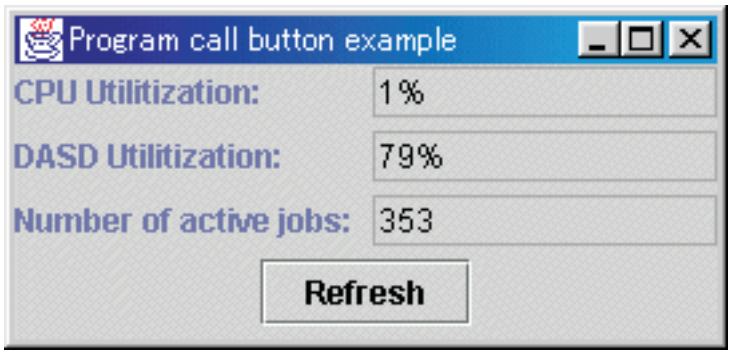
以下の例では、 2 つのパラメーターを追加します。

```
                    // 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



ProgramParameter Javadoc

ProgramCallButton Javadoc

ProgramCallMenuItem Javadoc

ActionCompletedListener Javadoc

ActionCompletedEvent Javadoc

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か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か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、キーよりも小さいか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。

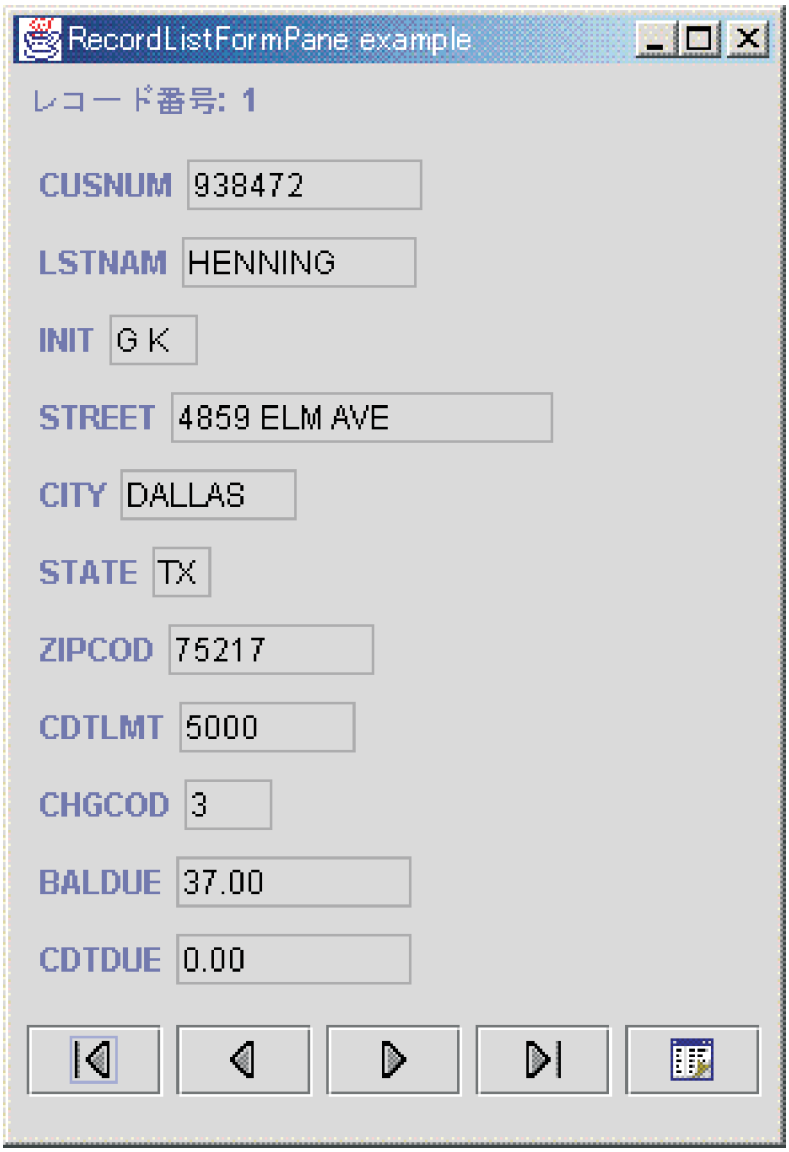
以下の例では、RecordListFormPane オブジェクトを作成し、それをフレームに追加します。

```
                    // 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



関連情報

RecordListFormPane Javadoc

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);
```

関連情報

RecordListTablePane Javadoc

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);
```

RecordListTableModel Javadoc

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、例外がスローされるのではなく、 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 server.
detailsPane.load();
```

ResourceListPane Javadoc

ResourceListDetailsPane Javadoc

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。

VSystemStatus Javadoc

VSystemStatusPane Javadoc

VSystemPool クラス:

VSystemPool クラスを使用すれば、 GUI 設計を使用して、サーバーからシステム・プール情報を検索し、設定することができます。 VSystemPool は、 VSystemStatusPane を含む、 vaccess パッケージ内のさまざまなペインとともに動作します。

以下のリストは、VSystemPool で使用できるいくつかのメソッドです。

- getActions() メソッドは、実行できるアクションのリストを戻します。
- getSystem() メソッドは、システム・プール情報が検出されるサーバーを戻します。
- setSystemPool() メソッドは、システム・プール・オブジェクトを設定します。

関連情報

VSystemPool Javadoc

VSystemStatusPane

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();
```

VSystemStatusPane Javadoc

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 を使用するか、または Java Foundation Classes (JFC) を使用して独自のペインを作成して、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、または setSystem() メソッドを使用してシステム名を設定します。

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();
```

関連情報

VSystemValueList Javadoc

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する必要があります。

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);
```

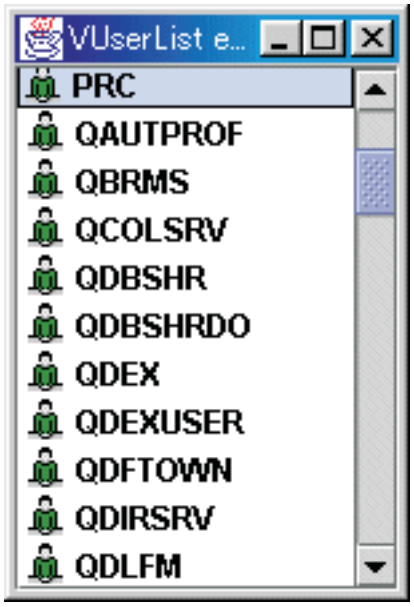
以下の例は、VUserAndGroup オブジェクトを使用する方法を示しています。

```
// 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。



VUser Javadoc

VUserAndGroup Javadoc

Graphical Toolbox および PDML

Graphical Toolbox は、Java 形式のカスタム・ユーザー・インターフェース・パネルの作成を可能にする 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、または System i Navigatorのプラグインにそのパネルを取り込むことができます。システムや他のソース (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入れることができます。

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レンダリングを JavaFoundation Classes を使って行うことにより、パネルを表示します。

注: PDML を使用するには、 Java ランタイム環境のバージョン 1.4 以降を実行する必要があります。

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 によるスケルトンを生成できます。必要なヘルプ・トピックを指定すると、必要に応じたそのヘルプ・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・タグがヘルプのスケルトン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・コードのスケルトン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

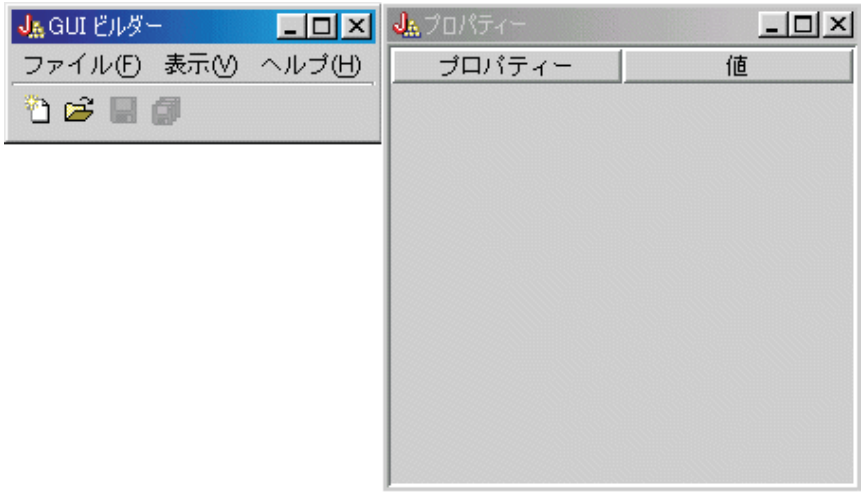
Graphical Toolbox の実行時環境では、イベント処理、ユーザー・データ妥当性検査、およびパネルの要素間の共通タイプの対話がサポートされます。プラットフォーム上での正しいユーザー・インターフェースの外観は、その使用しているオペレーティング・システムに基づいて自動的に設定されます。 GUI ビルダーによって外観をいろいろと切り替えれば、さまざまなプラットフォーム上でパネルが表示される様子を評価できます。

Graphical Toolbox 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、最初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

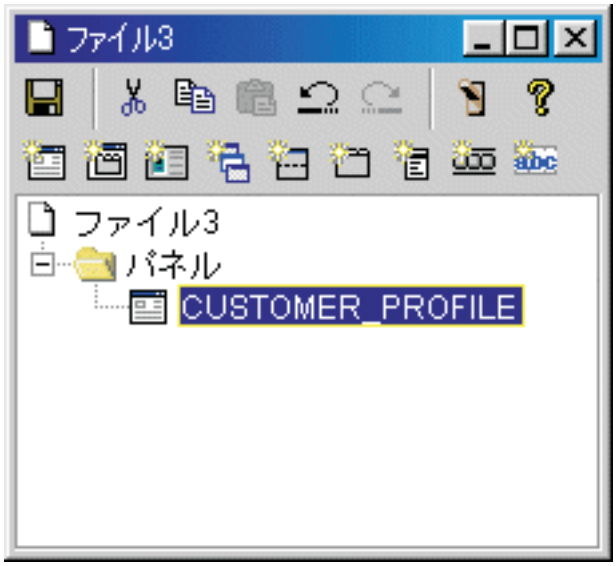
初めて GUI ビルダーを起動する際、図 1 の 2 つのウィンドウが表示されます。

図 1: GUI ビルダーのウィンドウ



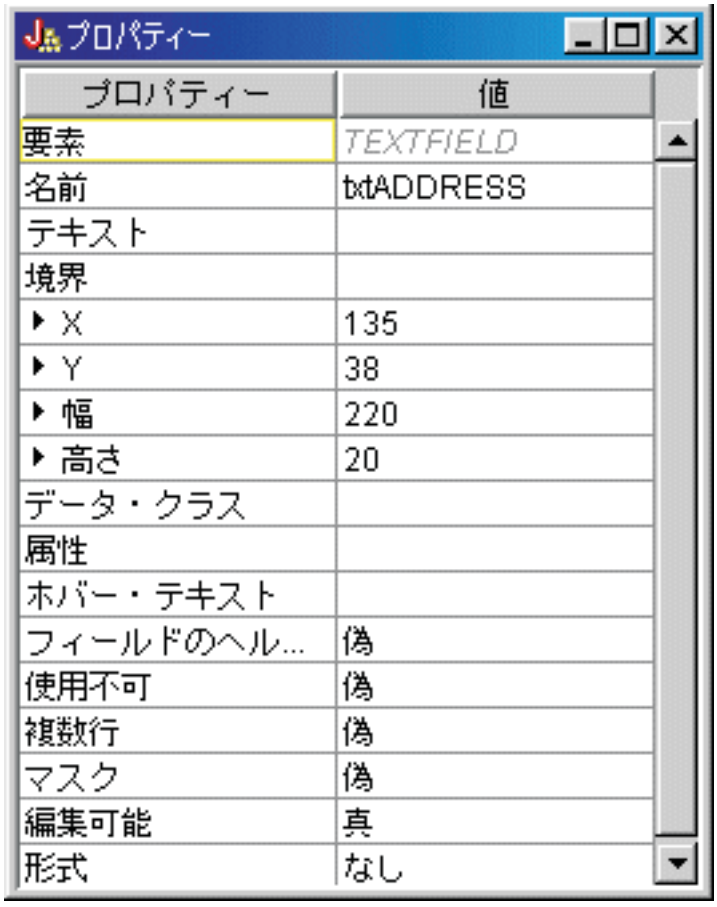
「ファイル・ビルダー」ウィンドウは、PDML ファイルを作成および編集するために使用します。

図 2: 「ファイル・ビルダー」ウィンドウ



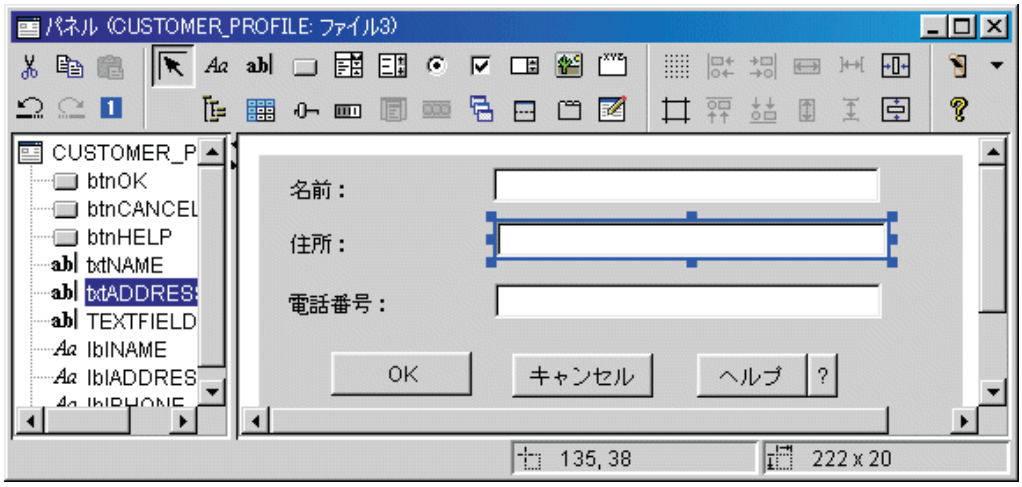
「プロパティー」ウィンドウを使用して、現在選択されているコントロールのプロパティーを表示したり変更を加えたりすることができます。

図 3: 「プロパティー」ウィンドウ



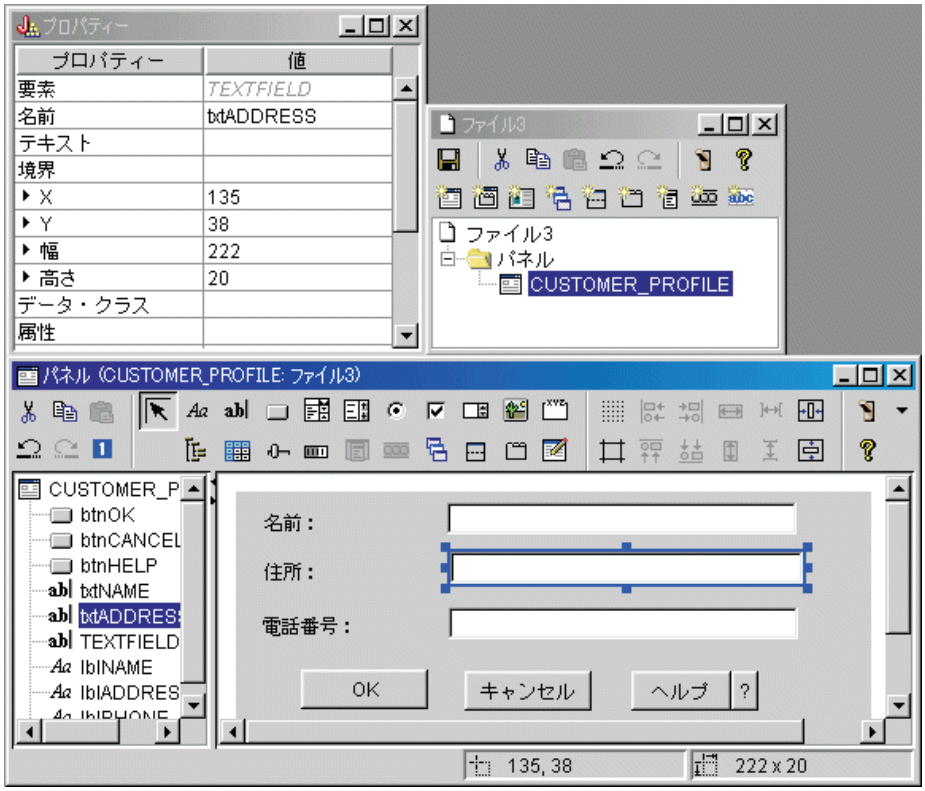
「パネル・ビルダー」ウィンドウを使用して、グラフィカル・ユーザー・インターフェースのコンポーネントを編集できます。ツールバーから希望のコンポーネントを選択して、パネルをクリックして任意の場所にそれを配置します。またツールバーには、コントロールのグループを位置合わせしたり、パネルをプレビューしたり、GUI ビルダー機能のオンライン・ヘルプを要求したりする機能も備えられています。それぞれのアイコンで行えることについての説明は、GUI ビルダーのパネル・ビルダー・ツールバーを参照してください。

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。

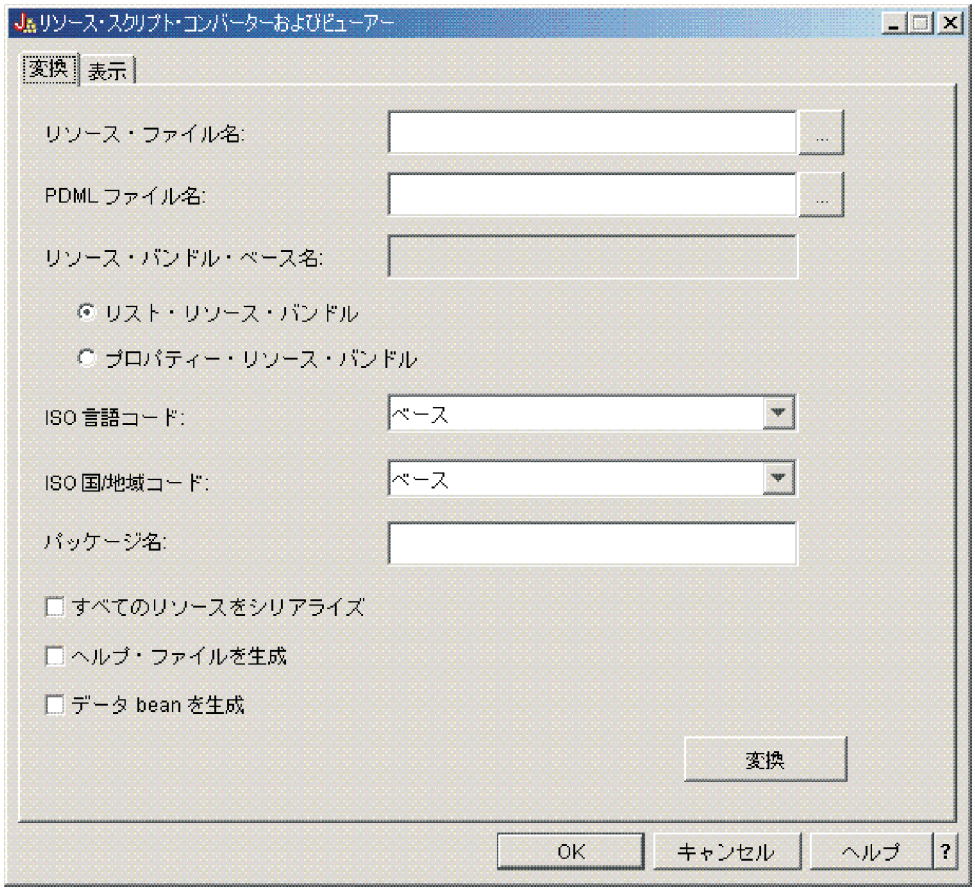
図 5: GUI ビルダーの各ウィンドウが協働する方法の例



リソース・スクリプト・コンバーター

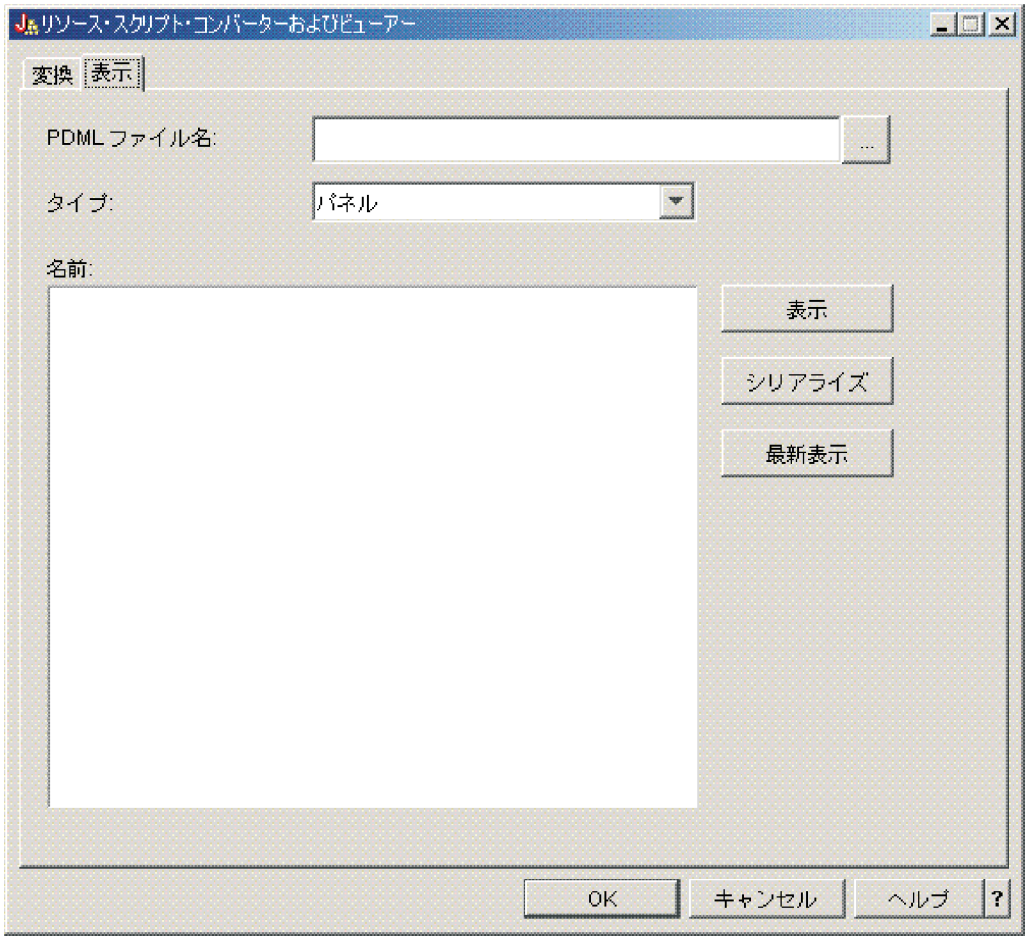
リソース・スクリプト・コンバーターは、 2 つのペインから成るタブ形式のダイアログです。「**変換**」 ペインには、 PDML に変換する Microsoft または VisualAge® for Windows の RC ファイルの名前を指定します。目的の PDML ファイル、および関連する Java リソース・バンドル (パネルの変換済みストリングを含む) に名前を指定することができます。また、パネルのオンライン・ヘルプのスケルトンを生成するよう要求したり、パネルのデータがあるオブジェクトの Java ソース・コードのスケルトンを生成したり、パネル定義を直列化して実行時間のパフォーマンス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。システム上に 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 i 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 i Access for Windows を使用した JAR ファイルのインストール

IBM i 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 IBM i Access for Windows の一部として出荷されています。初めて IBM i 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。IBM i Access for Windows をインストール済みの場合に、このコンポーネントをまだインストールしていなければ、選択セットアップ・プログラムを使って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 `classpath` 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¥jhall.jar;
```

CLASSPATH には XML パーサーも追加する必要があります。詳細は、以下のページを参照してください。

454 ページの『XML パーサーおよび XSLT プロセッサー』

IBM i Access for Windows を使用して Graphical Toolbox をインストールした場合には、IBM i Access for Windows のインストール先のドライブの **¥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。 IBM i 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は、これを反映します。

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 IBM i プログラムを呼び出します。 これらのクラスは、アプリケーションとともに再配布できます。
- **util400.jar**: IBM i データのフォーマットおよび IBM i 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。
- **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。
- **XML parser**: 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。

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、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 IBM i 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、検索順に追加されます。

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・ツールを使用して、ユーザー・インターフェースを作成します。

GUI ビルダーを開始するには、以下のコマンドを使用してください。

```
java com.ibm.as400.ui.tools.GUIBuilder [-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*

任意の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。これは UI 要素が配置されている長方形の領域を説明します。 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、イメージ、アニメーションなど) で構成することもできますし、カスタム・コントロールや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 (例えば、「切り取り」、「コピー」、および「貼り付け」)、 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、区切り記号およびカスケード・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ラックされます。

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可能性があります。

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 のスケルトンを生成できます。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) に挿入しなければなりません。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・コードのスケルトンを生成できます。 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・コードのスケルトンを生成し、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 ファイルの内容を直列化できます。直列化すると、ファイル中の UI リソースすべての短縮バイナリーが作成されます。直列化すると、パネルを表示する際に PDML ファイルは解釈されてはならないため、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。ロケールを指定すると、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が接尾部として、生成されるリソース・バンドルの名前に付けられます。

-h オンライン・ヘルプの HTML のスケルトンを生成します。

-d JavaBeans のソース・コードのスケルトンを生成します。

-s リソースをすべて直列化します。

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 ファイルにも反映される 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・コードのスケルトン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する 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する 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レンダリングを JavaFoundation Classes を使って行うことにより、パネルを表示します。

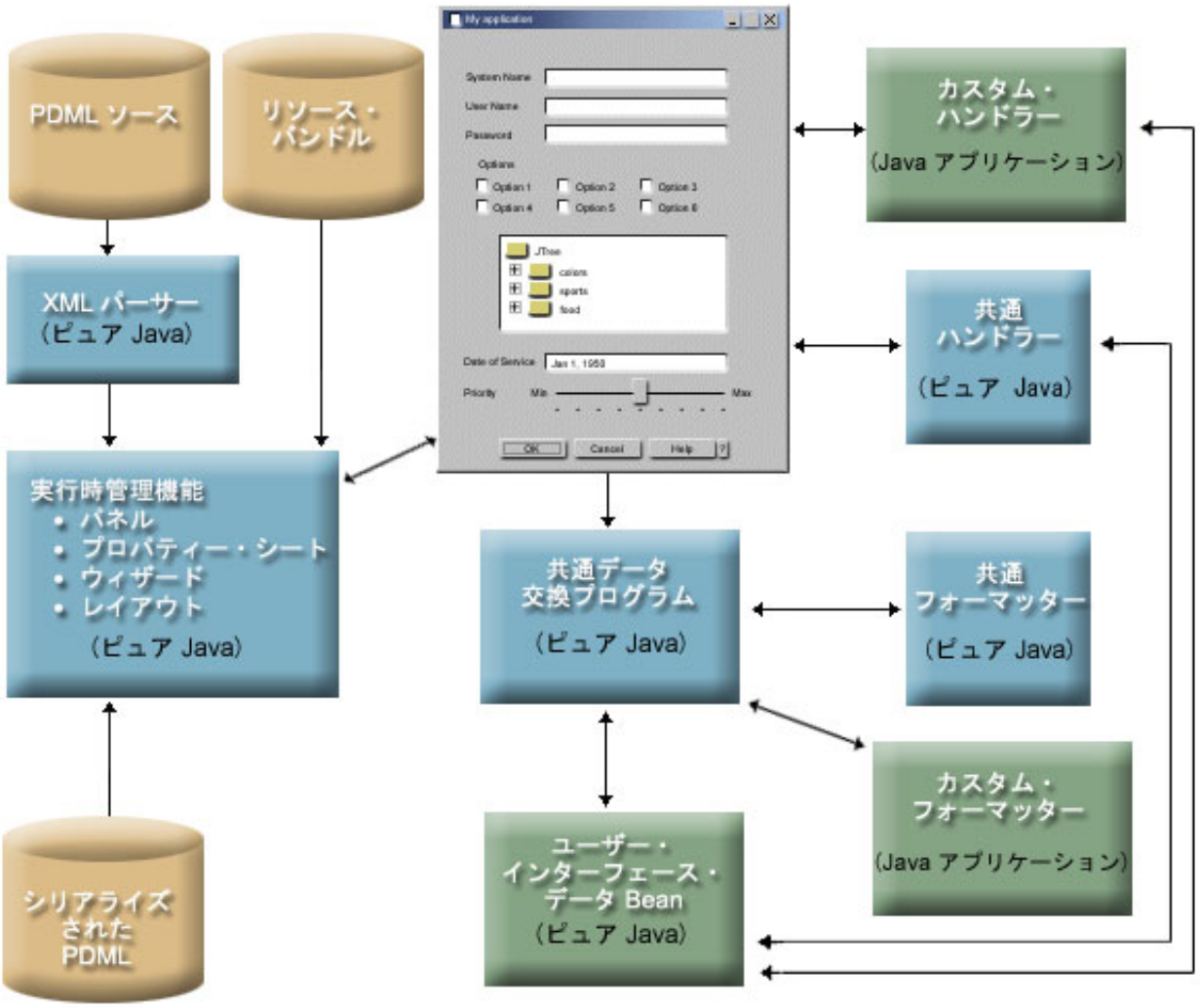
Graphical Toolbox の実行時環境は、以下のサービスを提供します。

- 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型) 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実装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。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

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。

関連情報

パッケージ com.ibm.as400.ui.framework.java の要約

図 1 の詳細説明: Graphical Toolbox の実行時環境 (rzahh504.gif)

『IBM Toolbox for Java: 実行時のパネルの表示』にある図

この図は、Graphical Toolbox の実行時環境の各要素がアプリケーション・コードと対話する方法を図解しています。

説明

この図は異なった形状、サイズ、および色からなるいくつかのボックスで構成されており、一方あるいは両側を指す矢印が末端にある線で互いに結ばれています。

図を視覚的に捕らえるために、便宜上、それを 3 つの列と 4 つの行に分割し、そのエリアに左上から右下へと順に番号を振ります。たとえば、1 行目には、エリア 1、2、および 3 があります。 2 行目には、エリア 4、5、および 6 があります。以下も同様です。

- エリア 2 と 5 にまたがるダイアログ・ボックスのイメージは、 Java プログラムの GUI インターフェースを表しています。そのダイアログ・ボックスは、チェック・ボックス、テキスト・フィールドなどのさまざまなオプションを扱っています。
- エリア 1 の上部にある 2 つの黄褐色のシリンダーは、 PDML ソースおよびリソース・バンドルというラベルが付いています。これらのシリンダーは、ストレージ・メディアにある PDML ソース・ファイルおよび Java リソース・ファイルを表しています。
- エリア 10 の PDML 直列化済みというラベルが付いた 1 つの黄褐色のシリンダーは、ストレージ・メディアにある 1 つ以上の直列化された PDML ファイルを表します。
- ダイアログ・ボックスの下の部分を取り囲んでいる 5 つの青い長方形は、 Graphical Toolbox のコンポーネントを表しています。左端の長方形から始めて左回りに見ていくと、次のようなラベルが付いています。
  - エリア 4 の XML パーサー (ピュア Java)。これはIBM XML パーサーを表します。
  - エリア 7 の 実行時管理機能 (ピュア Java)。ユーザーの Java プログラムは、実行時管理機能に含まれる 1 つ以上のオブジェクトのクライアントです。そのオブジェクトには、パネル、プロパティー・シート、ウィザード、およびレイアウトがあります。
  - エリア 8 の共通データ交換プログラム (ピュアJava)。
  - エリア 9 の共通フォーマッター (ピュア Java)。

  - エリア 6 の共通ハンドラー (ピュア Java)。
- 3 つの緑の長方形は、アプリケーション・プログラマーが与えたコードで、次のようなラベルが付いています。
  - エリア 3 のカスタム・ハンドラー (Java アプリケーション)。
  - エリア 12 のカスタム・フォーマッター (Java アプリケーション)。
  - エリア 11 のユーザー・インターフェース・データ beans (ピュア Java)。
- 線が多くの形状を結んでいます。
  - 単一の矢印を (片側に) 持つ線は、アクションを表します。単一矢印の線は、線の出発点にあるオブジェクトを使用するファンクションあるいはコンポーネントの方を指しています。以下の説明では、「使用する」という言葉は、あるオブジェクトによりアクションを行うコンポーネントを出発点とする単一矢印を持った線が、そのオブジェクトの方を指している、という意味です。
  - 2 つの矢印を (両側に) 持つ線は、対話を表しています。これらの線は情報の両方向交換を行うオブジェクトを結んでいます。以下の説明では、「対話する」という言葉は、コンポーネントが 2 つの矢印を持った線で結ばれている、という意味です。

Java プログラム の GUI インターフェース (エリア 2 と 5 にあるダイアログのイメージ) は、 Graphical Toolbox の実行時管理機能 (エリア 7 の青い長方形) と対話します。

ピュア Java である実行時管理機能には、パネル、プロパティー・シート、ウィザード、および GUI レイアウトがあります。 GUI を生成するために、実行時管理機能は、 Java リソース・バンドル (エリア 1 にある 2 つの黄褐色のシリンダーの内の 1 つ) および PDML データを使用します。実行時管理機能は、次のどちらかの方法により PDML データを処理することができます。

- 直列化された PDML ファイル (エリア 10 の黄褐色のシリンダー) を使用する。
- IBM XML パーサー (エリア 4 の青い長方形) を使用する。 XML パーサーは次に、PDML ソース・ファイル (エリア 1 にある 2 つのシリンダーの内の 1 つ) を使用します。

GUI を有効にした Java プログラムは、次のどちらかによりデータを操作します。

- GUI インターフェースがカスタム・ハンドラー (エリア 3 の緑の長方形) および共通ハンドラー (エリア 6 の青い長方形) と対話するようにする
- 共通データ交換プログラム (エリア 8 の青い長方形) が GUI インターフェースを使用して情報を取得するようにする

カスタム・ハンドラー、共通ハンドラー、および共通データ交換プログラムはすべて、ユーザー・インターフェース・データ beans (エリア 11 の緑の長方形) と対話して、情報をやり取りします。共通データ交換プログラムは、共通フォーマッター (エリア 9 の青い長方形) およびカスタム・フォーマッター (エリア 12 の緑の長方形) と対話して、データをユーザー・インターフェース・データ beans に合った様式に変換します。

GUI ビルダーによって生成されたヘルプ文書の編集

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スケルトンを生成してから、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 -->
```

および

```
<!-- 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の程度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。

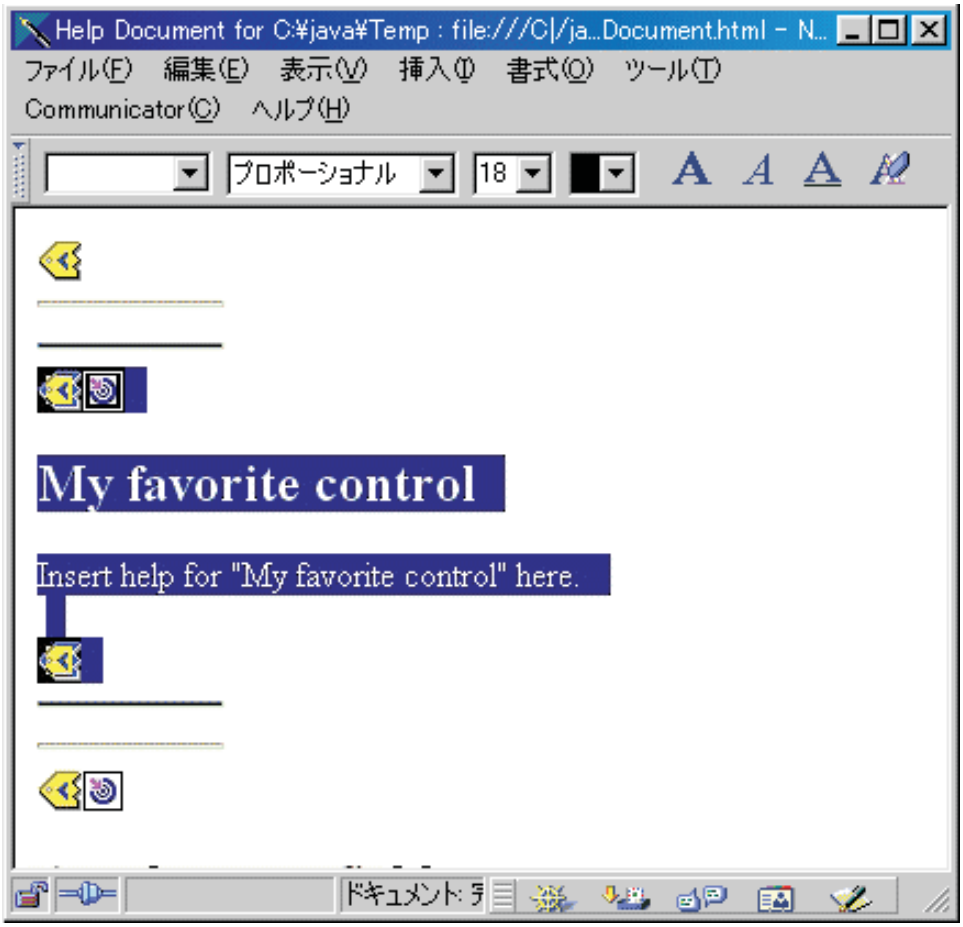
注:

- 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

ほとんどすべてのビジュアル HTML エディター内でヘルプの内容を編集できます。 HELPDOC タグはコメントであるため、一部のエディターでは表示されない場合があります。便宜上、ヘルプのスケルトンの SEGMENTBEGIN タグの直前および SEGMENTEND タグの直後に水平けい線が追加されます。これらの水平けい線は、ビジュアル・エディター内でセグメント全体の表示を明確にします。あるセグメントを移

動、コピー、または削除のために選択する場合、前後の水平けい線を選択すると、SEGMENTBEGIN および SEGMENTEND タグを確実に選択に含めることができます。これらの水平けい線は、完成した個々の HTML ファイルにコピーされません。



追加トピックの作成

ヘルプ文書に追加のトピック・セグメントを作成できます。多くの場合、これを行うには別のセグメントをコピーするのが最も簡単です。そのセグメントをコピーする場合、SEGMENTBEGIN および SEGMENTEND タグの前後にある水平けい線をコピーしてください。これにより、将来のビジュアル編集がより簡単になり、タグのミスマッチを避けるのに役立ちます。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。
- ヘルプのスケルトン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。
- 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。元のヘルプのスケルトン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スケルトン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。

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。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

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、コードがいくつか追加されました。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** メソッドでは、アプレットのペインが正しいサイズに調整され、それからそのパネル・サイズを保持するために、ブラウザー・ウィンドウのサイズ変更時に **paint** メソッドがオーバーライドされます。

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 ランタイム環境を提供するために Sun 社の Java プラグインの使用が勧められていますが、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** が Web サーバーに格納されています。他の XML パーサーも使用できます。詳しくは、454 ページの『XML パーサーおよび XSLT プロセッサー』を参照してください。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*直列化*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

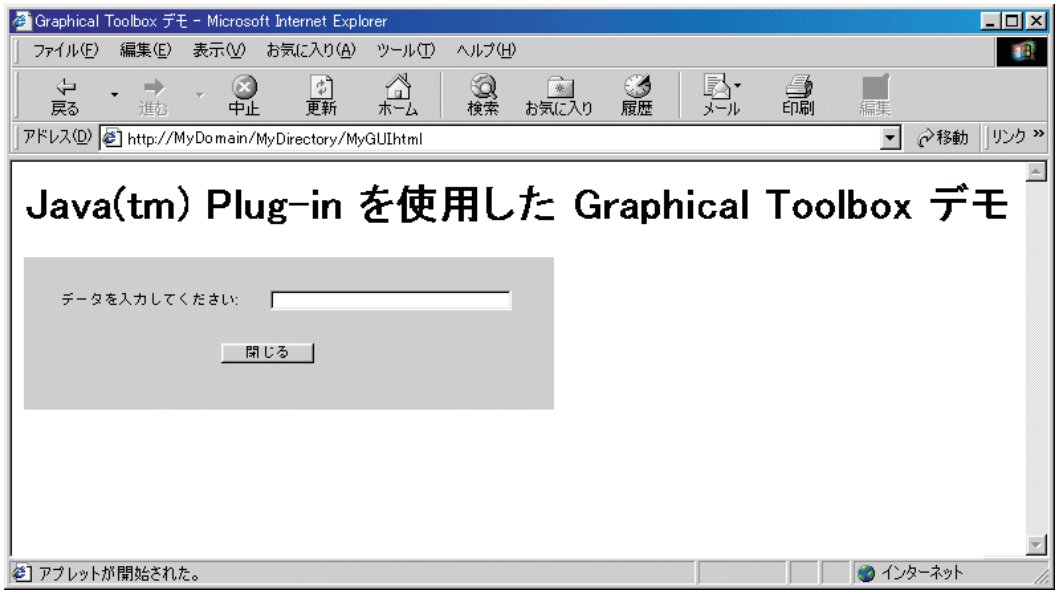
以下のステップを実行して、指定した Web サーバーにアプレットをインストールします。

1. **SampleApplet.java** をコンパイルします。
2. 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。
3. 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。
4. 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。
5. 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 ファイル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。

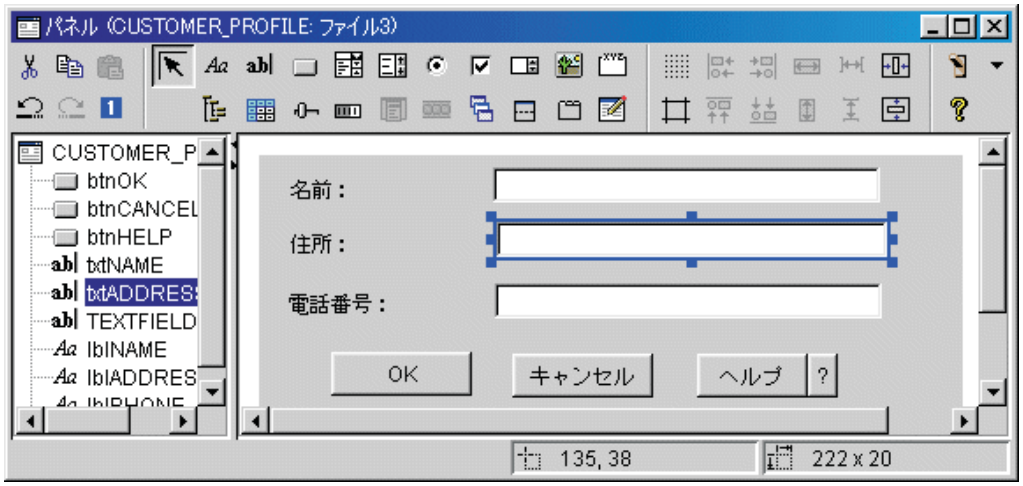
図 1: ブラウザーでのサンプル・アプレットの実行



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・ボックスを挿入できます。

 「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・ボックスを挿入できます。

 「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ァイル間でパネル・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 に関する詳細を表示できます。

IBM Toolbox for Java Bean

JavaBeans 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 のパブリック・クラスの多くは、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

以下の例では、プログラムで JavaBeans を使用する方法、および Visual bean ビルダーを使って JavaBeans からプログラムを作成する方法を示します。

599 ページの『例: IBM Toolbox for Java bean コード』

600 ページの『例: visual bean ビルダーを使用して bean を作成する』

JDBC

JDBC 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 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、IBM 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・ファイルが同じサーバー上にある場合、ネイティブ JDBC ドライバーを使用します。

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 および 5.0 に組み込まれています。
- JDBC 4.0 API は、Java SE バージョン 6 に組み込まれています。

IBM Toolbox for Java JDBC support for IBM i 7.1 に対する機能拡張

IBM i 7.1 では、JDBC サポートに対して多数の追加が行われました。

IBM Toolbox for Java JDBC サポートの拡張については、この後の項で詳述します。

- 『XML データ型のサポート』
- 344 ページの『データベース・メタデータの更新』
- 345 ページの『「現在コミット済み」のサポート』
- 345 ページの『配列型のサポート』
- 346 ページの『長いスキーマ名のサポート』

XML データ型のサポート

JDBC 4.0 インターフェース仕様には、XML データ型をサポートするための新しいメソッドとクラスが追加されています。 IBM Toolbox for Java では JDBC 4.0 ドライバーによって XML サポートが実装されています。これにより、JDBC クライアントは IBM i 7.1 XML サポートを簡単に利用できます。

Java String などのさまざまなデータ型を渡すことにより、JDBC SQLXML オブジェクト内のデータを設定できます。他のすべてのテキスト列データ型に対する操作と同様に、XML 列の列 CCSID にマッチさせるための必要な XML データ変換は IBM Toolbox for Java またはデータベース管理システム (DBMS) によって行われます。

DBMS は SQL ロケーターを使って XML データの読み取り/書き込みを実行します。

DBMS への入力として送られる XML は、正しい CCSID 変換を行うためにいくつかの特殊な規則に従って適切に扱われる必要があります。エンコード方式が指定された XML 宣言を持つ入力 XML データの場合、その特定のエンコード方式に従って XML データが正しくエンコードされている必要があります。例えば、UTF-16 (Java String) でエンコードされ、XML 宣言で UTF-8 エンコード方式が指定されているような XML は、正しくありません。 XML 入力データに宣言が含まれない場合、DBMS はデータ・ストリームが UTF-8 で送られることを想定します。この場合、IBM Toolbox for Java は XML データをまず UTF-8 に変換した後、そのデータを DBMS に送って処理させます。

注: データベース内の XML 列は、XML 宣言なしで保管されます。 XML データが取り出されるときに XML 宣言が動的に生成されます。このため、その XML データは入力として送られた XML データと同じにならない可能性があります。

DBMS は、適切な場合に XML データを最適化します。例えば、空のタグ "<tag><tag/>" は "</tag>" になります。一般的に言って、UTF-16 ではない入力 XML データの XML 宣言でエンコード方式が指定されている場合、PreparedStatement.setString を介してそれを JDBC に渡すべきではありません。その理由は、Strings のデータが UTF-16 で、指定されたエンコード方式に一致しないためです。

照会から XML 宣言が戻される方法を制御する規則がいくつかあります。 JDBC は、ResultSet getter メソッドが呼び出した型に基づいて宣言の可視性を扱います。文字を戻すメソッド (例えば getString()、getClob() など) は、データと共に XML 宣言を戻しません。バイナリー・データを戻すメソッド (例えば getBytes()、getBlob() など) は XML 宣言を戻します。これは、XML 宣言のエンコード方式と表の実際の XML 列の CCSID の相違が原因です。 getString() の呼び出しによって Unicode String が戻されますが、これは XML 宣言のエンコード方式で指定された CCSID と異なる可能性があることに注意してください。このような例外のために、バイナリー getter メソッドを使ってアクセスされる場合を除いて、宣言は単純に破棄されます。 getString のようなメソッドを使って XML タグを取得すると、その宣言が除去された状態になるため、XML データを他の XML データと簡単に連結することができます。

XML JDBC サポートは、以下の IBM Toolbox for Java JDBC メソッドで実装されています。

- AS400JDBCConnection.createSQLXML()
- AS400JDBCSQLXML - この新しいクラスの中のすべてのメソッド
- AS400JDBCPreparedStatement.setSQLXML(int parameterIndex, SQLXML xmlObject)
- AS400JDBCCallableStatement.setSQLXML(String parameterName, SQLXML xmlObject)
- AS400JDBCCallableStatement.getSQLXML(int parameterIndex)
- AS400JDBCCallableStatement.getSQLXML(String parameterName)
- AS400JDBCResultSet.getSQLXML(int columnIndex)
- AS400JDBCResultSet.getSQLXML(String columnLabel)
- AS400JDBCResultSet.updateSQLXML(int columnIndex, SQLXML xmlObject)
- AS400JDBCResultSet.updateSQLXML(String columnLabel, SQLXML xmlObject)
- AS400JDBCRowSet.setSQLXML(int parameterIndex, SQLXML xmlObject)
- AS400JDBCRowSet.setSQLXML(String parameterName, SQLXML xmlObject)

データベース・メタデータの更新

AS400JDBCDatabaseMetaData クラスのさまざまなメソッドを呼び出すことにより、データベース・メタデータが取得されます。 IBM i 7.1 以降、IBM Toolbox for Java JDBC のデフォルト動作として、標準的なシステム・ストアード・プロシージャーのセットからこのメタデータを取得します。これにより、他のプラットフォームの JDBC ドライバーに加えて、IBM i 固有の JDBC サポートもまた IBM Toolbox for Java に備わります。 IBM i 7.1 より前は、IBM Toolbox for Java はホスト・サーバーの ROI データ・サーバーからメタデータを取得していました。この手法は良く機能し、IBM Toolbox for Java は ODBC、.net、および他の IBM i クライアントとも調和してきました。ただし、この手法の問題点として、IBM 全体ですべての JDBC DB2 ドライバーが同じように機能する必要があります。すべての DB2 プラットフォームで共通のシステム・ストアード・プロシージャーのセットを使用することで、それが可能になります。これらのシステム・ストアード・プロシージャーを呼び出すことで、システム・データベース・メタデータが取得されます。

メタデータ機能の後方互換性を提供するには、新しい接続プロパティー "metadata source" を使用することで、データベース・メタデータを取得する古い方式を IBM Toolbox for Java JDBC に強制的に使用させることができます。

「現在コミット済み」のサポート

行レベルのロッキングを実行する分離レベルの下では、ロック・タイムアウトやデッドロックが発生する可能性があります。特に、そのような問題を防ぐよう設計されていないアプリケーションでその可能性が高くなります。スループットの高いデータベース・アプリケーションの中には、トランザクション処理中に出されるロックの待機を許容しないものがあります。また、非コミット・データの処理を許容しないにもかかわらず、読み取りトランザクションでの非ブロッキング動作を要求するアプリケーションもあります。

新しい「*現在コミット済み* (currently committed)」セマンティクスの下では、「現在コミット済み」を有効にした場合、(以前と同じように) コミット済みデータだけが戻されますが、読み取り側は書き込み側による行ロック解放を待機しなくなります。その代わり、読み取り側に戻されるデータは現在コミット済みのバージョン、つまり書き込み操作の開始前のデータに基づきます。

また、この機能は、更新処理中のデータが検出された場合にその結果を待つようデータベース・マネージャーに指示する 1 つの方法を実装します。

IBM Toolbox for Java JDBC には、データ・ソースと接続に関する「現在コミット済み」セマンティクスのサポートが追加されました。このサポートは、以下のクラスの更新で追加されています。

- AS400JDBCDataSource.CONCURRENTACCESS_NOT_SET
- AS400JDBCDataSource.CONCURRENTACCESS_USE_CURRENTLY_COMMITTED
- AS400JDBCDataSource.CONCURRENTACCESS_WAIT_FOR_OUTCOME
- AS400JDBCDataSource.CONCURRENTACCESS_SKIP_LOCKS
- AS400JDBCDataSource.setConcurrentAccessResolution (int)
- AS400JDBCDataSource.getConcurrentAccessResolution ()
- AS400JDBCConnection.setConcurrentAccessResolution (int)
- AS400JDBCConnection.getConcurrentAccessResolution ()

注: 接続に関して「同時アクセス解決」を設定した場合、変更後に接続に関して作成される新しいステートメントだけが影響を受けます。既存のステートメントは、ステートメント作成時に有効であった値を使用し続けます。また、この情報を設定するために、これに対応する接続プロパティー `"concurrent access resolution"` を使用することもできます。

配列型のサポート

IBM Toolbox for Java では、ストアード・プロシージャーのパラメーターで IBM i 7.1 SQL 配列データ型がサポートされます。ロケーターで戻されるデータを除いて、さまざまな DB2 型の配列がすべてサポートされます。

IBM Toolbox for Java JDBC では、ストアード・プロシージャーの IN、OUT、および INOUT パラメーターとして配列が追加的にサポートされます。ただし、配列を含むストアード・プロシージャーまたは他の照会から戻される ResultSets はサポートされません。

JDBC では、java.sql.CallableStatement クラスの中のストアード・プロシージャー呼び出しがサポートされます (IBM Toolbox for Java ではこれが AS400JDBCCallableStatement で実装されます)。

Sun の JDBC 仕様によると、JDBC ドライバーは java.sql.Array インターフェースを実装することで配列をサポートします。 IBM Toolbox for Java 実装はクラス com.ibm.as400.access.AS400JDBCArray の中に存在します。以下は、配列をサポートするために IBM Toolbox for Java で実装されている新しいインターフェースとメソッドです。

- AS400JDBC.PreparedStatement.setArray (int parameterIndex, Array x)
- AS400JDBCCallableStatement.getArray()
- AS400JDBCArray.getResultSet()
- AS400JDBCArray.getBaseType()
- AS400JDBCArray.getBaseTypeName()
- AS400JDBCArrayResultSet - この新しいクラスの中のすべてのメソッド
- AS400JDBConnection.createArrayOf(String typeName, Object[] elements)

注: AS400JDBCArrayResultSet は、ResultSet インターフェース・オブジェクト内の配列データのローカル・コピーです。

長いスキーマ名のサポート

IBM i 7.1 DBMS では 128 バイトのスキーマ名のサポートが追加されています。さらに、IBM Toolbox for Java JDBC は長いスキーマ名を追加的にサポートします。

IBM Toolbox for Java は、ホスト DBMS にデータを送る前にスキーマを Java Strings からシステム CCSID に変換します。 IBM Toolbox for Java では、デフォルト・スキーマとして長いスキーマ名を使用できます。ただし、ライブラリー・リスト内の長いスキーマ名はサポートされません。 "libraries" プロパティーで長いスキーマ名が指定された場合、ホストから警告が戻され、ライブラリーが追加されなかったというトレースが発行されます。 "libraries" プロパティーで最初の名前として長いスキーマ名が指定された場合、IBM Toolbox for Java はその名前をデフォルト・スキーマとして設定しますが、ライブラリー・リストにはそれを追加しません。ライブラリー・リスト内の長いスキーマ名を使用する必要があるクライアントは、DB2 SET PATH コマンドを使用する必要があります。

また、データベース・メタデータでも、パラメーターおよび戻される ResultSets で長いスキーマ名がサポートされます。

以下のメタデータ・メソッドで、長いスキーマ名がサポートされます。

- AS400JDBCDatabaseMetadata.getMaxSchemaNameLength()
- AS400JDBCDatabaseMetadata.getProcedures(String catalog, String schemaPattern, String procedureNamePattern)
- AS400JDBCDatabaseMetadata.getProcedureColumns(String catalog, String schemaPattern, String procedureNamePattern, String columnNamePattern)
- AS400JDBCDatabaseMetadata.getTables(String catalog, String schemaPattern, String tableNamePattern, String types[])
- AS400JDBCDatabaseMetadata.getSchemas()
- AS400JDBCDatabaseMetadata.getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)
- AS400JDBCDatabaseMetadata.getColumnPrivileges(String catalog, String schema, String table, String columnNamePattern)
- AS400JDBCDatabaseMetadata.getTablePrivileges(String catalog, String schemaPattern, String tableNamePattern)
- AS400JDBCDatabaseMetadata.getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable)
- AS400JDBCDatabaseMetadata.getVersionColumns(String catalog, String schema, String table)

- AS400JDBCDatabaseMetadata.getPrimaryKeys(String catalog, String schema, String table)
- AS400JDBCDatabaseMetadata.getImportedKeys(String catalog, String schema, String table)
- AS400JDBCDatabaseMetadata.getCrossReference(String parentCatalog, String parentSchema, String parentTable, String foreignCatalog, String foreignSchema, String foreignTable)
- AS400JDBCDatabaseMetadata.getIndexInfo(String catalog, String schema, String table, boolean unique, boolean approximate)
- AS400JDBCDatabaseMetadata.getUDTs(String catalog, String schemaPattern, String typeNamePattern, int[] types)
- AS400JDBCDatabaseMetadata.getSuperTypes(String catalog, String schemaPattern, String typeNamePattern)
- AS400JDBCDatabaseMetadata.getSuperTables(String catalog, String schemaPattern, String tableNamePattern)
- AS400JDBCDatabaseMetadata.getAttributes(String catalog, String schemaPattern, String typeNamePattern, String attributeNamePattern)
- AS400JDBCDatabaseMetadata.getSchemas(String catalog, String schemaPattern)
- AS400JDBCDatabaseMetadata.getFunctions(String catalog, String schemaPattern, String functionNamePattern)
- AS400JDBCDatabaseMetadata.getFunctionColumns(String catalog, String schemaPattern, String functionNamePattern, String columnNamePattern)

IBM Toolbox for Java JDBC support for IBM i 6.1 に対する機能拡張

IBM i 6.1 では、JDBC サポートに対して多数の追加が行われました。

IBM Toolbox for Java JDBC サポートの拡張については、この後の項で詳述します。

- 『JDBC 4.0 のサポート』
- 『照会ストレージの限界』
- 348 ページの『10 進浮動小数 (DECFLOAT) データ・タイプ』
- 349 ページの『サーバーへのクライアント・タイプとアプリケーション名の引き渡し』
- 350 ページの『カーソル名の最大長の拡張』
- 350 ページの『生成されたキーのサポート』
- 350 ページの『デフォルト値のサポートの改善』
- 350 ページの『GROUP BY 文節での最大値の増大』
- 351 ページの『バッチ更新のサポート』

JDBC 4.0 のサポート

以下のいずれかの JAR ファイルを使用すると、Java SE バージョン 6 で JDBC 4.0 API をサポートすることができます。

- `/QIBM/ProdData/HTTP/Public/jt400/lib/java6/jt400.jar`
- `/QIBM/ProdData/OS400/jt400/lib/java6/jt400Native.jar`

照会ストレージの限界

照会ストレージ限界プロパティーを使用して、照会で使用されるストレージを制限することができます。このプロパティーは、照会の推定ストレージ使用量に対して、指定したストレージの限界を比較します。推定ストレージ使用量が、指定したストレージ限界を超える場合、照会を実行できません。

V6R1 より、IBM Toolbox for Java JDBC ドライバーは、AS400JDBCDataSource クラスに以下のメソッドを追加します。

```
setQueryStorageLimit()
public void setQueryStorageLimit(int limit);
```

接続でステートメントが実行されるときに使用する照会ストレージ限界値を指定します。有効な値は、-1 から 2 147 352 578 までです。デフォルト値は -1 です。これは、特殊値 *NOMAX を示します。

```
getQueryStorageLimit()
public int getQueryStorageLimit()
```

接続でステートメントが実行されるときに使用する照会ストレージ限界値を戻します。デフォルト値は -1 です。これは、特殊値 *NOMAX を示します。

V6R1 より前のリリースの IBM i を実行するシステムでは、getQueryStorageLimit プロパティーは無視されます。

10 進浮動小数 (DECFLOAT) データ・タイプ

10 進浮動小数 (DECFLOAT) は、浮動小数と 10 進の両方のデータ・タイプから望ましいプロパティーを継承します。 DECFLOAT のデータ値は、その末尾のゼロが有効になるように保管されます。たとえば、2.0 と 2.00 は、バイナリー表現では異なります。ただし SQL 比較では、これらの値は等しいものとして扱われます。

IBM Toolbox for Java JDBC ドライバーは、次のようなメソッドを AS400JDBCDataSource クラスに追加しました。

```
setDecfloatRoundingMode()
public void setDecfloatRoundingMode(int String mode)
```

DECFLOAT 数値に対して使用する丸めモードを指定します。

```
getDecfloatRoundingMode()
public intString getDecfloatRoundingMode()
```

DECFLOAT 数値に対して使用する丸めモードを戻します。

これらのメソッドの有効な値には、次のものがあります。

偶数丸め (デフォルト値)
最も近い数字に丸めます。 2 つの数字から等距離の場合、最も近い偶数に丸めます。この定数の数値は 0 です。

四捨五入
最も近い数字に丸めます。 2 つの数字から等距離の場合は切り上げます。この定数の数値は 1 です。

切り捨て
より小さいほうの最も近い数字に丸めます。この定数の数値は 2 です。

上限値に切り上げ
正の無限大に丸めます。この定数の数値は 3 です。

下限値に切り捨て
負の無限大に丸めます。この定数の数値は 4 です。

五捨六入
最も近い数字に丸めます。 2 つの数字から等距離の場合は、切り捨てます。この定数の数値は 5 です。

切り上げ
より大きいほうの最も近い数字に丸めます。この定数の数値は 6 です。

サーバーへのクライアント・タイプとアプリケーション名の引き渡し

Web アプリケーションには、エンド・ユーザーとクライアントの情報をデータベースに渡すことによって、さらに詳しい情報が記録されるようにする手段が必要です。 IBM Toolbox for Java JDBC ドライバーでは、アプリケーションは、次のような java.sql.Connection.setClientInfo() メソッドを呼び出すことによって、この情報を指定変更することができます。

```
void AS400JDBCConnection.setClientInfo(java.lang.String name,java.lang.String value)
```

このメソッドは、名前で指定されたクライアント情報プロパティーの値を、value で指定された値に設定します。 IBM Toolbox for Java JDBC ドライバーでサポートされるクライアント情報プロパティーの詳細は、DatabaseMetadata.getClientInfoProperties メソッドの解説を参照してください。

```
void AS400JDBCConnection.setClientInfo(java.util.Properties properties)
```

このメソッドは、接続のクライアント情報プロパティーの値を設定します。 Properties オブジェクトには、設定するクライアント情報プロパティーの名前と値が入っています。プロパティー・リストに収められている一連のクライアント情報プロパティーによって、接続上の現在の一連のクライアント情報プロパティーが置き換えられます。接続上で現在設定されているプロパティーがプロパティー・リスト中にない場合、そのプロパティーは消去されます。空のプロパティー・リストを指定すると、接続上のすべてのプロパティーが消去されます。

```
String AS400JDBCConnection.getClientInfo(java.lang.String name)
```

このメソッドは、name で指定されたクライアント情報プロパティーの値を戻します。指定されたクライアント情報プロパティーが設定されていなくて、デフォルト値を持っていない場合、このメソッドはヌル を戻すことがあります。

```
Properties AS400JDBCConnection.getClientInfo()
```

このメソッドは、ドライバーでサポートされている各クライアント情報プロパティーの名前と現行値を示したリストを戻します。クライアント情報プロパティーが設定されていなくて、デフォルト値を持っていない場合、クライアント情報プロパティーの値はヌルになることがあります。

```
ResultSet AS400JDBCDatabaseMetaData.getClientInfoProperties()
```

このメソッドは、ドライバーがサポートするクライアント情報プロパティーのリストを検索します。 IBM Toolbox for Java JDBC ドライバーは、次のような情報を添付した結果セットを戻します。

表 2. getClientInfoProperties の結果セット

名前	最大長	デフォルト値	説明
ApplicationName	255	""	現在接続を利用しているアプリケーションの名前。

表 2. *getClientInfoProperties* の結果セット *(続き)*

名前	最大長	デフォルト値	説明
ClientAccounting	255	""	アカウント情報。
ClientHostname	255	""	接続を使用するアプリケーションが実行されているコンピューターのホスト名。
ClientProgramID	255	""	クライアント・プログラムの ID。
ClientUser	255	""	接続を使用するアプリケーションで作業を行っているユーザーの名前。 これは、接続の確立に使用されたユーザー名とは異なる場合があります。

カーソル名の最大長の拡張

V6R1 より、カーソルの最大長は 128 文字になります。これまでの最大長は 8 文字でした。アプリケーションは、java.sql.Statement.setCursorName() メソッドを呼び出して、カーソルの名前を設定することができます。

生成されたキーのサポート

これまでのリリースでは、複数行の挿入操作から戻される情報は 1 行のみでした。 V6R1 より、複数行の挿入操作に関するさらに詳しい情報にアクセスできます。それによって、ROWID、ID 列、シーケンス、または生成された式などの、生成列情報を検索できるようになります。生成されたキーのサポートは、次のような IBM Toolbox for Java JDBC メソッドにインプリメントされます。

- AS400JDBCStatement.getGeneratedKeys()
- AS400JDBCStatement.execute(String sql, int autoGeneratedKeys)
- AS400JDBCStatement.execute (String sql, int[] columnIndexes)
- AS400JDBCStatement.execute (String sql, String[] columnNames)
- AS400JDBCStatement.executeUpdate(String sql, int[] autoGeneratedKeys)
- AS400JDBCStatement.executeUpdate (String sql, int[] columnIndexes)
- AS400JDBCStatement.executeUpdate (String sql, String[] columnNames)
- AS400JDBCConnection.prepareStatement(String sql, int autoGeneratedKeys)
- AS400JDBCConnection.prepareStatement(String sql, int[] columnIndexes)
- AS400JDBCConnection.prepareStatement(String sql, String[] columnNames)

デフォルト値のサポートの改善

V6R1 より、IBM Toolbox for Java JDBC ドライバーは、DatabaseMetaData.getColumns() メソッドを介して、列のデフォルト値をストリングで戻します。デフォルト値がヌルの場合、値 'NULL' の付いたストリングが戻されます。

GROUP BY 文節での最大値の増大

DatabaseMetaData getMaxColumnsInGroupBy( ) の新しい値は 8000 になっています。

バッチ更新のサポート

V6R1 ではバッチ更新のサポートが改善されたため、複数行のバッチ挿入ステートメントの実行時に提供される情報が向上します。診断域と SQLCA にはフィールドが設けられ、そこには、正常完了したステートメント数が示されるので、エラーのロケーションの判別が容易になります。

IBM Toolbox for Java の JDBC ドライバーは、java.sql.BatchUpdateException の作成時にその情報を利用することができます。 AS400JDBCStatement および AS400JDBCPreparedStatement もまた、この情報を使用して、executeBatch() メソッドからの正しい情報を戻すことができます。

IBM Toolbox for Java JDBC support for IBM i 5.4 に対する機能拡張

いくつかの JDBC 機能が、IBM i 5.4 に対して拡張されました。

IBM Toolbox for Java JDBC サポートの拡張については、この後の項で詳述します。

- 『2 MB ステートメント・サイズ』
- 『128 バイト列名サポート』
- 『データベース・ホスト・サーバーのトレース・サポート』
- 352 ページの『eWLM Correlator サポート』

前のリリースの拡張 JDBC 機能について詳しくは、 352 ページの『バージョン 5 リリース 3 に対する JDBC の拡張』 および 354 ページの『i5/OS バージョン 5 リリース 2 用に拡張された JDBC 機能』を参照してください。

2 MB ステートメント・サイズ

V5R4 より前では、SQL ステートメント・サイズの限界は 65 535 バイトでした。これは、ステートメント・テキストが 1 バイト CCSID を使用して表現される場合は、65 535 文字に、ステートメント・テキストが 2 バイト CCSID を使用して表現される場合は、32 767 文字に相当します。お客様の中でも、特に自動的に SQL ステートメントを生成するアプリケーションをご使用のお客様は、この制限に影響されていました。

V5R4 では、IBM i ステートメント・サイズの限界は、2 メガバイト、または 2 097 152 バイトに拡張されました。 IBM Toolbox for Java JDBC ドライバーは常に 2 バイト Unicode で、ステートメント・テキストを送ります。したがって、ステートメントの最大長は文字数で、1 メガバイトまたは 1 048 576 文字になります。

128 バイト列名サポート

V5R4 以降、データベースは、SQL 表に対して 128 バイトまでの列名をサポートします。V5R4 より前では、30 バイトまでの列名がサポートされていました。IBM Toolbox for Java JDBC ドライバーによって、より長い名前が提供できるようになります。

128 バイトの列名が戻されない例外が、1 つあります。ローカル・パッケージ・キャッシングを使用し、列名が 30 文字を超える場合には、サーバーはシステム列名として、列名を戻します。

データベース・ホスト・サーバーのトレース・サポート

データベース・ホスト・サーバーのトレースをオンにする新規のオプションが、 Toolbox for Java JDBC ドライバーに追加されました。このフィーチャーをサポートするために、オプション "64" が "server

trace" 接続プロパティーに追加されました。詳細については、 356 ページの『IBM Toolbox for JavaJDBC プロパティー』 を参照してください。

eWLM Correlator サポート

IBM Toolbox for Java は、 IBM Enterprise Workload Manager (eWLM) correlator を受け入れ、 Application Response Measurement (ARM) API とともに使用する接続属性 correlator として、ホストに受け渡します。この correlator は、接続が完了した後いつでも AS400JDBCConnection クラスにある次のメソッドを使用して、ホストへ送ることができます。

setDB2eWLMCorrelator

```
public void setDB2eWLMCorrelator(byte[] bytes)
                          throws SQLException
```

eWLM Correlator を設定する。有効な correlator 値が使用されると仮定します。値がヌルの場合、すべての ARM/eWLM インプリメンテーションは、オフにされます。eWLM correlator は、i5/OS V5R3 またはこれ以降のサーバーを必要とします。この要求は、V5R2 (またはこれ以前) を実行するサーバーでは無視されます。

パラメーター:

- `bytes`: eWLM correlator 値
- `SQLException`: Sun Microsystems, Inc. Web サイトの クラス SQLException 情報を参照してください。

前のリリースの拡張 JDBC 機能について詳しくは、『バージョン 5 リリース 3 に対する JDBC の拡張』および 354 ページの『i5/OS バージョン 5 リリース 2 用に拡張された JDBC 機能』を参照してください。

バージョン 5 リリース 3 に対する JDBC の拡張

いくつかの JDBC 機能が、i5/OS バージョン 5 リリース 3 に対して拡張されました。

i5/OS バージョン 5 リリース 3 用に拡張された JDBC 機能には以下のものがあります。

- UTF-8 および UTF-16 サポート
- Binary および Varbinary サポート
- より高い小数点精度のサポート
- 2 GB の大容量オブジェクトのサポート
- 非センシティブ・カーソル・サポート
- マテリアライズ照会表のサポート

前のリリースの拡張 JDBC 機能について詳しくは、IBM Toolbox for JavaJDBC サポートに対する V5R2 拡張機能参照してください。

UTF-8 および UTF-16 サポート

UTF-8 データは 1208 の CCSID と一緒に文字フィールドに保管されます。UTF-8 文字は、非結合文字のバイト (1、2、3、または 4) の変数番号、および結合文字の任意の番号です。文字フィールドに指定された長さは、フィールドに入れることのできる最大バイト数です。 UTF-8 1208 CCSID を使用して、以下のデータ型にタグを付けることができます。

- 固定長文字 (CHAR)

- 可変長文字 (VARCHAR)
- 文字 LOB (CLOB)

UTF-16 データは 1200 の CCSID と一緒にグラフィックス・フィールドに保管されます。UTF-16 文字の長さは、非結合文字には 2 バイトまたは 4 バイト (つまり、代理) に、結合文字には任意のバイト数をとることができます。グラフィック・データ・フィールドに指定された長さは、フィールドに入れることのできる 2 バイトの最大数です。 UTF-16 1200 CCSID を使用して、以下のデータ型にタグを付けることができます。

- 固定長グラフィック (GRAPHIC)
- 可変長グラフィック (VARGRAPHIC)
- 2 バイト文字 LOB (DBCLOB)

Binary および Varbinary サポート

BINARY および VARBINARY データ型は、CHAR および VARCHAR データ型に似ていますが、これには文字データではなくバイナリー・データが含まれます。 BINARY フィールドは固定長です。 VARBINARY フィールドの長さは可変です。BINARY および VARBINARY データ型には以下の特性があります。

- バイナリー形式のコード化文字セット ID (CCSID) は 65535 です。
- 割り当てと比較において、バイナリー・データ型は他のバイナリー・データ型 (BINARY、VARBINARY、および BLOB) とのみ互換性があります。
- バイナリー・データ型用の埋め込み文字は、ブランク文字ではなく x'00' です。
- 切り捨てエラーを避けるために末尾の文字をストリップする必要がある状況では、x'00' 文字が末尾ブランクの代わりにストリップされます。
- バイナリー・データ型を比較する場合、2 つのフィールドが等しくなるためにはデータと長さが同じでなければなりません。後続ゼロは比較では無視されません。
- バイナリー・データ型を比較する際に、2 つのフィールドの長さが異なる場合、 2 つのフィールドが短いフィールドの長さまで同じであれば、短いフィールドは長いフィールドより小さいとみなされます。

より高い小数点精度のサポート

現在、小数点精度は 63 桁までサポートします。3 つのプロパティー、"minimum divide scale"、"maximum precision"、および "maximum scale" が追加され、6 つのメソッド、setMinimumDivideScale(int divideScale)、 getMinimumDivideScale()、 setMaximumPrecision(int precision)、 getMaximumPrecision()、 setMaximumScale(int scale)、および getMaximumScale() が AS400JDBCDataSource に追加されています。"Minimum divide scale" は、小数点除算の結果の最小位取り値を指定し、0 から 9 までの数値に設定されます。 "Maximum precision" は、データベースが使用する最大小数点精度を指定し、31 または 63 のいずれかに設定されます。 "Maximum scale" は、データベースが使用する最大位取りを指定し、0 から 63 までの数値に設定されます。

2 GB の大容量オブジェクト (LOB) のサポート

IBM Toolbox for Java JDBC の機能拡張により、最大 2 GB LOB のサポートを使用することができるようになりました。

非センシティブ・カーソル・サポート

カーソル・サポートは非センシティブ・カーソルをサポートするようになりました。ResultSet を TYPE_SCROLL_INSENSITIVE と一緒に使用する場合、非センシティブ・カーソルが使用されます。 ResultSet は、オープン時には基礎となるデータベースへの変更を表示しません。

マテリアライズ照会表のサポート

DatabaseMetaData.getTables() への呼び出しで TABLE_TYPE として "MATERIALIZED QUERY TABLE" を戻します。

i5/OS バージョン 5 リリース 2 用に拡張された JDBC 機能

いくつかの JDBC 機能が、i5/OS バージョン 5 リリース 2 に対して拡張されました。

i5/OS 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。i5/OS の V5R1 以降のバージョンに接続する場合、IBM 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スローします。

現在 IBM 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 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 JavaJava 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、i5/OS 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

IBM 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、IBM 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。列名が引用符で囲まれている場合、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、IBM Toolbox for Java は例外をスローします。

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. IBM 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 i5/OS 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 以前は、接続が作成された後に *NONE に切り替えられると IBM Toolbox for Java JDBC ドライバーは例外をスローしていました。

IBM Toolbox for JavaJDBC 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オプションであり、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』
- 357 ページの『サーバーのプロパティー』
- 360 ページの『形式のプロパティー』
- 361 ページの『パフォーマンスのプロパティー』
- 366 ページの『ソートのプロパティー』
- 367 ページの『その他のプロパティー』

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。

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	システム・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。

プロパティー	説明	必須かどうか	選択項目	デフォルト値
"auto commit"	新しい接続のデフォルト接続モードを自動コミット・モードに設定するかどうかを指定します。AS400JDBCConnection.setAutoCommit() を呼び出すと、各接続ごとにこのプロパティーがオーバーライドされます。 **注:** 自動コミット・モードの使用時に *NONE 以外のトランザクション分離レベルを使用するには、"true autocommit" プロパティーを true に設定する必要があります。	いいえ	"true" "false"	"true"
"concurrent access resolution"	接続で「現在コミット済み」アクセスを使用するかどうかを指定します。値 1 は「現在コミット済み」を使用することを示します。値 2 は「発信待機」を使用することを示します。値 3 は「ロックのスキップ」を使用することを示します。	いいえ	"1" "2" "3"	(システム・デフォルト)
"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"

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 sensitivity"	データベースから要求す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。	いいえ	"asensitive" "sensitive" "insensitive"	"asensitive"
"database name"	独立補助記憶域プール (ASP) への接続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、IBM i サーバーへの接続時にのみ適用されます。データベース名を指定する場合、その名前はサーバーのリレーショナル・データベース・ディレクトリーに存在する必要があり、独立 ASP またはシステム・デフォルト・データベースのどちらかに該当する必要があります。 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。 • このプロパティーを使って独立 ASP に該当するデータベースを指定した場合、独立 ASP への接続が行われます。 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初期 ASP グループを使用します。ジョブ記述が初期 ASP グループ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。
"decfloat rounding mode"	decfloat データ・タイプでの処理時に使用する丸めモードを指定します。 IBM i 5.4 以前のバージョン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	"half even" "half up" "down" "ceiling" "floor" "half down" "up"	"half even"

プロパティー	説明	必須かどうか	選択項目	デフォルト値
"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 SQL スキーマをオプションで設定します (デフォルト・ライブラリー)。ライブラリーの長さは 10 文字を超えることができません。 10 文字より長いライブラリーがある場合、SET PATH SQL ステートメントを使用する必要があります。 **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。 ライブラリー・リスト・プロパティーについて詳しくは、JDBC LibraryList プロパティーを参照してください。 **デフォルト SQL スキーマ**: サーバーはデフォルト SQL 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 SQL スキーマのみを調べます。接続 URL にデフォルト SQL スキーマを指定できます。接続 URL でデフォルト SQL スキーマを指定しない場合、使用される SQL 命名規則に応じて下記の条件が当てはまります。 • **SQL ネーミング**が使用される場合、 – 最初の項目は (*LIBL でない限り) デフォルト SQL 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 SQL 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 SQL スキーマになる。 • **システム・ネーミング**が使用される場合、 – デフォルト SQL 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。	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"
"maximum precision"	データベースが使用する 10 進数の最大精度を指定します。	いいえ	"31" "63"	"31"
"maximum scale"	データベースが使用する最大の位取りを指定します。	いいえ	"0"-"63"	"31"

プロパティー	説明	必須かどうか	選択項目	デフォルト値
"minimum divide scale"	10 進数の除算の結果のための最小の位取り値を指定します。	いいえ	"0" "1" "2" "3" "4" "5" "6" "7" "8" "9"	"0"
"package ccsid"	サーバーに送信される SQL パッケージおよびすべてのステートメントに使用する文字エンコードを指定します。	いいえ	"1200" (UCS-2) "13488" (UTF-16)	"13488"
"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"
"translate hex"	16 進数リテラルの解釈方法を指定します。	いいえ	"character" (16 進数リテラルを文字データと解釈する) "binary" (16 進数リテラルを 2 進数データとして解釈する)	"character"
"true autocommit"	接続で真の自動コミットのサポートを使用するかどうかを指定します。真の自動コミットとは、自動コミットがオンであり、かつ *NONE 以外の分離レベルで実行されていることを意味します。デフォルトでは、サーバー分離レベル *NONE で実行することにより、ドライバーが自動コミットを処理します。	いいえ	"true" (真の自動コミットを使用する。) "false" (真の自動コミットを使用しない。)	"false"
"xa loosely coupled support"	疎結合トランザクション・ブランチに対して、ロック共用を許可するかどうかを指定します。 **注:** IBM i 5.3 またはそれ以前を実行している場合、この設定は無視されます。	いいえ	"0" = ロックは共有できない "1" = ロックは共有できる	"0"

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。

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。このプロパティーは、「時刻形式」プロパティーが "hms" に設定されていない場合には効果がありません。	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。

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、パフォーマンスが改善されます。 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" (非レコード・ブロック) "1" (FOR FETCH ONLY が指定されていると、ブロック) "2" (FOR UPDATE の指定がない限り、ブロック)	"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、パフォーマンスが改善され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"

プロパティー	説明	必須かどうか	選択項目	デフォルト値
"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"	後続の要求があるまでカーソルのクローズを遅延さ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改善されます。	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"	"32768"
"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	""

プロパティー	説明	必須かどうか	選択項目	デフォルト値
"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) "select" (すべての SQL SELECT ステートメントをパッケージに保管)	"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 を発行するか、または接続に対する通知を出します。 "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 ステートメントの実行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改善されます。	いいえ	"true" "false"	"true"

プロパティー	説明	必須かどうか	選択項目	デフォルト値
"qaqqinilib"	QAQQINI ライブラリー名を指定します。使用する qaqqini ファイルに含まれるライブラリーを指定するのに使用します。qaqqini ファイルには、DB2 for i データベース・エンジンのパフォーマンスに影響を与える可能性のある属性のすべてが含まれます。	いいえ	"QAQQINI library name"	(サーバー・デフォルト)
"query optimize goal"	Query の最適化で、サーバーが使用すべきゴールを指定します。この設定値は、OPTIMIZATION_GOAL と呼ばれるサーバーの QAQQINI オプションに相当します。 **注:** IBM i 5.3 以前のバージョン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	"0" = 拡張動的パッケージを使用の場合、データの最初のブロック (*FIRSTIO) に対する Query を最適化します。パッケージを使用しない場合、結果セットの全体 (*ALLIO) に対する Query を最適化します。 "1" = データの最初のブロック (*FIRSTIO) に対する Query を最適化 "2" = 結果セットの全体 (*ALLIO) に対する Query を最適化	"0"
"query storage limit"	照会で使用されるストレージを制限します。このプロパティーは、照会の推定ストレージ使用量に対して、指定したストレージの限界を比較します。推定ストレージ使用量が、指定したストレージ限界を超える場合、照会を実行できません。	いいえ	-1 (*NOMAX) から 2 147 352 578 まで	-1。これは、特殊値 *NOMAX を示します。

プロパティー	説明	必須かどうか	選択項目	デフォルト値
"receive buffer size"	フロントエンド・ドライバーとシステムの間のソケット接続を通してデータを受信するのに使用されるバッファー・サイズを指定します。 **注:** これは、実際の受信バッファー・サイズは指定しません。これは、基盤となるソケット・コードでのヒントとして使用されるだけです。	いいえ	"1" - 最大サイズ	(プラットフォームによって異なります)
"send buffer size"	フロントエンド・ドライバーとシステムの間のソケット接続を通してデータを送信するのに使用されるバッファー・サイズを指定します。 **注:** これは、実際の送信バッファー・サイズは指定しません。これは、基盤となるソケット・コードでのヒントとして使用されるだけです。	いいえ	"1" - 最大サイズ	(プラットフォームによって異なります)
"variable field compression"	可変長フィールドを圧縮する必要があるかどうかを指定します。	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 進の値を基にソートする) "language" ("sort language" プロパティーで設定する言語を基にソートする) "table" ("sort table" プロパティーで設定するソート・シーケンス表を基にソートする)	"hex"
"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	""

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 weight"	レコードをソートする際にサーバーが大文字小文字をどのように扱うのかを指定します。 "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、両方向ストリング型、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 ステートメントを許可する) "read call" (SELECT および CALL ステートメントを許可する) "read only" (SELECT ステートメントのみ)	"all"
"autocommit exception"	自動コミットが使用可能になっている場合に、Connection.commit() または Connection.rollback() が呼び出されたら SQLException をスローするかどうかを指定します。	いいえ	"true" "false"	"false"
"bidi string type"	両方向データの出力ストリング型を指定します。詳細については、BidiStringType を参照してください。	いいえ	"" (両方向ストリング型を判別するのに CCSID を使用する) "0" (非両方向データ (LTR) に対するデフォルトのストリング型) "4" "5" "6" "7" "8" "9" "10" "11"	""

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
"bidi implicit reordering"	bidi (双方向) 暗黙 LTR-RTL 再配列を使用するかどうかを指定します。	いいえ	"true" "false"	"true"
"bidi numeric ordering"	数値配列双方向互換機能を使用するかどうかを指定します。	いいえ	"true" "false"	"false"
"data truncation"	**文字データ**の切り捨ての際に通知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。 • 切り捨てられた文字データがデータベースに書き込まれると、例外が発行されます。 • 切り捨てられた文字データを照会に使用すると、通知が出されます。 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、例外または通知は生成されません。 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通知が出されます。	いいえ	"true" "false"	"true"
"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 IBM i JVM 環境の場合、ネイティブの IBM Developer Kit for Java 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 IBM 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"

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
"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) および ResultSetMetaData.isAutoIncrement(int) メソッドをサポートできます。 **注:** 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、パフォーマンスが低下す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、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。	いいえ	"true" "false"	"false"
"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"
"hold input locators"	入力ロケーターを、タイプ保持ロケーターとして割り振るか、非保持ロケーターとして割り振るかを指定します。ロケーターがタイプ保持の場合、コミット実行時に解放されません。	いいえ	"true" (タイプ保持) "false"	"true"

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
"hold statements"	自動コミットがオフで、ステートメントが LOB ロケーターに関連付けられている場合に、トランザクション境界までそれらのステートメントをオープンのままにするかどうかを指定します。デフォルトでは、ステートメントのクローズ時に、そのステートメントに関連するすべてのリソースが解放されます。このプロパティーを true に設定するのは、ステートメントのクローズ後に LOB ロケーターへのアクセスが必要になった場合だけにしてください。	いいえ	"true" "false"	"false"
"ignore warnings"	ドライバーによる警告オブジェクト作成の対象としない SQL 状態のリストを指定します。デフォルトでは、IBM Toolbox for Java JDBC ドライバーはデータベースから戻されるそれぞれの警告に対して java.sql.SQLWarning オブジェクトを内部的に作成します。例えば、ストアード・プロシージャーから結果セットが戻されるたびに、SQLSTATE 0100C を伴う警告が作成されます。ストアード・プロシージャーを呼び出すアプリケーションのパフォーマンスを改善するために、この警告を無視しても安全です。	いいえ	無視すべき SQL 状態から成るコンマ区切りリスト。	""
"keep alive"	ソケット接続の状況が操作可能であることを定期的に検査するかどうかを指定します。	いいえ	"true" "false"	(プラットフォームによって異なります)
"key ring name"	サーバーとの SSL 接続で使用する鍵リング・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鍵リング・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 通信で使用する鍵リング・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鍵リング名を設定しない限り、このプロパティーには効力はありません。	いいえ	"key ring password"	""
"metadata source"	DatabaseMetaData の検索方法を指定します。これを "0" に設定した場合、Retrieve Object Information (ROI) (オブジェクト情報の検索 (ROI)) データ・フローを通して、データベース・メタデータが検索されます。 これを "1" に設定した場合、システムのストアード・プロシージャーの呼び出しによって、データベース・メタデータが検索されます。	いいえ	"0" (ROI アクセス) "1" (SQL ストアード・プロシージャー)	IBM i 6.1 以前のリリースでは "0"、他のすべてのリリースでは "1"

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
"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 はオプションです。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・コメント) "system" (IBM i オブジェクト記述)	"system"
"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。このプロパティーでは、このドライバーを使って他のデータベースに接続することができます。 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。	いいえ	"true" (すべてのクライアント/サーバー通信を暗号化する) "false" (パスワードのみを暗号化する)	"false"

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
"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ない) "2" (JDBC サーバー・ジョブでデータベース・モニターを開始する) "4" (JDBC サーバー・ジョブでデバッグを開始する) "8" (JDBC サーバー・ジョブが終了したらジョブ・ログを保管する) "16" (JDBC サーバー・ジョブでジョブ・トレースを開始する) "32" (SQL 情報を保管する) "64" (データベース・ホスト・サーバーのトレースのアクティベーションをサポートする) 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。	"0"
"thread used"	ホスト・サーバーとの通信でスレッドを使用するかどうかを指定します。	いいえ	"true" "false"	"true"

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
"toolbox trace"	IBM Toolbox for Java トレースのどのカテゴリーをログに記録するかを指定します。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設定します。トレース・メッセージは System.out に記録されます。	いいえ	"" "none" "datastream" (ローカル・ホストとリモート・システムの間のデータ・フローをログに記録する) "diagnostic" (オブジェクトの状態情報をログに記録する) "error" (例外の原因となるエラーをログに記録する) "information" (コードを使用して制御のフローを追跡するのに使用する) "warning" (リカバリー可能なエラーをログに記録する) "conversion" (Unicode とネイティブ・コード・ページの間の文字セット変換をログに記録する) "proxy" (クライアントと Proxy サーバーの間のデータ・フローをログに記録する) "pcml" (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る) "jdbc" (JDBC 情報をログに記録する) "all" (すべてのカテゴリーをログに記録する) "thread" (スレッド情報をログに記録する)	""

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
"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"
"translate boolean"	PreparedStatement.setObject()、CallableStatement.setObject()、または ResultSet.updateObject() メソッドを使用して文字フィールド/パラメーターの値を設定するときのブール・オブジェクトの解釈方法を指定します。このプロパティーを "true" に設定すると、ブール・オブジェクトは、"true" または "false" として文字フィールドに保管されます。このプロパティーを "false" に設定すると、ブール・オブジェクトは、"1" または "0" として文字フィールドに保管されます。	いいえ	"true" "false"	"true"

関連資料

86 ページの『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 の JDBC ドライバーを DriverManager に登録する必要があります。

JDBC Librarylist プロパティー

JDBC LibraryList プロパティーは、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・ライブラリーをオプションで指定します (デフォルト SQL スキーマ)。

以下の表の例では、次に示す前提事項があります。

- MYLIBDAW というライブラリーには MYFILE_DAW が含まれます。
- 以下の SQL ステートメントを実行しています。

```
"SELECT * FROM MYFILE_DAW"
```

シナリオ	SQL ネーミング	システム・ネーミング
基本的な規則	検索されるライブラリーは 1 つだけです。 • URL 上でデフォルト SQL スキーマを指定した場合、それが使用されます。それがデフォルトのライブラリーになります。 • URL 上でデフォルト SQL スキーマを指定しなかった場合、"libraries" プロパティー内の最初のライブラリーが使用されます。それがデフォルトのライブラリーになります。 • URL 上でデフォルト SQL スキーマおよび "libraries" プロパティーが指定されなかった場合、サインオンされたユーザー・プロファイルと同じ名前のライブラリーが使用されます。 ジョブのライブラリー・リストはライブラリー・プロパティー内のライブラリーを使用して更新されます。これにより、一部のトリガーおよびストアード・プロシージャーの動作に影響がある可能性があります。これは、ステートメント内の修飾なしの名前には影響しません。	ジョブのライブラリー・リストはライブラリー・プロパティーのライブラリーを使用して更新されます。あるデフォルト SQL スキーマが URL 上で指定される場合、それがデフォルトのライブラリーになります。
1. URL にデフォルト SQL スキーマが指定されていません。"libraries" プロパティーが指定されていません。	デフォルトの SQL スキーマは、ユーザー・プロファイル名です。	デフォルトの SQL スキーマはありません。ジョブのライブラリー・リストが検索されます。
2. URL にデフォルト SQL スキーマが指定されています。"libraries" プロパティーが指定されていません。	デフォルトの SQL スキーマは、URL で指定されているスキーマです。	デフォルトの SQL スキーマは、URL で指定されているスキーマです。 SQL ステートメントの修飾なしの名前を解決するために、ライブラリー・リストが検索されることはありません。
3. URL にデフォルト SQL スキーマが指定されていません。"libraries" プロパティーが指定されています。	デフォルトの SQL スキーマは、プロパティーで指定されている最初のライブラリーです。	デフォルトの SQL スキーマはありません。プロパティーで指定されるすべてのライブラリーが検索されます。
4. URL にデフォルト SQL スキーマが指定されています。"libraries" プロパティーが指定されています。	デフォルトの SQL スキーマは、URL で指定されているスキーマです。プロパティーは無視されます。	デフォルトの SQL スキーマは、URL で指定されているスキーマです。 SQL ステートメントの修飾なしの名前を解決するために、ライブラリー・リストが検索されることはありません。
5. URL にデフォルト SQL スキーマが指定されていません。"libraries" プロパティーが指定されており、リスト内のライブラリーが無効です。	デフォルトの SQL スキーマは、プロパティーで指定されている最初のライブラリーです。	デフォルトの SQL スキーマはありません。リスト内のライブラリーの 1 つが見つからないため、ライブラリー・リストを変更できません。このため、ジョブのライブラリー・リストが使用されます。

シナリオ	SQL ネーミング	システム・ネーミング
6. URL にデフォルト SQL スキーマが指定されていません。 "libraries" プロパティーが指定されており、リスト内の 2 番目のライブラリーにファイルがあります。	デフォルトの SQL スキーマは、プロパティーで指定されている最初のライブラリーです。残りのライブラリーは無視されます。	すべてのライブラリーが存在する場合、デフォルトの SQL スキーマはありません。リスト上のすべてのライブラリーが検索され、リストはジョブのライブラリー・リストを置換します。 リスト上のライブラリーの 1 つが存在しない場合、ジョブのライブラリー・リストは変更されません。
7. URL にデフォルト SQL スキーマが指定されていません。 "libraries" プロパティーが指定されており、リストはコンマで始まっています。	デフォルトの SQL スキーマは、ユーザー・プロファイルです。	デフォルトの SQL スキーマはありません。リスト上のすべてのライブラリーが検索され、リストはジョブのライブラリー・リストを置換します。
8. URL にデフォルト SQL スキーマが指定されていません。 "libraries" プロパティーが指定されており、リストは *LIBL で始まっています。	デフォルトの SQL スキーマは、ユーザー・プロファイルです。	デフォルトの SQL スキーマはありません。リスト上のすべてのライブラリーが検索され、指定されたライブラリーはリストの終わりに追加されます。
9. URL にデフォルト SQL スキーマが指定されていません。 "libraries" プロパティーが指定されており、リストは *LIBL で終わっています。	デフォルトの SQL スキーマは、プロパティーで指定されている最初のライブラリーです。残りのライブラリーは無視されます。	デフォルトの SQL スキーマはありません。リスト上のすべてのライブラリーが検索され、指定されたライブラリーはジョブのライブラリー・リストの先頭に追加されます。
10. URL に指定されているデフォルト SQL スキーマが無効です。	デフォルトの SQL スキーマはありません。ユーザー・プロファイルが使用されます。	デフォルトの SQL スキーマはありません。ジョブのライブラリー・リストが使用されます。

注: デフォルトの SQL スキーマが URL で指定されており、ライブラリー・プロパティーが使用されない場合、デフォルトの SQL スキーマが現行ライブラリー・リストの前に付加されます。

JDBC SQL タイプ

JDBC 仕様で説明されているすべての SQL タイプが DB2 for IBM i によってサポートされるわけではありません。

サポートされない SQL タイプ

SQL タイプがサポートされない場合、JDBC ドライバーは類似した SQL タイプを代わりに使用します。

次の表は、サポートされない SQL タイプと、JDBC ドライバーがそれぞれの代わりに使用する SQL タイプです。

サポートされない SQL タイプ	代わりに使用される SQL タイプ
BIT	SMALLINT
TINYINT	SMALLINT
LONGVARCHAR	VARCHAR
LONGVARBINARY	VARBINARY

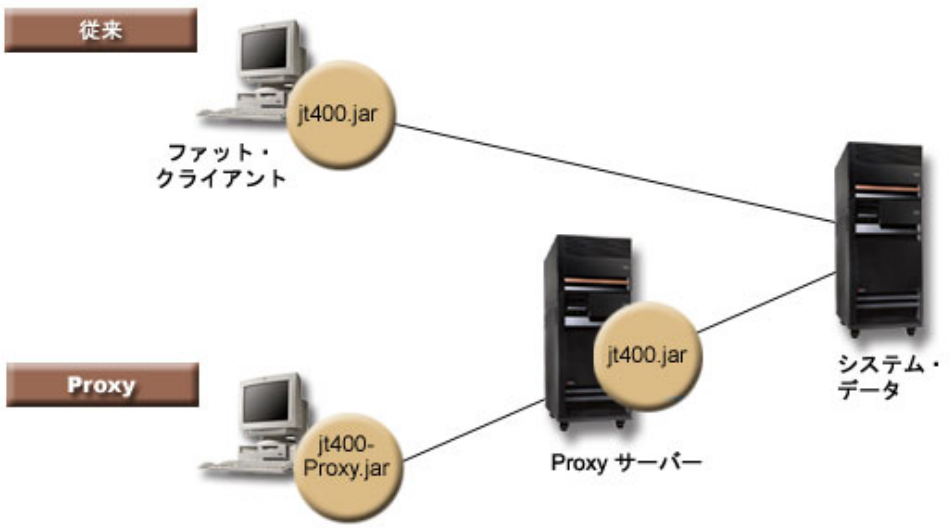
Proxy サポート

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 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 クラスのセットを構成します。 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 IBM iサーバーを使用できます。

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

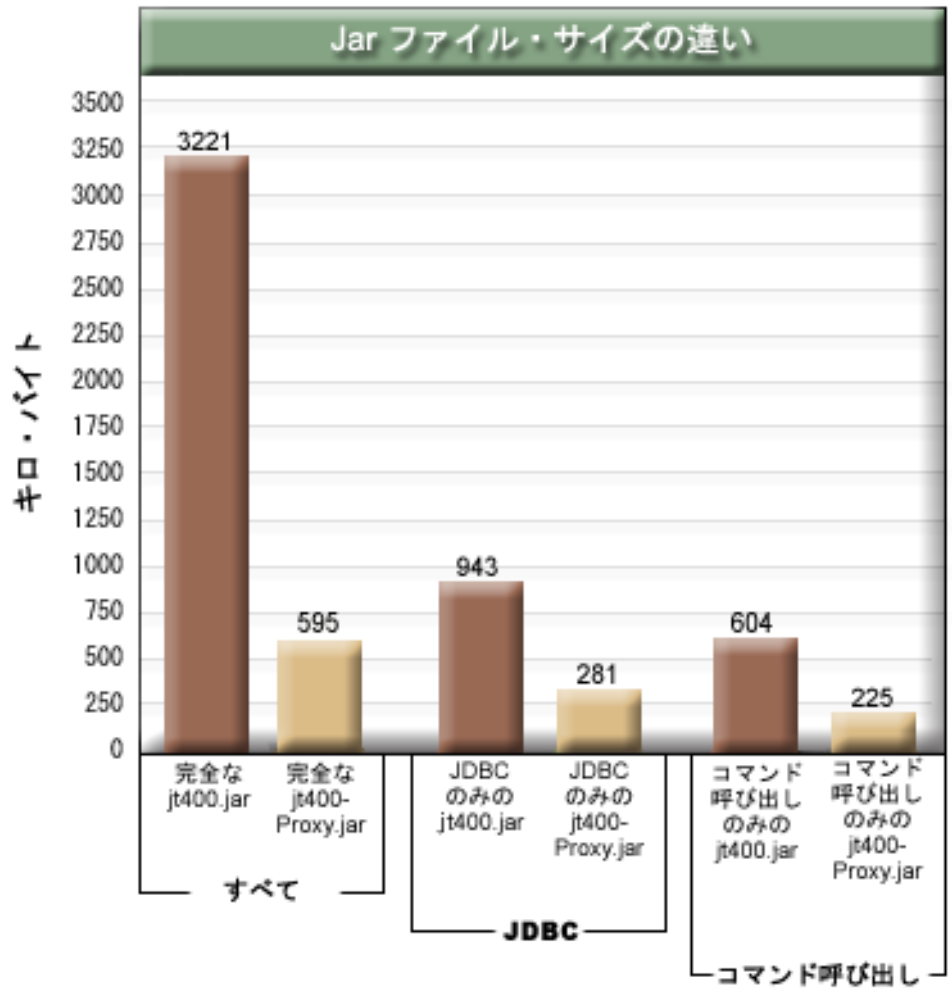
ると、パブリック・インターフェースはアプリケーションと一緒でなければなりませんが、要求を処理するためのクラスは別の JVM で実行できます。 Proxy サポートは、パブリック・インターフェースを変更しません。同じプログラムを、IBM Toolbox for Java の Proxy バージョンと、標準のバージョンで実行できます。

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。トンネルの欠点は、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、ソケット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。

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 つの例が以下に示されています。

- 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。

例: Proxy サポートを使用する Java アプリケーションの実行

以下の例では、 Proxy サポートを使用して Java アプリケーションを実行する手順を示します。

1. Proxy サーバーとして機能するマシンを選択します。 Proxy サーバー・マシン上の Java 環境および CLASSPATH には、 `jt400.jar` ファイルが含まれます。このマシンは、システムへ接続可能でなければなりません。
2. 以下のように入力して、Proxy サーバーをこのマシン上で開始します。 `java com.ibm.as400.access.ProxyServer -verbose` verbose を指定することにより、クライアントの接続および切断時にモニターできます。
3. クライアントとして機能するマシンを選択します。クライアント・マシン上の Java 環境および CLASSPATH には、 `jt400Proxy.jar` ファイルおよびご使用のアプリケーション・クラスが含まれます。このマシンは Proxy サーバーに接続可能でなければなりませんが、システムへの接続は必要ありません。
4. `com.ibm.as400.access.AS400.proxyServer` システム・プロパティーの値が Proxy サーバーの名前になるように正しく設定して、アプリケーションを実行します。これを行う簡単な方法は、ほとんどの Java 仮想マシンを呼び出すときに、-D オプションを指定することです。 `java -Dcom.ibm.as400.access.AS400.proxyServer=psMachineName YourApplication`
5. アプリケーションを実行すれば、 (ステップ 2 で verbose を設定した場合) アプリケーションが少なくとも 1 つの接続を、 Proxy サーバーに対して確立したことを確認できます。

例: Proxy サポートを使用する Java アプレットの実行

以下の例では、 Proxy サポートを使用して Java アプレットを実行する手順を示します。

1. Proxy サーバーとして機能するマシンを選択します。アプレットがネットワーク接続を開始できるのは、最初にダウンロード元となったマシンに対してだけです。そのため、Proxy サーバーを HTTP サーバーと同じマシン上で実行することが最善です。 Proxy サーバー・マシン上の Java 環境および CLASSPATH には、 jt400.jar ファイルが含まれます。
2. 以下のように入力して、Proxy サーバーをこのマシン上で開始します。 `java com.ibm.as400.access.ProxyServer -verbose` verbose を指定することにより、クライアントの接続および切断時にモニターできるようになります。
3. 実行する前にアプレット・コードをダウンロードしなければならないので、コードのサイズを最も小さくすることをお勧めします。 AS400ToolboxJarMaker は、アプレットが使用するコンポーネントのコードだけを組み込むことにより、 jt400Proxy.jar をかなり小さくすることができます。たとえば、アプレットが JDBC だけを使用する場合、以下のコマンドを実行して、 jt400Proxy.jar ファイルに最少の量のコードだけが組み込まれるようにしてください。

```
java utilities.AS400ToolboxJarMaker -source jt400Proxy.jar -destination jt400ProxySmall.jar
                                    -component JDBC
```

4. アプレットでは、 com.ibm.as400.access.AS400.proxyServer システム・プロパティーの値が Proxy サーバーの名前になるように正しく設定しなければなりません。アプレットに対してこれを行う簡単な方法は、コンパイル済みプロパティー・クラスを使用することです (例)。このクラスをコンパイルして、生成された Properties.class ファイルを com/ibm/as400/access ディレクトリー (ご使用の HTML ファイルを取り出したものと同じパス) に配置してください。たとえば、HTML ファイルが /mystuff/HelloWorld.html である場合、 Properties.class は /mystuff/com/ibm/as400/access に置かれます。
5. jt400ProxySmall.jar を、 HTML ファイルと同じディレクトリー (ステップ 4 での /mystuff/) に入れます。
6. HTML ファイル内で、以下のようにアプレットを参照してください。

```
<APPLET archive="jt400Proxy.jar, Properties.class" code="YourApplet.class"
        width=300 height=100> </APPLET>
```

例: Tunneling Proxy サポートを使用する Java アプリケーションの実行

以下の例では、 Tunneling Proxy サポートを使用して Java アプリケーションを実行する手順を示します。

1. Proxy サーバーを実行したい HTTP サーバーを選択してから、それをサーブレット com.ibm.as400.access.TunnelProxyServer (jt400.jar の中) を実行するように構成します。**注:** HTTP サーバーが、データを含むシステム、またはアプリケーションが使用するリソースに確実に接続するようにしてください。これは、サーブレットがそのシステムに接続して要求を実行するためです。
2. クライアントとして作動するマシンを選択して、クライアント・マシン上の CLASSPATH が、必ず jt400Proxy.jar ファイルとアプリケーション・クラスを設定してください。クライアントは HTTP サーバーに接続可能でなければなりませんが、システムへの接続は必要ありません。
3. com.ibm.as400.access.AS400.proxyServer プロパティーの値が、 URL 形式の HTTP サーバーの名前になるように設定します。
4. com.ibm.as400.access.AS400.proxyServer プロパティーの値が、 URL 形式の HTTP サーバーの名前になるように設定して、アプリケーションを実行します。これを行う簡単な方法は、ほとんどの JVM で -D オプションを指定することです。

```
java -Dcom.ibm.as400.access.AS400.proxyServer=http://psMachineName YourApplication
```

注: Proxy クライアント・コードは、 "servlet" とサーブレット名をサーバー名に連結することによって、正しいサーブレット URL を作成します。この例では、http://psMachineName を http://psMachineName/servlet/TunnelProxyServer に変換します。

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、J2SE、バージョン 1.4 およびこれ以降のバージョンに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。

以前に sslight を使用したことがある場合は、JSSE へマイグレーションする必要があります。現在の IBM i 5.4 では、JSSE が唯一のサポートされるパッケージであり、sslight はもはや出荷されていません。

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、IBM i 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 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、IBM Toolbox for Java 2 Micro Edition 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。

Toolbox ME 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 (jt400Micro.jar) は、いわゆる JTOpen という、Toolbox for Java のオープン・ソース・バージョンに含まれています。 JTOpen に組み込まれている ToolboxME 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 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 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。

- サポートされているバージョンの Java Standard Edition
- 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 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 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 クラス。 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 IBM i 要件

ToolboxME のダウンロードとセットアップ

JTOpen に入っている IBM Toolbox for Java 2 Micro Edition (jt400Micro.jar) は、別途ダウンロードする必要があります。

ToolboxME 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 の Web サイト  からダウンロードできます。

ToolboxME 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。
- IBM i ホスト・サーバーが、ターゲットのデータを収容している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。

詳細は、以下のページを参照してください。

10 ページの『IBM Toolbox for Java を ワークステーションにインストールする』

9 ページの『IBM Toolbox for Java をシステムにインストールする』

ToolboxME を使用する上で重要な概念

IBM Toolbox for Java 2 Micro Edition 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 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)

この構成は、より大規模なデバイス構成で必要とされる機能を提供するために、最小限必要な API と Java 仮想マシンに必要な能力を定義します。 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  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® Everyplace Micro Environment の一部)
- Sun K Virtual Machine (KVM) (CLDC の一部)
- 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 クラス

com.ibm.as400.micro パッケージは、Tier0 装置からサーバーのデータおよびリソースにアクセスするためのアプリケーションの作成に必要なクラスを提供します。

注: IBM Toolbox for Java 2 Micro Edition クラスを使用するには、別途 ToolboxME コンポーネントをダウンロードしてセットアップする必要があります。

ToolboxME には以下のクラスがあります。

com.ibm.as400.micro パッケージ

384 ページの『Tier0 デバイス』

MEServer クラス

IBM Toolbox for Java MEServer クラスを使用して、 IBM Toolbox for Java 2 Micro Edition 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満たします。 MEServer は IBM 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 クラスを使うには、 ToolboxME 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 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。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。

関連情報

MEServer Javadoc

384 ページの『Tier0 デバイス』

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。 IBM Toolbox for Java 2 Micro Edition AS400 クラスを使用して、Tier0 デバイスからシステムにサインオンします。

注: ToolboxME クラスを使うには、 ToolboxME 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 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しません。

例: AS400 クラスを使用する

以下の例は、AS400 クラスを使用して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();
```

AS400 Javadoc

26 ページの『AS400 クラス』

IBM Toolbox for Java AS400 クラスは、サーバー上のサーバー・ジョブへのソケット接続のセットおよびサーバーに対するサインオン動作を管理します。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。

384 ページの『Tier0 デバイス』

『CommandCall クラス - Micro パッケージ』

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 IBM i コマンドを呼び出します。

CommandCall クラス - Micro パッケージ

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 IBM i コマンドを呼び出します。

注: IBM Toolbox for Java 2 Micro Edition クラスを使用するには、別途 ToolboxME コンポーネントを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。

例: CommandCall を使用する

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();
```

関連資料

34 ページの『CommandCall クラス』
CommandCall クラスを使用すると、Java プログラムが非対話式の IBM i コマンドを呼び出せるようになります。

関連情報

Micro パッケージ CommandCall Javadoc

384 ページの『Tier0 デバイス』

DataQueue クラス - Micro パッケージ

DataQueue クラスを使用して、 Tier0 デバイスが IBM i データ待ち行列での読み取り/書き込みを行えるようにします。 micro パッケージの DataQueue クラス (com.ibm.as400.micro.DataQueue]) は、 access パッケージの DataQueue クラス (com.ibm.as400.access.DataQueue) で使用可能な機能の変更済みサブセットを提供します。

注: IBM Toolbox for Java 2 Micro Edition クラスを使用するには、別途 ToolboxME コンポーネントを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。

- ストリングとしてのエントリーの読み取りまたは書き込みを行う
- バイトの配列としてのエントリーの読み取りまたは書き込みを行う

エントリーの読み取りまたは書き込みを行うには、データ待ち行列が常駐する サーバーの名前、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。

例: DataQueue を使用してデータ待ち行列への読み取りおよび書き込みを行う

以下の例は、DataQueue クラスを使用して IBM i データ待ち行列で項目の読み取り/書き込みを行う方法を示します。

```
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();
```

関連資料

48 ページの『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。

384 ページの『ToolboxME を使用する上で重要な概念』
IBM Toolbox for Java 2 Micro Edition 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。

関連情報

DataQueue Javadoc

384 ページの『Tier0 デバイス』

Micro パッケージ - ProgramCall クラス

ProgramCall クラスを使用し、 Tier0 デバイスを使用可能にして IBM i 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。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。

注: IBM Toolbox for Java 2 Micro Edition クラスを使用するには、別途 ToolboxME コンポーネントをダウンロードしてセットアップする必要があります。詳しくは、 7 ページの『ToolboxME の要件』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サーバー
- 413 ページの『プログラム呼び出しマークアップ言語』 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 ProgramCall Javadoc を参照してください。PCML の詳細については、 413 ページの『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。

例: ProgramCall を使用する

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();
```

ProgramCall Javadoc (micro パッケージ)

166 ページの『ProgramCall クラス』

IBM Toolbox for Java ProgramCall クラスでは、Java プログラムが IBM i 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 IBM i プログラムから戻されたデータが入ります。 IBM i 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 IBM i 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。

384 ページの『Tier0 デバイス』

JdbcMe クラス

IBM Toolbox for Java 2 Micro Edition 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。

この後の項では、データへのアクセスとその使用を取り上げ、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 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

ToolboxME 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。

384 ページの『Tier0 デバイス』

ToolboxME 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、 IBM 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: IBM Toolbox for Java 2 Micro Edition クラスを使用するには、別途 ToolboxME コンポーネントを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 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];
```

注: 前の構文例は 2 行になっているため、それを簡単に参照して印刷することができます。通常、URL は 1 行で表示され、切れ目や余分のスペースはありません。

サーバー名を指定する必要があります。指定しないと JdbcMeDriver が例外をスローします。デフォルト SQL 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: デフォルト SQL 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。

この例では、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 SQL schema, port or
// properties are specified.
Connection c  = JdbcMeDriver.getConnection("jdbc:as400://mysystem.helloworld.com;meserver=myMeServer;"
                                           "auser",
                                           "apassword");
```

例: デフォルト SQL 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。

この例では、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 default SQL 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: サーバー・データベースへの接続

この例では、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: データベースからの切断

この例では、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();
```

JdbcMeConnection Javadoc

AS400JDBCConnection Javadoc

384 ページの『Tier0 デバイス』

JdbcMeDriver クラス:

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。 JdbcMeDriver クラスは、 IBM 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。

注: IBM Toolbox for Java 2 Micro Edition クラスを使用するには、別途 ToolboxME コンポーネントをダウンロードしてセットアップする必要があります。詳しくは、 383 ページの『ToolboxME のダウンロードとセットアップ』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、以下のようなコードを使用します。

```
Connection c = JdbcMeDriver.getConnection(
   "jdbc:as400://mysystem.myworld.com;meserver=myMeSrvr;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。

関連資料

『ResultSet』
IBM Toolbox for Java 2 Micro Edition 結果セット・クラスは、
JdbcMeLiveResultSet、JdbcMeOfflineResultSet、JdbcMeResultSetMetaData です。

関連情報

JdbcMeDriver Javadoc

384 ページの『Tier0 デバイス』

ResultSet:

IBM Toolbox for Java 2 Micro Edition 結果セット・クラスは、
JdbcMeLiveResultSet、JdbcMeOfflineResultSet、JdbcMeResultSetMetaData です。

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 クラスを使うには、 ToolboxME 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 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、 IBM 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。使用可能な両方向スクロール・カーソル位置は次のとおりです。

- absolute
- first
- 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 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、Javadoc の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、 IBM 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。デバイスに置かれるデータは、すでにあるか、 JdbcMeStatement.executeToOfflineData() メソッドを呼び出してデバイスに置

いてある可能性があります。 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、 IBM 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 JdbcMeResultSetMetaData を使用して、 JdbcMeLiveResultSet または JdbcMeOfflineResultSet 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();
```

関連情報

JDBCMeLiveResultSet Javadoc

AS400JDBCResultSet Javadoc

JDBCMeResultSetMetaData Javadoc

AS400JDBCResultSetMetaData Javadoc

JDBCMeOfflineResultSet Javadoc

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。

注: IBM Toolbox for Java 2 Micro Edition クラスを使用するには、別途 ToolboxME コンポーネントを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 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
- 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、以下の例を参照してください。

755 ページの『例: ToolboxME、MIDP、および IBM Toolbox for Java』

関連資料

384 ページの『ToolboxME を使用する上で重要な概念』

IBM Toolbox for Java 2 Micro Edition 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。

関連情報

JdbcMeOfflineData Javadoc

JdbcMeStatement クラス:

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。
JdbcMeStatement クラスは、 IBM 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。

注: IBM Toolbox for Java 2 Micro Edition クラスを使用するには、別途 ToolboxME コンポーネントを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 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。

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();
```

関連情報

JdbcMeStatement Javadoc

AS400JDBCStatement Javadoc

ToolboxME プログラムを作成して実行する

以下の情報を参考にすれば、IBM Toolbox for Java 2 Micro Edition 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 の作業例および独自の ToolboxME 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 の例の作成を開始する前に、ご使用の環境が ToolboxME の要件を満たしていることを確認してください。

ToolboxME の例を作成する

Tier0 デバイス用に ToolboxME 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 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は、必ず、選択した Java 仮想マシンまたは MIDP をデバイスにプリインストールしてください。

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 の概念を参照してください。

ToolboxME の例を実行する

Tier0 デバイスで ToolboxME 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 の例: JdbcDemo.java

作業用の IBM Toolbox for Java 2 Micro Edition プログラムとしてこの例を作成するには、以下の .java ファイルをテキストまたは Java エディターにコピーして、いくつかの変更を行い、その後コンパイルします。

ソース・コードをコピーするには、マウスを使用して以下の Java コードをすべて選択し、それから右マウス・ボタン・クリックし、「**コピー**」を選択します。ご使用のエディターにそのコードを貼り付けるには、エディターで空の文書を作成し、その空の文書で右マウス・ボタン・クリックして、「**貼り付け**」を選択します。 JdbcDemo.java という名前で新しい文書が保管されたことを確認してください。

.java ファイルの作成後、 プログラム例を作成して実行する際の指示に戻ってください。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////////////////////////
//
// ToolboxME example. This program demonstrates how your wireless
// device can connect to the server and use JDBC to perform work on a
// remote database.
//
////////////////////////////////////////////////////////////////////////////////

import java.sql.*;               // SQL Interfaces provided by JdbcMe
import com.ibm.as400.micro.*;    // JdbcMe implementation
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.microedition.io.*;  // Part of the CLDC specification
import de.kawt.*;                // Part of the CLDC specification

class DemoConstants
{
    // These constants are actually used mainly by the demo
```

```
    // for the JDBC driver. The Jdbc and JDBC  application
    // creator IDs ( http://www.palmos.com/dev )
    // are reserved at palm computing.
    public static final int demoAppID    = 0x4a444243; // JDBC
    // Make the dbCreator something else so that the
    // user can actually see the Palm DB seperately from
    // the JdbcDemo application.
    public static final int dbCreator    = 0x4a444231; // JDB1
    public static final int dbType       = 0x4a444231; // JDB1
}

/**
 * Little configuration dialog box to display the
 * current connections/statements, the
 * URL being used, user id and password
 */
class ConfigurationDialog extends Dialog implements ActionListener
{
    TextField         data;
    ConfigurationDialog(Frame w)
    {
        super(w, "Configuration");

        // Show/Modify current URL connection
        data = new TextField(JdbcDemo.mainFrame.jdbcPanel.url);
        add("Center", data);

        // Ok button.
        Panel panel = new Panel();
        Button button = new Button("Ok");
        button.addActionListener(this);
        panel.add(button);
        add("South", panel);
        pack();
    }

    public void actionPerformed(ActionEvent e)
    {
        JdbcDemo.mainFrame.jdbcPanel.url = data.getText();
        data = null;
        setVisible(false);
    }
}

/**
 * Little configuration dialog box to display the
 * current connections/statements, the
 * URL being used, user id and password
 */
class MultiChoiceDialog extends Dialog implements ActionListener
{
    Choice            task;
    ActionListener    theListener;
    MultiChoiceDialog(Frame w, String title, String prompt, String choices[], ActionListener it)
    {
        super(w, title);
        theListener = it;

        // Show/Modify current URL connection
        Label txt = new Label(prompt);
        add("West", txt);
        task = new Choice();
        for (int i=0; i<choices.length; ++i)
        {
            task.add(choices[i]);
        }
        task.select(0);
```

```
        add("Center", task);

        // Ok button.
        Panel panel = new Panel();
        Button button = new Button("Ok");
        button.addActionListener(this);
        panel.add(button);
        button = new Button("Cancel");
        button.addActionListener(this);
        panel.add(button);
        add("South", panel);
        pack();
    }

    /**
     *  Determine the action performed.
     **/
    public void actionPerformed(ActionEvent e)
    {
        int choice = task.getSelectedIndex();
        setVisible(false);
        if (e.getActionCommand().equals("Ok"))
        {
            if (theListener != null)
            {
                ActionEvent ev = new ActionEvent(this,
                                                 ActionEvent.ACTION_PERFORMED,
                                                 task.getItem(choice));
                theListener.actionPerformed(ev);
            }
            task = null;
        }
        else
        {
            // No-op
        }
    }
}

/**
 * The JdbcPanel is the main panel of the application.
 * It displays the current connection and statement
 * at the top.
 * A text field for entering SQL statements next.
 * A Results field for displaying each column of data
 * or results.
 * An task list and a 'go' button so that different
 * tasks can be tried.
 */
class JdbcPanel extends Panel implements ActionListener
{
    public final static int TASK_EXIT         =  0;
    public final static int TASK_NEW          =  1;
    public final static int TASK_CLOSE        =  2;
    public final static int TASK_EXECUTE      =  3;
    public final static int TASK_PREV         =  4;
    public final static int TASK_NEXT         =  5;
    public final static int TASK_CONFIG       =  6;
    public final static int TASK_TOPALMDB     =  7;
    public final static int TASK_FROMPALMDB   =  8;
    public final static int TASK_SETAUTOCOMMIT=  9;
    public final static int TASK_SETISOLATION = 10;
    public final static int TASK_COMMIT       = 11;
    public final static int TASK_ROLLBACK     = 12;


    // JDBC objects.
```

```
java.sql.Connection  connObject  = null;
Statement            stmtObject  = null;
ResultSet            rs          = null;
ResultSetMetaData    rsmd        = null;

String        lastErr     = null;
String        url         = null;
Label         connection  = null;
Label         statement   = null;
TextField     sql         = null;
List          data        = null;
final Choice  task;

/**
 * Build the GUI.
 */
public JdbcPanel()
{
    // The JDBC URL
    // Make sure to edit the following line so that it correctly specifies the
    // the MEServer and the server to which you want to connect.
    url = "jdbc:as400://mySystem;user=myUidl;password=myPwd;meserver=myMEServer;";

    Panel    p1left = new Panel();
    p1left.setLayout(new BorderLayout());
    connection = new Label("None");
    p1left.add("West", new Label("Conn:"));
    p1left.add("Center", connection);

    Panel    p1right = new Panel();
    p1right.setLayout(new BorderLayout());
    statement = new Label("None");
    p1right.add("West", new Label("Stmt:"));
    p1right.add("Center", statement);

    Panel    p1 = new Panel();
    p1.setLayout(new GridLayout(1,2));
    p1.add(p1left);
    p1.add(p1right);

    Panel    p2 = new Panel();
    p2.setLayout(new BorderLayout());
    p2.add("North", new Label("Sql:"));
    sql = new TextField(25);
    sql.setText("select * from QIWS.QCUSTCDT"); // Default query
    p2.add("Center", sql);

    Panel    p3 = new Panel();
    p3.setLayout(new BorderLayout());
    data = new List();
    data.add("No Results");
    p3.add("North", new Label("Results:"));
    p3.add("Center", data);

    Panel    p4 = new Panel();

    task = new Choice();
    task.add("Exit");             // TASK_EXIT
    task.add("New");              // TASK_NEW
    task.add("Close");            // TASK_CLOSE
    task.add("Execute");          // TASK_EXECUTE
    task.add("Prev");             // TASK_PREV
    task.add("Next");             // TASK_NEXT
    task.add("Config");           // TASK_CONFIGURE
    task.add("RS to PalmDB");     // TASK_TOPALMDB
    task.add("Query PalmDB");     // TASK_FROMPALMDB
    task.add("Set AutoCommit");   // TASK_SETAUTOCOMMIT
```

```
    task.add("Set Isolation");    // TASK_SETISOLATION
    task.add("Commit");           // TASK_COMMIT
    task.add("Rollback");         // TASK_ROLLBACK
    task.select(TASK_EXECUTE);    // Start off here.
    p4.add("West", task);

    Button b = new Button("Go");
    b.addActionListener(this);
    p4.add("East", b);

    Panel prest = new Panel();
    prest.setLayout(new BorderLayout());
    prest.add("North", p2);
    prest.add("Center", p3);
    Panel pall = new Panel();
    pall.setLayout(new BorderLayout());
    pall.add("North", p1);
    pall.add("Center", prest);

    setLayout(new BorderLayout());
    add("Center", pall);
    add("South", p4);
}

/**
 * Do a task based on whichever task is
 * currently selected in the task list.
 */
public void actionPerformed(ActionEvent e)
{
    if (e.getSource() instanceof MultiChoiceDialog)
    {
        String   cmd = e.getActionCommand();
        processExtendedCommand(cmd);
        return;
    }

    switch (task.getSelectedIndex())
    {
    case TASK_EXIT:
        System.exit(0);
        break;
    case TASK_NEW:
        JdbcPanel.this.goNewItems();
        break;
    case TASK_PREV:
        JdbcPanel.this.goPrevRow();
        break;
    case TASK_NEXT:
        JdbcPanel.this.goNextRow();
        break;
    case TASK_EXECUTE:
        if (connObject == null || stmtObject == null)
            JdbcPanel.this.goNewItems();

        JdbcPanel.this.goExecute();
        break;
    case TASK_CONFIG:
        JdbcPanel.this.goConfigure();
        break;
    case TASK_CLOSE:
        JdbcPanel.this.goClose();
        break;
    case TASK_TOPALMDB:
        if (connObject == null || stmtObject == null)
            JdbcPanel.this.goNewItems();
```

```
            JdbcPanel.this.goResultsToPalmDB();
            break;
        case TASK_FROMPALMDB:
            JdbcPanel.this.goQueryFromPalmDB();
            break;
        case TASK_SETAUTOCOMMIT:
            JdbcPanel.this.goSetAutocommit();
            break;
        case TASK_SETISOLATION:
            JdbcPanel.this.goSetIsolation();
            break;
        case TASK_COMMIT:
            JdbcPanel.this.goTransact(true);
            break;
        case TASK_ROLLBACK:
            JdbcPanel.this.goTransact(false);
            break;

        default :
        {
                Dialog dialog = new FeedbackDialog(JdbcDemo.mainFrame, "Error", "Task not implemented");
                dialog.show();
                dialog = null;
            }
        }
    }

    public void processExtendedCommand(String cmd)
    {
        try
        {
            if (cmd.equals("true"))
            {
                connObject.setAutoCommit(true);
                return;
            }
            if (cmd.equals("false"))
            {
                connObject.setAutoCommit(false);
                return;
            }
            if (cmd.equals("read uncommitted"))
            {
                connObject.setTransactionIsolation(java.sql.Connection.TRANSACTION_READ_UNCOMMITTED);
                return;
            }
            if (cmd.equals("read committed"))
            {
                connObject.setTransactionIsolation(java.sql.Connection.TRANSACTION_READ_COMMITTED);
                return;
            }
            if (cmd.equals("repeatable read"))
            {
                connObject.setTransactionIsolation(java.sql.Connection.TRANSACTION_REPEATABLE_READ);
                return;
            }
            if (cmd.equals("serializable"))
            {
                connObject.setTransactionIsolation(java.sql.Connection.TRANSACTION_SERIALIZABLE);
                return;
            }
            throw new IllegalArgumentException("Invalid command: " + cmd);
        }
        catch (Exception e)
        {
            JdbcDemo.mainFrame.exceptionFeedback(e);
            return;
```

```
    }
}

/**
 * Perform commit or rollback processing.
 */
public void goTransact(boolean commit)
{
    if (connObject == null)
    {
        FeedbackDialog dialog = new FeedbackDialog(JdbcDemo.mainFrame,
                                                   "Skip",
                                                   "Connection not allocated");
        dialog.show();
        dialog = null;
        return;
    }
    try
    {
        if (commit)
            connObject.commit();
        else
            connObject.rollback();
    }
    catch (Exception e)
    {
        JdbcDemo.mainFrame.exceptionFeedback(e);
    }
}

/**
 * Prompt the user for setting the autocommit value
 * Real work handled by the actionPerformed method
 * calling processExtendedCommand().
 */
public void goSetAutocommit()
{
    if (connObject == null)
    {
        FeedbackDialog dialog = new FeedbackDialog(JdbcDemo.mainFrame,
                                                   "Skip",
                                                   "Connection not allocated");
        dialog.show();
        dialog = null;
        return;
    }
    try
    {
        String currentValue;
        if (connObject.getAutoCommit())
            currentValue = "Now: true";
        else
            currentValue = "Now: false";

        Dialog dialog = new MultiChoiceDialog(JdbcDemo.mainFrame,
                                              "Set Autocommit",
                                              currentValue,
                                              new String[]{ "true", "false"},
                                              this);
        dialog.show();
        dialog = null;
    }
    catch (Exception e)
    {
        JdbcDemo.mainFrame.exceptionFeedback(e);
    }
}
```

```
/**
 * Prompt the user for setting the isolation level,
 * real work handled by the actionPerformed() method
 * calling processExtendedCommand().
 */
public void goSetIsolation()
{
    if (connObject == null)
    {
        FeedbackDialog dialog = new FeedbackDialog(JdbcDemo.mainFrame,
                                                   "Skip",
                                                   "Connection not allocated");
        dialog.show();
        dialog = null;
        return;
    }
    try
    {
        int   level = connObject.getTransactionIsolation();
        String currentLevel;
        switch (level)
        {
        case java.sql.Connection.TRANSACTION_READ_UNCOMMITTED:
            currentLevel = "Now: read uncommitted";
            break;
        case java.sql.Connection.TRANSACTION_READ_COMMITTED:
            currentLevel = "Now: read committed";
            break;
        case java.sql.Connection.TRANSACTION_REPEATABLE_READ:
            currentLevel = "Now: repeatable read";
            break;
        case java.sql.Connection.TRANSACTION_SERIALIZABLE:
            currentLevel = "Now: serializable";
            break;
        default : {
                currentLevel = "error";
            }
        }
        Dialog dialog = new MultiChoiceDialog(JdbcDemo.mainFrame,
                                              "Set Isolation Level",
                                              currentLevel,
                                              new String[]{ "read uncommitted",
                                                            "read committed",
                                                            "repeatable read",
                                                            "serializable"},
                                              this);
        dialog.show();
        dialog = null;
    }
    catch (Exception e)
    {
        JdbcDemo.mainFrame.exceptionFeedback(e);
    }
}

/**
 * Create a new connection or statement.
 * Only one connection and statement is currently
 * supported.
 */
public void goNewItems()
{
    if (connObject != null || stmtObject != null)
    {
        FeedbackDialog dialog = new FeedbackDialog(JdbcDemo.mainFrame,
                                                   "Skip",
```

```
                                                "Conn/Stmt already allocated");
            dialog.show();
            dialog = null;
        }
        if (connObject == null)
        {
            try
            {
                connObject = DriverManager.getConnection(url);
                //connection.setText(Integer.toString(((JdbcMeConnection)connObject).getId()));
                connection.repaint();
            }
            catch (Exception e)
            {
                JdbcDemo.mainFrame.exceptionFeedback(e);
                return;
            }
        }
        if (stmtObject == null)
        {
            try
            {
                try
                {
                    stmtObject = connObject.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                                            ResultSet.CONCUR_READ_ONLY);
                }
                catch (Exception e)
                {
                    // Try again...  DB2 NT version 6.1 doesn't support
                    // Scollable result sets, so we'll assume other
                    // JDBC 2.0 databases don't either. We'll attempt
                    // to create another.
                    try
                    {
                        stmtObject = connObject.createStatement();
                    }
                    catch (Exception ex)
                    {
                        // If the second try failed, rethrow the
                        // first exception. Its probably
                        // a more meaninful error.
                        throw e;
                    }
                    FeedbackDialog dialog = new FeedbackDialog(JdbcDemo.mainFrame,
                                                               "2nd try worked",
                                                               "Non-scrollable result set");
                    dialog.show();
                    dialog = null;
                }

                statement.repaint();
            }
            catch (Exception e)
            {
                JdbcDemo.mainFrame.exceptionFeedback(e);
                return;
            }
        }
    }


    /**
     *  Close the statement and connection.
     **/
    public void goClose()
    {
```

```
    // Close the statement.
    if (stmtObject != null)
    {
        if (rs != null)
        {
            try
            {
                rs.close();
            }
            catch (Exception e)
            {
            }
            rs = null;
            rsmd = null;
        }
        try
        {
            stmtObject.close();
        }
        catch (Exception e)
        {
        }
        stmtObject = null;
        statement.setText("None");
        statement.repaint();
    }

    // Clost the connection.
    if (connObject != null)
    {
        try
        {
            connObject.close();
        }
        catch (Exception e)
        {
        }
        connObject = null;
        connection.setText("None");
        connection.repaint();
    }
    data.removeAll();
    data.add("No Results");
    data.repaint();
    sql.repaint();
    return;
}

/**
 * display the configuration dialog.
 **/
public void goConfigure()
{
    // Note there is no model dialog support in KAWT, this only
    // works because the data to be changed (url) is set before
    // this dialog is used, and the user cannot access the
    // main frame while this is up on the palm (i.e. all dialogs
    // in Kawt are modal).
    ConfigurationDialog dialog = new ConfigurationDialog(JdbcDemo.mainFrame);
    dialog.show();
    dialog = null;
}

/**
 *  Execute the specified query.
 **/
public void goExecute()
```

```
{
    // Get the currently selected statement.
    try
    {
        if (rs != null)
            rs.close();

        rs = null;
        rsmd = null;
        boolean results = stmtObject.execute(sql.getText());
        if (results)
        {
            rs = stmtObject.getResultSet();
            rsmd = rs.getMetaData();
            // Show the first row
            goNextRow();
        }
        else
        {
            data.removeAll();
            data.add(stmtObject.getUpdateCount() + " rows updated");
            data.repaint();
        }
    }
    catch (Exception e)
    {
        JdbcDemo.mainFrame.exceptionFeedback(e);
    }
}


/**
 *  Move to the next row in the result set.
 **/
public void goNextRow()
{
    try
    {
        if (rs == null || rsmd == null)
            return;

        int   count = rsmd.getColumnCount();
        int   i;
        data.removeAll();
        if (!rs.next())
            data.add("End of data");
        else
        {
            for (i=1; i>=count; ++i)
            {
                data.add(rs.getString(i));
            }
        }
        data.repaint();
    }
    catch (Exception e)
    {
        JdbcDemo.mainFrame.exceptionFeedback(e);
    }
}


/**
 *  Move to the previous row in the result set.
 **/
public void goPrevRow()
{
```

```
    try
    {
        if (rs == null || rsmd == null)
            return;

        int   count = rsmd.getColumnCount();
        int   i;
        data.removeAll();
        if (!rs.previous())
            data.add("Start of data");
        else
        {
            for (i=1; i<=count; ++i)
            {
                data.add(rs.getString(i));
            }
        }
        data.repaint();
    }
    catch (Exception e)
    {
        JdbcDemo.mainFrame.exceptionFeedback(e);
    }
}

/**
 *  Perform a query and store the results in the local devices database
 **/
public void goResultsToPalmDB()
{
    try
    {
        if (stmtObject == null)
        {
            FeedbackDialog dialog = new FeedbackDialog(JdbcDemo.mainFrame, "Skip", "No Statement");
            dialog.show();
            dialog = null;
            return;
        }

        boolean results =
           ((JdbcMeStatement)stmtObject).executeToOfflineData(sql.getText(),
                                                              "JdbcResultSet",
                                                              DemoConstants.dbCreator,
                                                              DemoConstants.dbType);
        if (!results)
        {
            FeedbackDialog dialog = new FeedbackDialog(JdbcDemo.mainFrame, "No Data", "Not a query");
            dialog.show();
            dialog = null;
            return;
        }
        data.removeAll();
        data.add("Updated Palm DB 'JdbcResultSet'");
        data.repaint();
    }
    catch (Exception e)
    {
        JdbcDemo.mainFrame.exceptionFeedback(e);
    }
}


/**
 *  Perform a query from the database that resides on the palm device.
 **/
public void goQueryFromPalmDB()
```

```
    {
        try
        {
            if (rs != null)
            {
                rs.close();
                rs = null;
            }
            rs = new JdbcMeOfflineResultSet ("JdbcResultSet",
                                             DemoConstants.dbCreator,
                                             DemoConstants.dbType);
            rsmd = rs.getMetaData();
            // If we want to debug some output, this
            // method can be used to dump the contents
            // of the PalmDB represented by the result set
            // (Uses System.out so its mostly useful in
            // the Palm emulator when debugging your
            // applications.
            // ((JdbcMeOfflineResultSet)rs).dumpDB(true);

            // show the first row.
            goNextRow();
        }
        catch (SQLException e)
        {
            JdbcDemo.mainFrame.exceptionFeedback(e);
        }
    }
}

public class JdbcDemo extends Frame
{
    /** An ActionListener that ends the application. Only
     * one is required, and can be reused
     */
    private static ActionListener    exitActionListener = null;
    /**
     * The main application in this process.
     */
    static         JdbcDemo mainFrame = null;

    JdbcPanel      jdbcPanel = null;

    public static ActionListener getExitActionListener()
    {
        if (exitActionListener == null)
        {
            exitActionListener = new ActionListener()
            {
                public void actionPerformed(ActionEvent e)
                {
                    System.exit(0);
                }
            };
        }
        return exitActionListener;
    }

    /**
     *  Demo Constructor
     **/
    public JdbcDemo()
    {
        super("Jdbc Demo");
        setLayout(new BorderLayout());

        jdbcPanel = new JdbcPanel();
```

```
        add("Center", jdbcPanel);

        addWindowListener(new WindowAdapter()
         {
             public void windowClosing(WindowEvent e)
             {
                 System.exit(0);
             }
         });
        setSize(200,300);
        pack();
    }

    public void exceptionFeedback(Exception e)
    {
        Dialog dialog = new FeedbackDialog(JdbcDemo.mainFrame, e);
        dialog.show();
        dialog = null;
    }

    /**
     * Main method.
     **/
    public static void main(String args[])
    {
        try
        {
            mainFrame = new JdbcDemo();
            mainFrame.show();
            mainFrame.jdbcPanel.goConfigure();
        }
        catch (Exception e)
        {
            System.exit(1);
        }
    }
}
```

ToolboxME for iSeries の作業例

以下の IBM Toolbox for Java 2 Micro Edition の作業例は、Mobile Information Device Profile (MIDP) で ToolboxME 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・ファイル例すべてをダウンロードします。

746 ページの『例: ToolboxME、MIDP、および JDBC』

755 ページの『例: ToolboxME、MIDP、および IBM Toolbox for Java』

『ToolboxME 例のダウンロード』

ToolboxME アプリケーションを構築する方法についての詳細は、 398 ページの『ToolboxME プログラムを作成して実行する』 を参照してください。

MIDP の詳細については、 385 ページの『Mobile Information Device Profile (MIDP)』を参照してください。

ToolboxME 例のダウンロード

IBM Toolbox for Java 2 Micro Edition の例を作業用のワイヤレス・アプリケーションに構築するには、すべてのソース・ファイルおよび追加の指示が必要になります。

この例をダウンロードして構築するには、以下の手順に従ってください。

1. ソース・ファイルをダウンロードする (microsamples.zip)
2. microsamples.zip をこの目的のために作成したディレクトリーに解凍する
3. ワイヤレス・アプリケーションの例を構築するのに役立つ、 398 ページの『ToolboxME プログラムを作成して実行する』に述べられている指示に従う

ソースをコンパイルし、 Tier0 デバイス用の実行可能ファイルを構築する前に、以下の詳細を参照してください。

- 7 ページの『ToolboxME の要件』
- 383 ページの『ToolboxME のダウンロードとセットアップ』

Extensible Markup Language のコンポーネント

IBM Toolbox for Java には、 XML パーサーを含むいくつかの Extensible Markup Language (XML) コンポーネントが組み込まれています。

XML コンポーネントを使用すると、以下のさまざまなタスクを簡単に実行することができます。

- グラフィカル・ユーザー・インターフェースの作成
- システム上のプログラムの呼び出しおよび結果の取得
- システム上のデータ・フォーマットの指定

プログラム呼び出しマークアップ言語

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。

注: PCML の使用に関心があるか、あるいはすでに使用している場合、Extensible Program Call Markup Language (XPCML) の使用を考慮してください。XPCML は、XML スキーマのサポートを提供することによって、PCML の機能性および使用可能度を向上させます。 XPCML を含め、IBM Toolbox for Java XML コンポーネントについての詳細は、Extensible Markup Language コンポーネント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。

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クライアント 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。

PCML を使用するための要件

PCML 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、PCML 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。

注: PCML ファイルをあらかじめ直列化してある場合、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。

関連資料

8 ページの『IBM Toolbox for Java のためのワークステーション要件』
ワークステーションが以下の要件を満たしていることを確認してください。

454 ページの『XML パーサーおよび XSLT プロセッサー』
IBM Toolbox for Java のパッケージや機能の中には、実行時に、 CLASSPATH 環境変数に拡張可能マークアップ言語 (XML) パーサーまたは Extensible Stylesheet Language Transformations (XSLT) プロセッサーが設定されている必要のあるものもあります。

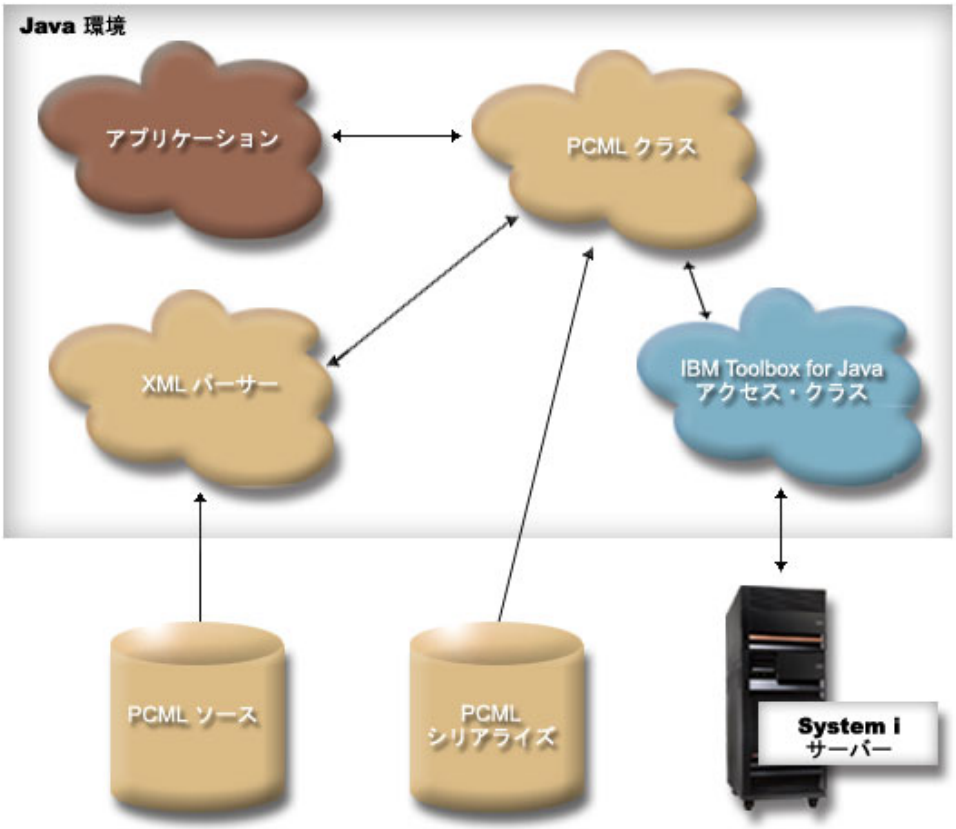
PCML を使用して IBM i プログラム呼び出しを作成する

PCML を使用して IBM i 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。

設計プロセスによっては、1 つ以上の PCML ソース・ファイルを作成し、そこに、Java アプリケーションによって呼び出される IBM i プログラムへの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。

次に、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 IBM i 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 オブジェクトを作成すると、 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。 IBM Toolbox for Java で XML パーサーを使用する方法について詳しくは XML パーサーおよび XSLT プロセッサーを参照してください。

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、 IBM i 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 クラスを直列化することができます。そうすると、ProgramCallDocument は直列化ファイルを使って作成されます。この場合、実行時に XML パーサーは使用されません。 直列化 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 を使用して、PCML ソース・ファイルを見つけます。

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 ソースは検索します。 .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");
```

直列化 PCML ファイルの使用

実行時のパフォーマンスを向上させるため、直列化 PCML ファイルを使用することができます。直列化 PCML ファイルには、PCML を表す直列化 Java オブジェクトが含まれます。直列化オブジェクトは、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。

直列化 PCML ファイルを使用すると、 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直列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、直列化すると myDoc.pcml.ser という名前のファイルになります。

PCML ソース・ファイルと直列化 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 という名前の直列化 PCML ファイルをまず検出しようとします。直列化 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 IBM i 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、IBM i 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のソースが与えられる場合、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 では使用すべきではありません。

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 IBM i 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。(以下の例は、幅の制限のために変更されています)

  ```
  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。(以下の例は、幅の制限のために変更されています)

```
/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.lengthOfInfo"
 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、 IBM i 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。

属性	値	説明
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統合ファイル・システム・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しなければなりません。 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。

属性	値	説明
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エントリー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 IBM i 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 IBM i 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>** 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*	この要素が存在する最高水準の IBM i のバージョンを指定します。 IBM i 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、IBM i 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、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 桁以上の数字です。 "v" の値は、1 から 255 までででなければなりません。 "r" および "m" の値は、0 から 255 までででなければなりません。

属性	値	説明
minvrm=	*version-string*	この要素が存在する最低水準の IBM i のバージョンを指定します。 IBM i 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、IBM i 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、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 桁以上の数字です。 "v" の値は、1 から 255 までででなければなりません。 "r" および "m" の値は、0 から 255 までででなければなりません。

属性	値	説明
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>** 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、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。

属性	値	説明
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* 属性は通常、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祖先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祖先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祖先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祖先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。

属性	値	説明
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>** 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** 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。 この属性を省略すると、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"** と指定してください。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、その属性がオプションであることを示します。オプション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。 *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。	使われているデータのタイプ (文字、整数、パック、ゾーン、浮動小数点、バイト、または構造体) を示します。 データ型が異なれば、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。

属性	値	説明
bidistringtype=	*DEFAULT*。DEFAULT は、非両方向データ (LTR) のデフォルトのストリング型です。 *ST4*。*ST4* は、ストリング型 4 です。 *ST5*。*ST5* は、 ストリング型 5 です。 *ST6*。*ST6* は、ストリング型 6 です。 *ST7*。*ST7* は、 ストリング型 7 です。 *ST8*。*ST8* は、ストリング型 8です。 *ST9*。*ST9* は、 ストリング型 9 です。 *ST10*。*ST10* は、ストリング型 10 です。 *ST11*。*ST11* は、ストリング型 11 です。	両方向ストリング型を **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型は、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)。 ストリング型は、 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>** 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*。*onebyte* は、各文字のサイズを指定します。 *twobyte*。*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。

属性	値	説明
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>** 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*。*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>** 要素 (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>** 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、データ型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。

属性	値	説明
maxvrm=	*version-string*	この要素が存在する最高水準の IBM i のバージョンを指定します。 IBM i 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、IBM i 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、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 桁以上の数字です。 "v" の値は、1 から 255 までででなければなりません。 "r" および "m" の値は、0 から 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 IBM i のバージョンを指定します。 IBM i 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、IBM i 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、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 桁以上の数字です。 "v" の値は、1 から 255 までででなければなりません。 "r" および "m" の値は、0 から 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。

属性	値	説明
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>** 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。

属性	値	説明
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祖先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祖先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祖先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祖先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。

属性	値	説明
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>** 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、IBM i・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*。*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*。*value* は整数値を表します。**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型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。

属性	値	説明
trim=	*right*。*right* は、末尾空白文字を切り取ることを意味する、デフォルトの動作です。 *left*。*left* は、先頭空白文字を切り取ることを意味します。 *both*。*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*。*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:

データ型が異なれば、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型がリストされています。

データ型	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。

データ型	長さ	精度
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* のどちらか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>
```

Graphical Toolbox および PDML

Graphical Toolbox は、Java 形式のカスタム・ユーザー・インターフェース・パネルの作成を可能にする 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、または System i Navigatorのプラグインにそのパネルを取り込むことができます。システムや他のソース (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入れることができます。

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レンダリングを JavaFoundation Classes を使って行うことにより、パネルを表示します。

注: PDML を使用するには、 Java ランタイム環境のバージョン 1.4 以降を実行する必要があります。

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 によるスケルトンを生成できます。必要なヘルプ・トピックを指定すると、必要に応じたそのヘルプ・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・タグがヘルプのスケルトン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・コードのスケルトン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

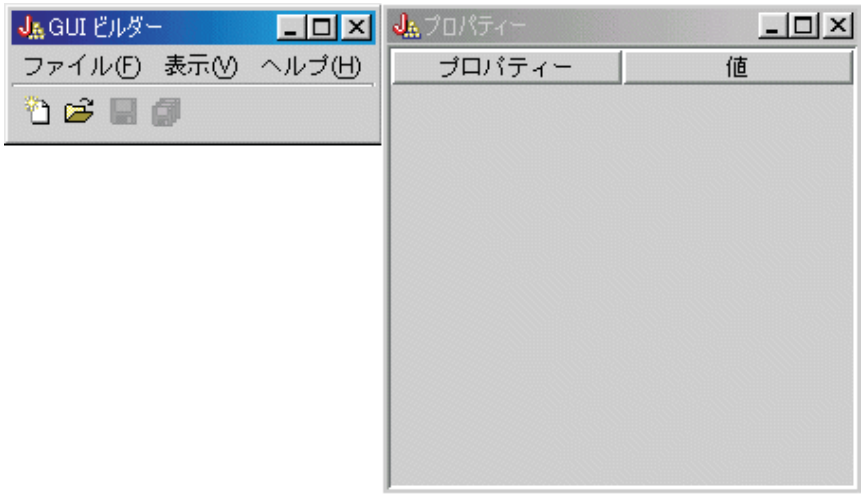
Graphical Toolbox の実行時環境では、イベント処理、ユーザー・データ妥当性検査、およびパネルの要素間の共通タイプの対話がサポートされます。プラットフォーム上での正しいユーザー・インターフェースの外観は、その使用しているオペレーティング・システムに基づいて自動的に設定されます。 GUI ビルダーによって外観をいろいろと切り替えれば、さまざまなプラットフォーム上でパネルが表示される様子を評価できます。

Graphical Toolbox 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、最初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

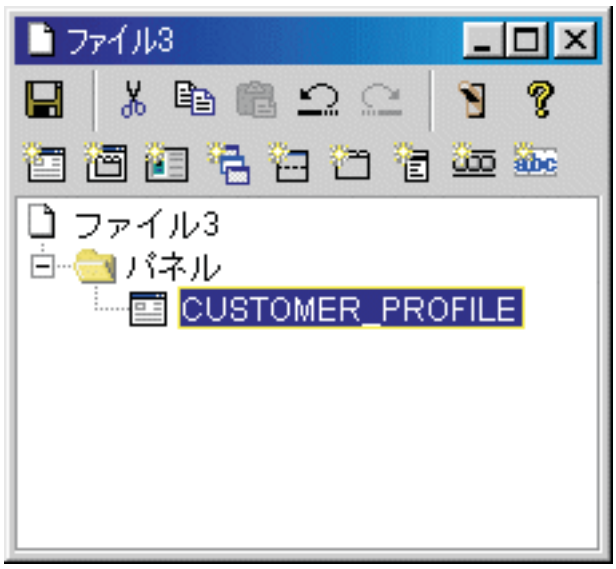
初めて GUI ビルダーを起動する際、図 1 の 2 つのウィンドウが表示されます。

図 1: GUI ビルダーのウィンドウ



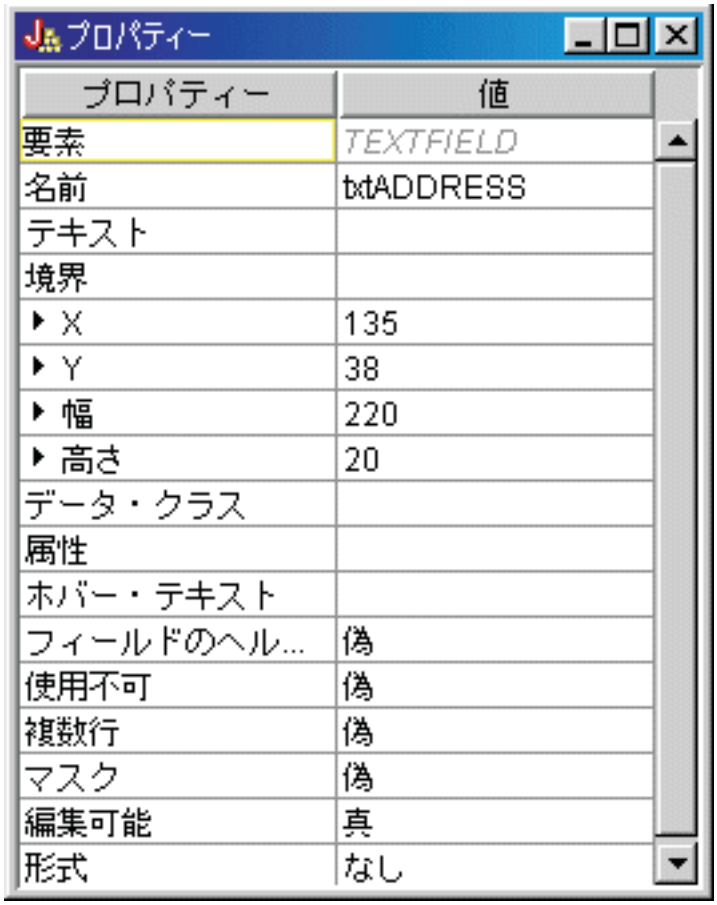
「ファイル・ビルダー」ウィンドウは、PDML ファイルを作成および編集するために使用します。

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コントロールの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。

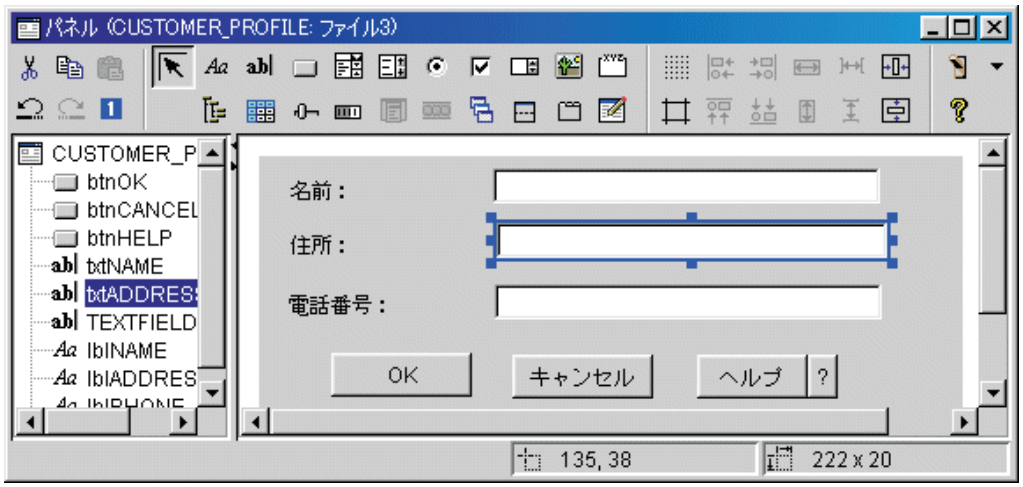
図 3: 「プロパティー」ウィンドウ



「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

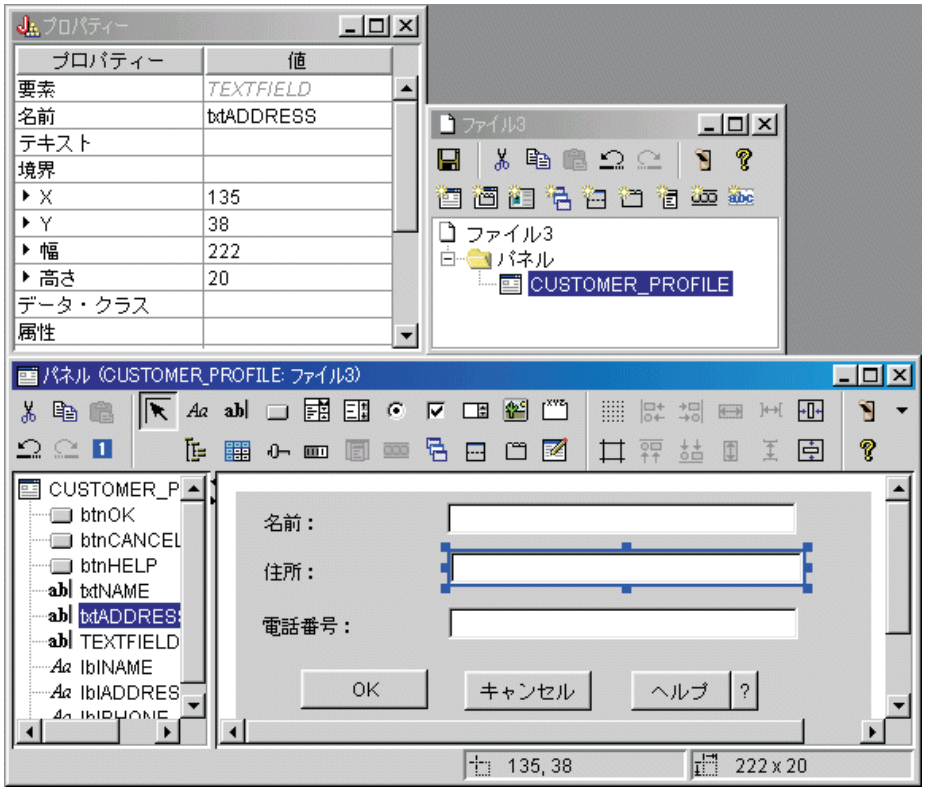
それを配置します。またツールバーには、コントロールのグループを位置合わせしたり、パネルをプレビューしたり、GUI ビルダー機能のオンライン・ヘルプを要求したりする機能も備えられています。それぞれのアイコンで行えることについての説明は、GUI ビルダーのパネル・ビルダー・ツールバーを参照してください。

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。

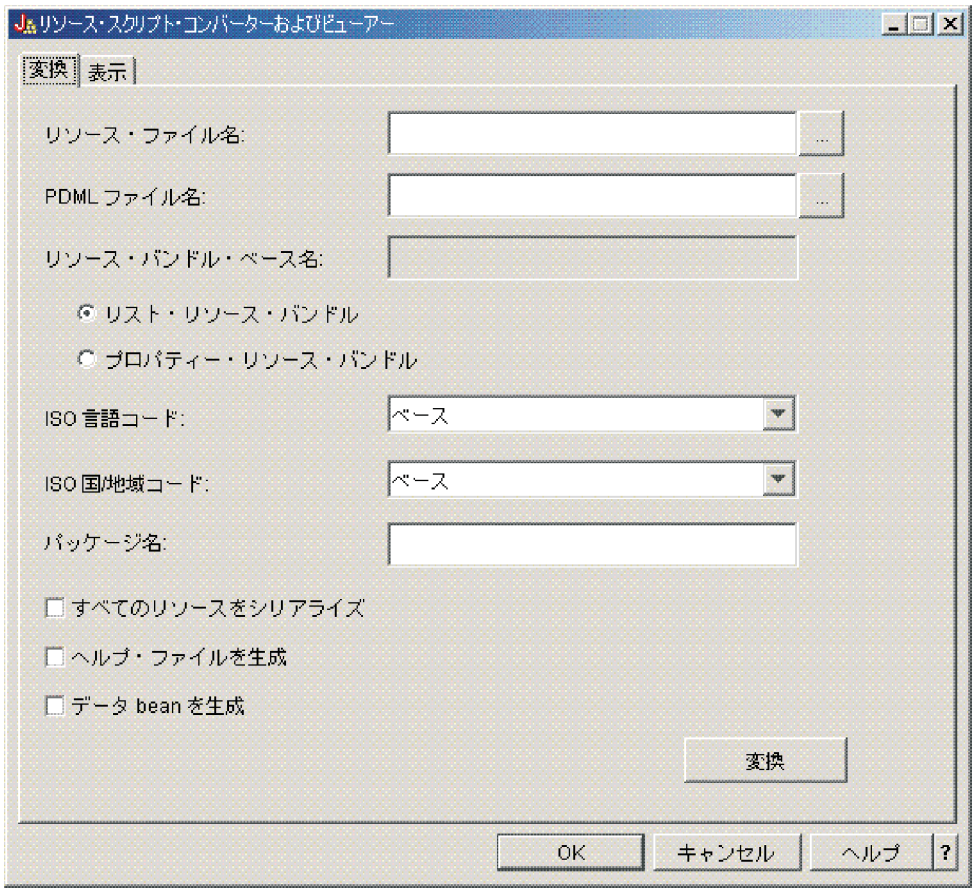
図 5: GUI ビルダーの各ウィンドウが協働する方法の例



リソース・スクリプト・コンバーター

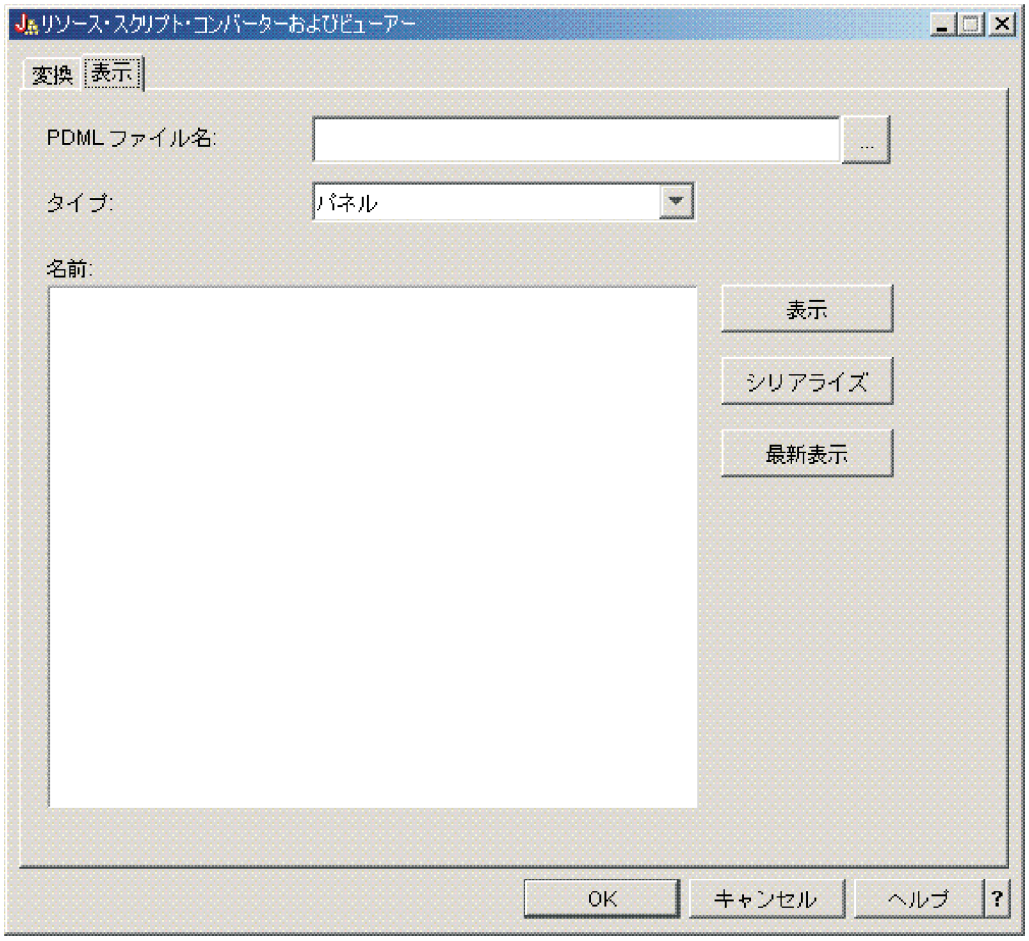
リソース・スクリプト・コンバーターは、 2 つのペインから成るタブ形式のダイアログです。「**変換**」 ペインには、 PDML に変換する Microsoft または VisualAge for Windows の RC ファイルの名前を指定します。目的の PDML ファイル、および関連する Java リソース・バンドル (パネルの変換済みストリングを含む) に名前を指定することができます。また、パネルのオンライン・ヘルプのスケルトンを生成するよう要求したり、パネルのデータがあるオブジェクトの Java ソース・コードのスケルトンを生成したり、パネル定義を直列化して実行時間のパフォーマンスを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



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、IBM i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) データ型の実用的なサブセットを表します。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、IBM 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 指向の言語です。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。

関連情報

データ記述仕様 (DDS)

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。詳しくは、 454 ページの『XML パーサーおよび XSLT プロセッサー』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直列化してある場合、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。

関連資料

8 ページの『IBM Toolbox for Java のためのワークステーション要件』
ワークステーションが以下の要件を満たしていることを確認してください。

例: IBM 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 IBM 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ときには、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ときには、RFML ファイルを変更すればよく、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、IBM 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"));
```

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 IBM 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

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・タイプとデータ型の値を取得および設定する
- 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・ファイルを直列化する

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 IBM Toolbox for Java クラスと共に使用する

RecordFormatDocument クラスを以下の IBM 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 IBM 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 のデータ型のみを扱うため。
- 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、IBM i 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。

関連情報

RecordFormatDocument Javadoc

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。

IBM 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・ファイルの例が入っています。

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 ファイル内での要素を宣言して定義します。

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 (DTD):

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:

RFML 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。

data タグの属性を以下にリストしてあります。ブラケット [] で囲まれた属性は、その属性がオプションであることを示します。オプション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。	使われているデータのタイプ (文字、整数、パック、ゾーン、浮動小数点、バイト、または構造体) を示します。 データ型が異なれば、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。

属性	値	説明
bidistringtype=	*DEFAULT*。DEFAULT は、非両方向データ (LTR) のデフォルトのストリング型です。 *ST4*。*ST4* は、ストリング型 4 です。 *ST5*。*ST5* は、 ストリング型 5 です。 *ST6*。*ST6* は、ストリング型 6 です。 *ST7*。*ST7* は、 ストリング型 7 です。 *ST8*。*ST8* は、ストリング型 8 です。 *ST9*。*ST9* は、 ストリング型 9 です。 *ST10*。*ST10* は、ストリング型 10 です。 *ST11*。*ST11* は、ストリング型 11 です。	両方向ストリング型を **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型は、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>** 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。

属性	値	説明
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>** 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>** 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、データ型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。
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型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。

関連情報

BidiStringType Javadoc

RFML rfml タグ:

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。

rfml タグの属性を以下にリストしてあります。ブラケット [] で囲まれた属性は、その属性がオプションであることを示します。オプション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。 V5R3 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:

RFML 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。

recordformat タグの属性を以下にリストしてあります。ブラケット [] で囲まれた属性は、その属性がオプションであることを示します。オプション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	説明
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:

RFML 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。

struct タグの属性を以下にリストしてあります。ブラケット [] で囲まれた属性は、その属性がオプションであることを示します。オプション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	説明
name=	*name*	**<struct>** 要素の名前を指定します。

XML パーサーおよび XSLT プロセッサー

IBM Toolbox for Java のパッケージや機能の中には、実行時に、 CLASSPATH 環境変数に拡張可能マークアップ言語 (XML) パーサーまたは Extensible Stylesheet Language Transformations (XSLT) プロセッサーが設定されている必要のあるものもあります。

使用するパーサーおよびプロセッサーを判別するには、以下の情報を参照してください。

どの IBM Toolbox for Java パッケージおよび機能に XML パーサーまたは XSLT プロセッサーが必要かについては、以下のページを参照してください。

11 ページの『JAR ファイル』

XML パーサー

パッケージまたは機能に XML パーサーが必要な場合は、実行時に CLASSPATH 中に XML パーサーを含めなければなりません。 XML パーサーは以下の要件を満たしていなければなりません。

- JAXP に準拠している。
- クラス org.apache.xerces.parsers.SAXParser が拡張されている。
- 完全なスキーマの妥当性検査をサポートしている。

注: ユーザーが XPCML の使用を試みている場合は、パーサーには完全なスキーマの妥当性検査のサポートのみ必要です。 PCML のみ使用する場合は、完全なスキーマの妥当性検査は必要ありません。

Java 2 Software Developer Kit (J2SDK) バージョン 1.4 には、適切な XML パーサーが組み込まれています。旧バージョンの J2SDK を使用している場合は、以下のいずれかのメソッドを使用して、適切な XML パーサーにアクセスしてください。

- x4j400.jar (Apache 社製のIBM バージョンの Xerces XML パーサー) を使用する。
- Apache 社の Web サイト から、 Xerces XML パーサーをダウンロードする。
- システム上の `/QIBM/ProdData/OS400/xml/lib` ディレクトリー中で互換性のある XML パーサーを使用する。

注: `/QIBM/ProdData/OS400/xml/lib` 中にある最新バージョンのパーサーを使用することを考慮してください。例えば、xmlapis11.jar および xerces411.jar は、両方とも完全な妥当性検査をサポートしているパーサーです。

ご使用のサーバー上でこれらのパーサーを使用するか、またはワークステーションにコピーできます。

注: XPCML の実行要件を満たしている XML パーサーは、 PCML および RFML を実行できます。 XPCML は直列化をサポートしていないことに注意してください。

XSLT プロセッサー

パッケージまたは機能に XSLT プロセッサーが必要な場合は、実行時に CLASSPATH 中に XSLT プロセッサーを含めなければなりません。 XSLT プロセッサーは以下の要件を満たしていなければなりません。

- JAXP に準拠している。
- クラス javax.xml.transform.Transformer を含む。

Java Software Developer Kit (J2SDK) バージョン 1.4 には、適切な XSLT プロセッサーが組み込まれています。旧バージョンの J2SDK を使用している場合は、以下のいずれかのメソッドを使用して、適切な XSLT プロセッサーにアクセスしてください。

- xslparser.jar (Apache 社製の IBM バージョンの Xalan XSLT プロセッサー) を使用する。
- Apache 社の Web サイト から、 Xalan XSLT プロセッサーをダウンロードする。
- システム上の `/QIBM/ProdData/OS400/xml/lib` ディレクトリー中で互換性のある XSLT プロセッサーを使用する。

ご使用のシステム上でこれらのプロセッサーを使用するか、またはワークステーションにコピーできます。

Extensible Program Call Markup Language

Extensible Program Call Markup Language (XPCML) は、 XML スキーマをサポートしており、プログラム呼び出しマークアップ言語 (PCML) の機能と使用可能度を拡張します。 XPCML は直列化をサポートしていないので、XPCML 文書は PCML とは違って直列化できません。

しかし、XPCML は、PCML に比べて複数の有効な拡張が行われています。

- プログラム・パラメーターの値を指定して渡す。
- サーバーへのプログラム呼び出しの結果を XPCML で取り出す。
- 既存の PCML 文書を同等の XPCML 文書に変換する。
- XPCML スキーマを拡張およびカスタマイズし、単純および複雑なエレメントおよび属性を新しく定義する。

XPCML についての詳細は、以下のページを参照してください。

PCML に対する XPCML の利点
PCML と比較した XPCML の有効な拡張に関する詳細が分かります。

要件
XPCML を使用するためのソフトウェア要件についてお読みください。

XPCML のスキーマおよび構文
XPCML スキーマが XPCML 構文を定義する方法と、使用可能なサーバー・データ型について参照します。特定のプログラミング上の必要に合うようにスキーマを拡張およびカスタマイズする方法が分かります。

XPCML の使用
XPCML の拡張の活用する方法について学びます。さまざまな種類のパラメーターをサーバー・プログラムに渡す方法や、戻されたパラメーター・データを検索する方法が含まれます。 XPCML コードを圧縮する方法 (コードの使いやすさや読みやすさが向上する) や、現行の PCML 使用可能アプリケーションで XPCML を使用する方法が分かります。

XPCML 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。 XML の使用についての詳細は、以下のページを参照してください。

Extensible Markup Language のコンポーネント

XML Toolkit

W3C Architecture domain: XML Schema 

PCML に対する XPCML の利点

Extensible Program Call Markup Language (XPCML) は、 PCML に比べて複数の有効な拡張が行われています。

- プログラム・パラメーターの値を指定して渡す。
- IBM i サーバーへのプログラム呼び出しの結果を XPCML で取り出す。
- 既存の PCML 文書を同等の XPCML 文書に変換する。
- XPCML スキーマを拡張およびカスタマイズし、単純および複雑なエレメントおよび属性を新しく定義する。

プログラム・パラメーターの値を指定して渡す。

XPCML は XML スキーマを使用して、プログラム・パラメーターのタイプを定義します。一方 PCML はデータ型定義 (DTD) を使用します。解析時に XML パーサーは、スキーマで定義されているとおりに、該当するパラメーターに対して、パラメーターとして入力されているデータ値を妥当性検査します。パラメーターのデータ型には、ストリング、整数、long などの多数のタイプがあります。このプログラム・パラメーターの値を指定して渡す機能により、 PCML が大幅に改善されています。 PCML では、パラメーターの値を検証できるのは、PCML 文書の解析後のみです。さらに、PCML でパラメーター値を検証するには、妥当性検査を実行するようにアプリケーションをコーディングする必要が頻出します。

プログラム呼び出しの結果を XPCML で取り出す

XPCML には、プログラム呼び出しの結果を XPCML として取り出す機能もあります。 PCML では、プログラム呼び出しの結果を取得するには、プログラムの呼び出し後に ProgramCallDocument クラスの getValue メソッドの 1 つを呼び出します。 XPCML では、getValue メソッドを使用できますが、XPCML に generateXPCML メソッドを呼び出させることもできます。こうすると、プログラム呼び出しの結果を XPCML として戻します。

既存の PCML 文書を XPCML 文書に変換する

ProgramCallDocument クラスの新しいメソッドである transformPCMLToXPCML を使用すると、既存の PCML 文書を同等の XPCML 文書に変換できます。こうすると、既存の IBM i プログラム呼び出し文書用に XPCML ソースを書き込まなくても、新しい XPCML 機能の利点を活用できます。

XPCML スキーマを拡張およびカスタマイズする

XPCML は、 XPCML スキーマで指定されているパラメーター・タイプを拡張する新しいパラメーター・タイプを定義できるという点で拡張できます。 XPCML を圧縮することにより、XPCML スキーマを拡張して新しいデータ型定義を作成し、 XPCML 文書の読み易さおよび使用可能度を単純化して改善できます。

XPCML を使用するための要件

Extensible Program Call Markup Language (XPCML) 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。

詳細は、以下のページを参照してください。

8 ページの『IBM 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』

さらに、XPCML の使用要件には、XML スキーマ・ファイル、XML パーサー、および Extensible Stylesheet Language Transformation (XSLT) プロセッサーの要件が含まれます。

XML スキーマ・ファイル

スキーマを含むファイルの場所が XPCML 文書に認識されていなければなりません。 IBM Toolbox for Java のデフォルト・スキーマは xpcml.xsd で、 jt400.jar ファイル中にあります。

デフォルトのスキーマを使用するには、xpcml.xsd を jt400.jar から抽出してから、このファイルを適切な場所に置いてください。以下の手順は、ワークステーション上で .xsd ファイルを抽出する方法の 1 つを示しています。

xpcml.xsd スキーマ・ファイルの抽出

- jt400.jar を含むディレクトリー中でコマンド行セッションを開始する。
- 以下のコマンドを使用して、.xsd ファイルを抽出する。

```
jar xvf  jt400.jar   com/ibm/as400/data/xpcml.xsd
```

注: jt400.jar を含むディレクトリーから上記のコマンドを実行しない場合は、 jt400.jar への完全修飾パスを指定できます。

デフォルトのスキーマ・ファイル (または任意のスキーマ・ファイル) は、どのディレクトリーにも置くことができます。要件は 1 つのみあり、それは <xpcml> タグ中の xsi:noNamespaceSchemaLocation 属性を使用して、スキーマ・ファイルの場所を指定することです。

スキーマの場所は、ファイル・パスまたは URL として指定できます。

注: 以下の例ではスキーマ・ファイルとして xpcml.xsd を使用していますが、 xpcml.xsd を拡張しているどのファイルでも指定できます。

- XPCML ファイルと同じディレクトリーを指定するには、 xsi:noNamespaceSchemaLocation='xpcml.xsd' を使用する。
- 完全修飾パスを指定するには、 xsi:noNamespaceSchemaLocation='c:¥myDir¥xpcml.xsd' を使用する。
- URL を指定するには、 xsi:noNamespaceSchemaLocation='http://myServer/xpcml.xsd' を使用する。

HTML バージョンの xpcml.xsd ファイルを参照するには、以下のページを参照してください。

460 ページの『xpcml.xsd スキーマ・ファイル』

XML パーサーおよび XSLT プロセッサー

実行時には、 CLASSPATH 環境変数中に XML パーサーおよび XSLT プロセッサーが組み込まれていなければなりません。詳細は、以下のページを参照してください。

454 ページの『XML パーサーおよび XSLT プロセッサー』

XPCML のスキーマおよび構文

XPCML ソース・ファイルという XPCML 文書には、システム上のプログラムの呼び出しを完全に定義したタグおよびデータが含まれています。

XPCML は文書タイプ定義 (DTD) の代わりに XML スキーマを使用するので、 PCML の場合には使用できない方法で XPCML を使用できます。

- 入力パラメーターの値をプログラムに XML エレメントとして渡す。
- 出力パラメーターの値をプログラムから XML エレメントとして受け取る。
- プログラムに渡された値を XML パーサーに自動的に妥当性検査させる。
- スキーマを拡張し、単純なエレメントや複雑なエレメントを新しく定義する。

XPCML のスキーマおよび構文についての詳細は、以下のページを参照してください。

XPCML ソースと PCML ソースとの比較
XPCML ソースと PCML ソースを比較した例を調べます。この例は、XPCML の方が機能が多く、ソース・データを読み書きしやすくなることを示しています。

XPCML スキーマ
XPCML スキーマ・ファイルについて調べ、 XPCML スキーマの使用法と拡張方法の詳細について学びます。

XPCML の構文
スキーマで XPCML の定義に使用される XPCML 構文エレメントのリストを検討します。

XPCML タグ属性
XPCML スキーマで定義されているエレメントごとに、さまざまな属性について説明します。

XPCML ソースと PCML ソースとの比較:

XPCML はいくつかの点で PCML と異なりますが、主要な違いの 1 つは、 XPCML を使用すると XPCML ソース・ファイル中に入力パラメーターの値を指定できることです。

PCML を使用する場合は、<data> タグの init 属性を使用して、 PCML ソース中にデータ・エレメントの初期値を指定します。しかし、PCML を使用して値を指定する場合には以下の制限があります。

- init 属性を使用して配列値を設定できない。
- init 値の妥当性検査は、PCML 文書の解析後のみ行われる。

PCML で配列値を指定するには、まず PCML 文書を読み取って解析してから、一連の ProgramCallDocument.setValue() 呼び出しを実行しなければなりません。

XPCML を使用すると、単一エレメントおよび配列の値の指定が簡単になります。

- XPCML ソース・ファイルのスカラーおよび配列エレメントの両方の値を指定する。
- 指定された配列値を解析時に妥当性検査する。

以下に XPCML と PCML の違いについて、単純な比較をしています。個々の例は、IBM i プログラムのプログラム呼び出しを定義しています。

例: IBM i プログラムの呼び出し

以下の例は、prog1 という IBM i プログラムを呼び出します。

XPCML ソース・コード

```
<?xml version="1.0" encoding="UTF-8"?>
<xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation='xpcml.xsd' version="4.0">
    <program name="prog1" path="/QSYS.LIB/W95LIB.LIB/PROG1.PGM">
      <parameterList>
         <stringParm name="parm1"  passDirection="in" length="10">Parm1</stringParm>
         <intParm    name="parm2"  passDirection="in">5</intParm>
         <shortParm  name="parm3"  passDirection="in">3</shortParm>
      </parameterList>
      </program>
</xpcml>
```

PCML ソース・コード

```
<pcml version="4.0">
  <program name="prog1" path="QSYS.LIB/MYLIB.LIB/PROG1.PGM">
     <data name="parm1"  type="char" usage="input" length="10" init="Parm1"/>
     <data name="parm2"  type="int"  usage="input" length="4"  init="5"/>
     <data name="parm3"  type="int"  usage="input" length="2"  precision="16" init="3"/>
    </program>
</pcml>
```

例: ストリング・パラメーターの配列を使用して IBM i プログラムを呼び出す

以下の例は、prog2 という IBM i プログラムを呼び出し、ストリング・パラメーターの配列として parm1 を定義します。以下の XPCML の機能に注意してください。

- 配列中の個々のエレメントの値を初期化する。
- 完全に妥当性検査する XML パーサーが検査できるエレメントの内容として入力値を指定する。

Java コードを作成せずにこの XPCML 機能を活用できます。

PCML は XPCML のパフォーマンスには及びません。 PCML は、配列中の個々のエレメントの値を初期化できません。 PCML は解析時に init 値を妥当性検査できません。 XPCML の機能と同等にするには、PCML 文書を読み取って解析してから、 Java アプリケーションをコーディングして個々の配列エレメントの値を設定する必要があります。コードを作成して、パラメーターを妥当性検査する必要もあります。

XPCML ソース・コード

```
<?xml version="1.0" encoding="UTF-8"?>
<xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="xpcml.xsd" version="4.0">

    <program name="prog2" path="/QSYS.LIB/W95LIB.LIB/PROG2.PGM">
      <parameterList>
         <arrayOfStringParm name="parm1" passDirection="in"
            length="10" count="3">
             <i>Parm1-First value</i>
             <i>Parm1-Second value</i>
             <i>Parm1-Third value</i>
         </arrayOfStringParm>
         <longParm name="parm2"  passDirection="in">5</longParm>
         <zonedDecimalParm name="parm3"  passDirection="in"
            totalDigits="5" fractionDigits="2">32.56</zonedDecimalParm>
      </parameterList>
      </program>
</xpcml>
```

PCML ソース・コード

```
<pcml version="4.0">
  <program name="prog2" path="QSYS.LIB/MYLIB.LIB/PROG2.PGM">
     <data name="parm1"  type="char"  usage="input" length="20" count="3"/>
```

```
        <data name="parm2"  type="int"   usage="input" length="8"  init="5"/>
        <data name="parm3"  type="zoned" usage="input" length="5"  precision="2" init="32.56"/>
       </program>
   </pcml>
```

xpcml.xsd スキーマ・ファイル:

表示や印刷を行いやすくするために、この HTML バージョンの xpcml.xsd の一部の行は折り返されて 2 行になっています。ソース xsd ファイル中の同じ行は 1 行になっています。

xpcml.xsd ファイルの使用法について詳しくは、XPCML を使用するための要件を参照してください。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
<?xml version="1.0" encoding="UTF-8"?>

<!--////////////////////////////////////////////////////////////////////////////
//
// JTOpen (IBM Toolbox for Java - OSS version)
//
// Filename: xpcml.xsd
//
// The source code contained herein is licensed under the IBM Public License
// Version 1.0, which has been approved by the Open Source Initiative.
// Copyright (C) 1997-2008 International Business Machines Corporation and
// others. All rights reserved.
//
////////////////////////////////////////////////////////////////////////////-->

<xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'>

<xs:annotation>
 <xs:documentation>
  Schema for xpcml (eXtended Program Call Markup Language).
 </xs:documentation>
</xs:annotation>

<xs:element name="xpcml">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="structOrProgram" minOccurs="1" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="version" use="required">
     <xs:simpleType>
       <xs:restriction base="xs:string">
         <xs:enumeration value="4.0"/>
       </xs:restriction>
     </xs:simpleType>
   </xs:attribute>
  </xs:complexType>

  <!-- Define key/keyref link between the name of a struct   -->
  <!-- and the struct attribute of a parameter field.        -->
  <xs:key name="StructKey">
    <xs:selector xpath="struct"/>
    <xs:field xpath="@name"/>
  </xs:key>
  <xs:keyref name="spRef" refer="StructKey">
     <xs:selector xpath="structParm" />
     <xs:field xpath="@struct" />
  </xs:keyref>
</xs:element>

<!-- Program tag and attributes -->
<xs:element name="program" substitutionGroup="structOrProgram">
  <xs:complexType>
```

```
    <xs:sequence>
      <xs:element ref="parameterList" minOccurs="1" maxOccurs="1"/>
         <!-- Used as a wrapper tag around the parameter list for the program. -->
    </xs:sequence>
    <!-- Name of the program to call. -->
    <xs:attribute name="name" type="string50" use="required" />
    <!-- Path to the program object. Default is to assume in library QSYS. -->
    <xs:attribute name="path" type="xs:string"/>
    <!-- Specifies the order in which parameters should be parsed. -->
    <!-- Value is a blank-separated list of parameter names. -->
    <xs:attribute name="parseOrder" type="xs:string"/>
    <!-- The entry point name within a service program. -->
    <xs:attribute name="entryPoint" type="xs:string"/>
    <!-- The type of value, if any, returned from a service program call. -->
    <xs:attribute name="returnValue" type="returnValueType"/>
    <!-- When calling a Java program and System i -->
    <!-- program is on same server -->
    <!-- and is thread-safe, set to true to call the
    <!-- System i program in same job -->
    <!-- and on same thread as the Java program. -->
    <xs:attribute name="threadSafe" type="xs:boolean"  />
    <!-- The CCSID of the entry point name within a service program. -->
    <xs:attribute name="epccsid" type="ccsidType"/>
  </xs:complexType>
</xs:element>

<!-- A parameter list is made up of one or more parameters. -->
<xs:element name="parameterList">
  <xs:complexType>
      <xs:group ref="programParameter" minOccurs="1" maxOccurs="unbounded"/>
   </xs:complexType>
</xs:element>

<!-- All the different kinds of program parameters that we understand. -->
<xs:group name="programParameter">
      <xs:choice>
 <xs:element ref="stringParmGroup"/>
 <xs:element ref="stringParmArrayGroup"/>
 <xs:element ref="intParmGroup"/>
 <xs:element ref="intParmArrayGroup"/>
        <xs:element ref="unsignedIntParmGroup"/>
        <xs:element ref="unsignedIntParmArrayGroup"/>
 <xs:element ref="shortParmGroup"/>
 <xs:element ref="shortParmArrayGroup"/>
 <xs:element ref="unsignedShortParmGroup"/>
 <xs:element ref="unsignedShortParmArrayGroup"/>
 <xs:element ref="longParmGroup"/>
 <xs:element ref="longParmArrayGroup"/>
 <xs:element ref="zonedDecimalParmGroup"/>
 <xs:element ref="zonedDecimalParmArrayGroup"/>
 <xs:element ref="packedDecimalParmGroup"/>
        <xs:element ref="packedDecimalParmArrayGroup"/>
 <xs:element ref="floatParmGroup"/>
 <xs:element ref="floatParmArrayGroup"/>
 <xs:element ref="doubleParmGroup"/>
 <xs:element ref="doubleParmArrayGroup"/>
 <xs:element ref="hexBinaryParmGroup"/>
 <xs:element ref="hexBinaryParmArrayGroup"/>
        <xs:element ref="structParmGroup"/>
        <xs:element ref="structParmArrayGroup"/>
        <xs:element ref="structArrayGroup"/>
        <xs:element ref="struct"/>
      </xs:choice>

</xs:group>

<!-- Abstract type for all data parameter types. -->
```

```
<xs:element name="stringParmGroup" type="stringParmType" abstract="true" />
<xs:element name="stringParmArrayGroup" type="stringParmArrayType" abstract="true" />
<xs:element name="intParmGroup" type="intParmType" abstract="true" />
<xs:element name="intParmArrayGroup" type="intParmArrayType" abstract="true" />
<xs:element name="unsignedIntParmGroup" type="unsignedIntParmType" abstract="true" />
<xs:element name="unsignedIntParmArrayGroup" type="unsignedIntParmArrayType" abstract="true" />
<xs:element name="shortParmGroup" type="shortParmType" abstract="true" />
<xs:element name="shortParmArrayGroup" type="shortParmArrayType" abstract="true" />
<xs:element name="unsignedShortParmGroup" type="unsignedShortParmType" abstract="true" />
<xs:element name="unsignedShortParmArrayGroup" type="unsignedShortParmArrayType" abstract="true" />
<xs:element name="longParmGroup" type="longParmType" abstract="true" />
<xs:element name="longParmArrayGroup" type="longParmArrayType" abstract="true" />
<xs:element name="zonedDecimalParmGroup" type="zonedDecimalParmType" abstract="true" />
<xs:element name="zonedDecimalParmArrayGroup" type="zonedDecimalParmArrayType" abstract="true" />
<xs:element name="packedDecimalParmGroup" type="packedDecimalParmType" abstract="true" />
<xs:element name="packedDecimalParmArrayGroup" type="packedDecimalParmArrayType" abstract="true" />
<xs:element name="floatParmGroup" type="floatParmType" abstract="true" />
<xs:element name="floatParmArrayGroup" type="floatParmArrayType" abstract="true" />
<xs:element name="doubleParmGroup" type="doubleParmType" abstract="true" />
<xs:element name="doubleParmArrayGroup" type="doubleParmArrayType" abstract="true" />
<xs:element name="hexBinaryParmGroup" type="hexBinaryParmType" abstract="true" />
<xs:element name="hexBinaryParmArrayGroup" type="hexBinaryParmArrayType" abstract="true" />
<xs:element name="structParmGroup" type="structParmType" abstract="true" />
<xs:element name="structParmArrayGroup" type="structParmArrayType" abstract="true" />
<xs:element name="structArrayGroup" type="structArrayType" abstract="true"
            substitutionGroup="structOrProgram" />

<!-- String parameter -->
   <xs:element name="stringParm" type="stringParmType" substitutionGroup="stringParmGroup"
               nillable="true"/>
     <xs:complexType name="stringParmType">
       <xs:simpleContent>
          <xs:extension base="stringFieldType">
            <xs:attributeGroup ref="commonParmAttrs"/>
          </xs:extension>
       </xs:simpleContent>
   </xs:complexType>

<!-- Array of string parameters -->
  <xs:element name="arrayOfStringParm" type="stringParmArrayType"
              substitutionGroup="stringParmArrayGroup" nillable="true" />
  <xs:complexType name="stringParmArrayType">
    <xs:sequence>
      <xs:element name="i" type="stringElementType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attributeGroup ref="commonParmAttrs"/>
    <xs:attributeGroup ref="commonFieldAttrs"/>
    <!-- The number of elements in the array. -->
    <!-- 'count' is required if you want to input and/or output array data as XPCML. -->
    <xs:attribute name="count" type="xs:string" />
    <!-- The number of characters in each string. -->
    <xs:attribute name="length" type="xs:string"/>
    <!-- The host CCSID for each string. -->
    <xs:attribute name="ccsid" type="xs:string"/>
    <!-- Specifies how to trim whitespace (left, right, both, none). -->
    <xs:attribute name="trim" type="trimType" />
    <!-- The size of each character ('chartype' in PCML). -->
    <xs:attribute name="bytesPerChar" type="charType" />
   <!-- The bidirectional string type. -->
    <xs:attribute name="bidiStringType" type="bidiStringTypeType" />
  </xs:complexType>

      <xs:complexType name="stringElementType">
        <xs:simpleContent>
           <xs:extension base="xs:string">
                <!-- The index into the array. -->
                <xs:attribute name="index" type="xs:nonNegativeInteger"  />
```

```
          </xs:extension>
        </xs:simpleContent>
       </xs:complexType>


<!-- Integer parameter (4 bytes on server) -->
    <xs:element name="intParm" type="intParmType"  nillable="true" substitutionGroup="intParmGroup"   />
      <xs:complexType name="intParmType" >
         <xs:simpleContent>
            <xs:extension base="intFieldType">
              <xs:attributeGroup ref="commonParmAttrs"/>
            </xs:extension>
         </xs:simpleContent>
      </xs:complexType>

<!-- intParm array type -->
  <xs:element name="arrayOfIntParm" type="intParmArrayType" substitutionGroup="intParmArrayGroup"
              nillable="true" />
  <xs:complexType name="intParmArrayType">
    <xs:sequence>
      <!-- 'i' is the tag used for non-struct array elements. -->
      <xs:element name="i" type="intElementType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <!-- 'count' is required if you want to input and/or output array data as XPCML. -->
    <xs:attribute name="count" type="xs:string" />
    <xs:attributeGroup ref="commonParmAttrs"/>
    <xs:attributeGroup ref="commonFieldAttrs"/>
  </xs:complexType>

  <xs:complexType name="intElementType">
    <xs:simpleContent>
       <xs:extension base="xs:int">
            <xs:attribute name="index" type="xs:nonNegativeInteger"  />
       </xs:extension>
    </xs:simpleContent>
  </xs:complexType>


<!-- Unsigned Integer parameter (4 bytes on server) -->
    <xs:element name="unsignedIntParm" type="unsignedIntParmType" nillable="true"
                substitutionGroup="unsignedIntParmGroup"   />
      <xs:complexType name="unsignedIntParmType">
         <xs:simpleContent>
            <xs:extension base="unsignedIntFieldType">
              <xs:attributeGroup ref="commonParmAttrs"/>
            </xs:extension>
         </xs:simpleContent>
      </xs:complexType>

<!-- unsigned intParm array type -->
  <xs:element name="arrayOfUnsignedIntParm" type="unsignedIntParmArrayType"
              substitutionGroup="unsignedIntParmArrayGroup" nillable="true" />
  <xs:complexType name="unsignedIntParmArrayType">
    <xs:sequence>
      <xs:element name="i" type="unsignedIntElementType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <!-- 'count' is required if you want to input and/or output array data as XPCML. -->
    <xs:attribute name="count" type="xs:string" />
    <xs:attributeGroup ref="commonParmAttrs"/>
    <xs:attributeGroup ref="commonFieldAttrs"/>
  </xs:complexType>

  <xs:complexType name="unsignedIntElementType">
    <xs:simpleContent>
       <xs:extension base="xs:unsignedInt">
            <xs:attribute name="index" type="xs:nonNegativeInteger"  />
       </xs:extension>
```

```
    </xs:simpleContent>
  </xs:complexType>


<!-- Short integer parameter (2 bytes on server) -->
   <xs:element name="shortParm" type="shortParmType" nillable="true" substitutionGroup="shortParmGroup"/>
      <xs:complexType name="shortParmType">
         <xs:simpleContent>
            <xs:extension base="shortFieldType">
              <xs:attributeGroup ref="commonParmAttrs"/>
            </xs:extension>
         </xs:simpleContent>
      </xs:complexType>

<!-- shortParm array type -->
  <xs:element name="arrayOfShortParm" type="shortParmArrayType" substitutionGroup="shortParmArrayGroup"
              nillable="true" />
  <xs:complexType name="shortParmArrayType">
    <xs:sequence>
      <xs:element name="i" type="shortElementType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <!-- 'count' is required if you want to input and/or output array data as XPCML. -->
    <xs:attribute name="count" type="xs:string" />
    <xs:attributeGroup ref="commonParmAttrs"/>
    <xs:attributeGroup ref="commonFieldAttrs"/>
  </xs:complexType>

  <xs:complexType name="shortElementType">
    <xs:simpleContent>
       <xs:extension base="xs:short">
            <xs:attribute name="index" type="xs:nonNegativeInteger"  />
       </xs:extension>
    </xs:simpleContent>
  </xs:complexType>


<!-- Unsigned Short integer parameter (2 bytes on server) -->
    <xs:element name="unsignedShortParm" type="unsignedShortParmType" nillable="true"
                substitutionGroup="unsignedShortParmGroup"   />
      <xs:complexType name="unsignedShortParmType">
         <xs:simpleContent>
            <xs:extension base="unsignedShortFieldType">
              <xs:attributeGroup ref="commonParmAttrs"/>
            </xs:extension>
         </xs:simpleContent>
      </xs:complexType>

<!-- unsignedShortParm array type -->
  <xs:element name="arrayOfUnsignedShortParm" type="unsignedShortParmArrayType"
              substitutionGroup="unsignedShortParmArrayGroup" nillable="true" />
  <xs:complexType name="unsignedShortParmArrayType">
    <xs:sequence>
      <xs:element name="i" type="unsignedShortElementType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <!-- 'count' is required if you want to input and/or output array data as XPCML. -->
    <xs:attribute name="count" type="xs:string" />
    <xs:attributeGroup ref="commonParmAttrs"/>
    <xs:attributeGroup ref="commonFieldAttrs"/>
  </xs:complexType>

  <xs:complexType name="unsignedShortElementType">
    <xs:simpleContent>
       <xs:extension base="xs:unsignedShort">
            <xs:attribute name="index" type="xs:nonNegativeInteger"  />
       </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
```

```
<!-- Long integer parameter (8 bytes on server) -->
    <xs:element name="longParm" type="longParmType" nillable="true" substitutionGroup="longParmGroup"  />
      <xs:complexType name="longParmType">
         <xs:simpleContent>
            <xs:extension base="longFieldType">
              <xs:attributeGroup ref="commonParmAttrs"/>
            </xs:extension>
         </xs:simpleContent>
      </xs:complexType>

<!-- longParm array type -->
  <xs:element name="arrayOfLongParm" type="longParmArrayType" substitutionGroup="longParmArrayGroup"
              nillable="true" />
  <xs:complexType name="longParmArrayType">
    <xs:sequence>
      <xs:element name="i" type="longElementType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <!-- 'count' is required if you want to input and/or output array data as XPCML. -->
    <xs:attribute name="count" type="xs:string" />
    <xs:attributeGroup ref="commonParmAttrs"/>
    <xs:attributeGroup ref="commonFieldAttrs"/>
  </xs:complexType>

  <xs:complexType name="longElementType">
    <xs:simpleContent>
       <xs:extension base="xs:long">
            <xs:attribute name="index" type="xs:nonNegativeInteger"  />
       </xs:extension>
    </xs:simpleContent>
  </xs:complexType>


<!-- ZonedDecimal parameter -->
    <xs:element name="zonedDecimalParm" type="zonedDecimalParmType" nillable="true"
                substitutionGroup="zonedDecimalParmGroup"   />
      <xs:complexType name="zonedDecimalParmType">
         <xs:simpleContent>
            <xs:extension base="zonedDecimalFieldType">
              <xs:attributeGroup ref="commonParmAttrs"/>
            </xs:extension>
         </xs:simpleContent>
      </xs:complexType>

<!-- zonedDecimalParm array type -->
  <xs:element name="arrayOfZonedDecimalParm" type="zonedDecimalParmArrayType"
              substitutionGroup="zonedDecimalParmArrayGroup" nillable="true" />
  <xs:complexType name="zonedDecimalParmArrayType">
    <xs:sequence>
      <xs:element name="i" type="zonedDecimalElementType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attributeGroup ref="commonParmAttrs"/>
    <xs:attributeGroup ref="commonFieldAttrs"/>
    <!-- 'count' is required if you want to input and/or output array data as XPCML. -->
    <xs:attribute name="count" type="xs:string" />
    <!-- The total number of digits in the field ('length' in PCML). -->
    <xs:attribute name="totalDigits" type="xs:positiveInteger"/>
    <!-- The number of fractional digits ('precision' in PCML). -->
    <xs:attribute name="fractionDigits" type="xs:nonNegativeInteger"/>
  </xs:complexType>

  <xs:complexType name="zonedDecimalElementType">
    <xs:simpleContent>
       <xs:extension base="xs:decimal">
            <xs:attribute name="index" type="xs:nonNegativeInteger"  />
       </xs:extension>
```

```
    </xs:simpleContent>
  </xs:complexType>

<!-- packedDecimal parameter -->
    <xs:element name="packedDecimalParm" type="packedDecimalParmType" nillable="true"
                substitutionGroup="packedDecimalParmGroup"   />
      <xs:complexType name="packedDecimalParmType">
         <xs:simpleContent>
            <xs:extension base="packedDecimalFieldType">
              <xs:attributeGroup ref="commonParmAttrs"/>
            </xs:extension>
         </xs:simpleContent>
      </xs:complexType>

<!-- packedDecimalParm array type -->
  <xs:element name="arrayOfPackedDecimalParm" type="packedDecimalParmArrayType"
              substitutionGroup="packedDecimalParmArrayGroup" nillable="true" />
  <xs:complexType name="packedDecimalParmArrayType">
    <xs:sequence>
      <xs:element name="i" type="packedDecimalElementType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attributeGroup ref="commonParmAttrs"/>
    <xs:attributeGroup ref="commonFieldAttrs"/>
    <!-- 'count' is required if you want to input and/or output array data as XPCML. -->
    <xs:attribute name="count" type="xs:string" />
    <xs:attribute name="totalDigits" type="xs:positiveInteger"/>
    <xs:attribute name="fractionDigits" type="xs:nonNegativeInteger"/>
  </xs:complexType>

  <xs:complexType name="packedDecimalElementType">
    <xs:simpleContent>
       <xs:extension base="xs:decimal">
            <xs:attribute name="index" type="xs:nonNegativeInteger"  />
       </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

<!-- Float parameter (4 bytes on server) -->
   <xs:element name="floatParm" type="floatParmType" nillable="true" substitutionGroup="floatParmGroup"/>
      <xs:complexType name="floatParmType">
         <xs:simpleContent>
            <xs:extension base="floatFieldType">
              <xs:attributeGroup ref="commonParmAttrs"/>
            </xs:extension>
         </xs:simpleContent>
      </xs:complexType>

<!-- floatParm array type -->
  <xs:element name="arrayOfFloatParm" type="floatParmArrayType" substitutionGroup="floatParmArrayGroup"
              nillable="true" />
  <xs:complexType name="floatParmArrayType">
    <xs:sequence>
      <xs:element name="i" type="floatElementType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <!-- 'count' is required if you want to input and/or output array data as XPCML. -->
    <xs:attribute name="count" type="xs:string" />
    <xs:attributeGroup ref="commonParmAttrs"/>
    <xs:attributeGroup ref="commonFieldAttrs"/>
  </xs:complexType>

  <xs:complexType name="floatElementType">
    <xs:simpleContent>
       <xs:extension base="xs:float">
            <xs:attribute name="index" type="xs:nonNegativeInteger"  />
       </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
```

```
<!-- Double parameter (8 bytes on server) -->
    <xs:element name="doubleParm" type="doubleParmType" nillable="true"
                substitutionGroup="doubleParmGroup"   />
      <xs:complexType name="doubleParmType">
         <xs:simpleContent>
            <xs:extension base="doubleFieldType">
              <xs:attributeGroup ref="commonParmAttrs"/>
            </xs:extension>
         </xs:simpleContent>
      </xs:complexType>

<!-- doubleParm array type -->
  <xs:element name="arrayOfDoubleParm" type="doubleParmArrayType"
              substitutionGroup="doubleParmArrayGroup" nillable="true" />
  <xs:complexType name="doubleParmArrayType">
    <xs:sequence>
      <xs:element name="i" type="doubleElementType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <!-- 'count' is required if you want to input and/or output array data as XPCML. -->
    <xs:attribute name="count" type="xs:string" />
    <xs:attributeGroup ref="commonParmAttrs"/>
    <xs:attributeGroup ref="commonFieldAttrs"/>
  </xs:complexType>

  <xs:complexType name="doubleElementType">
    <xs:simpleContent>
       <xs:extension base="xs:double">
            <xs:attribute name="index" type="xs:nonNegativeInteger"  />
       </xs:extension>
    </xs:simpleContent>
  </xs:complexType>


<!-- Hex binary parameter (any number of bytes; unsigned) -->
    <xs:element name="hexBinaryParm" type="hexBinaryParmType"  substitutionGroup="hexBinaryParmGroup"  />
      <xs:complexType name="hexBinaryParmType">
         <xs:simpleContent>
            <xs:extension base="hexBinaryFieldType">
              <!-- The field length in bytes ('length' in PCML). -->
              <xs:attribute name="totalBytes" type="xs:string"/>
              <xs:attributeGroup ref="commonParmAttrs"/>
            </xs:extension>
         </xs:simpleContent>
      </xs:complexType>

<!-- hexBinaryParm array type -->
  <xs:element name="arrayOfHexBinaryParm" type="hexBinaryParmArrayType"
              substitutionGroup="hexBinaryParmArrayGroup" nillable="true" />
  <xs:complexType name="hexBinaryParmArrayType">
    <xs:sequence>
      <xs:element name="i" type="hexBinaryElementType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="totalBytes" type="xs:string"/>
    <!-- 'count' is required if you want to input and/or output array data as XPCML. -->
    <xs:attribute name="count" type="xs:string" />
    <xs:attributeGroup ref="commonParmAttrs"/>
    <xs:attributeGroup ref="commonFieldAttrs"/>
  </xs:complexType>

  <xs:complexType name="hexBinaryElementType">
    <xs:simpleContent>
       <xs:extension base="xs:hexBinary">
            <xs:attribute name="index" type="xs:nonNegativeInteger"  />
       </xs:extension>
    </xs:simpleContent>
```

```
  </xs:complexType>

<!-- Structure parm type -->
  <xs:element name="structParm" type="structParmType"  substitutionGroup="structParmGroup"   />
      <xs:complexType name="structParmType">
         <xs:complexContent>
            <xs:extension base="structureParmArray">
              <xs:attribute name="struct" type="string50"/>
              <!-- Specifies whether the parameter is passed by value or reference ('passby' in PCML).-->
              <!-- Value only allowed for integer parameters. -->
              <xs:attribute name="passMode" type="passModeType"/>
              <!-- 'count' is required if you want to input and/or output array data as XPCML. -->
              <xs:attribute name="count" type="xs:string"/>
            </xs:extension>
         </xs:complexContent>
      </xs:complexType>

<!-- Structure parm array type -->
  <xs:element name="arrayOfStructParm" type="structParmArrayType"
              substitutionGroup="structParmArrayGroup" nillable="true" />
  <xs:complexType name="structParmArrayType">
    <xs:sequence>
        <!-- struct_i tag represents struct or struct parm array elements. -->
        <xs:element name="struct_i" type="structElementType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <!-- 'count' is required if you want to input and/or output array data as XPCML. -->
    <xs:attribute name="count" type="xs:string" />
    <xs:attributeGroup ref="commonParmAttrs"/>
    <xs:attributeGroup ref="commonFieldAttrs"/>
    <xs:attribute name="struct" type="string50"/>
  </xs:complexType>

  <xs:complexType name="structElementType">
    <xs:complexContent>
       <xs:extension base="structureParmArray">
            <xs:attribute name="index" type="xs:nonNegativeInteger"  />
       </xs:extension>
    </xs:complexContent>
  </xs:complexType>


<!-- Struct element -->
  <xs:element name="struct" type="structureParmArray" substitutionGroup="structOrProgram" />

<!-- Struct array type -->
  <xs:element name="arrayOfStruct" type="structArrayType" substitutionGroup="structArrayGroup"
              nillable="true" />
  <xs:complexType name="structArrayType">
    <xs:sequence>
       <!-- struct_i tag represents struct elements in an array. -->
       <xs:element name="struct_i" type="structElementType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <!-- The name of the struct. -->
    <xs:attribute name="name" type="string50"/>
    <!-- Number of elements in the array. -->
    <!-- 'count' is required if you want to input and/or output array data as XPCML. -->
    <xs:attribute name="count" type="xs:string" />
    <!-- Specifies whether this is an input, output, or input-output struct ('usage' in PCML). -->
    <xs:attribute name="passDirection" type="passDirectionType"/>
    <!-- The offset to the struct within an output parameter. -->
    <xs:attribute name="offset" type="xs:string" />
    <!-- The base location from which the 'offset' attribute is relative. -->
    <xs:attribute name="offsetFrom" type="xs:string" />
    <!-- The number of bytes to reserve for output data for the element. -->
    <xs:attribute name="outputSize" type="xs:string" />
    <!-- The lowest version of IBM i on which this element exists. -->
    <xs:attribute name="minvrm" type="string10" />
```

```
    <!-- The highest version of IBM i on which this element exists. -->
    <xs:attribute name="maxvrm" type="string10" />
  </xs:complexType>


<!-- Attributes that are common to all data field types. -->
  <xs:attributeGroup name="commonParmAttrs">
     <!-- Specifies whether this is an input, output, or input-output parameter ('usage' in PCML). -->
     <!-- The default value if none is specified is 'inherit'. -->
     <xs:attribute name="passDirection" type="passDirectionType"/>
     <!-- Specifies whether the parameter is passed by reference or value ('passby' in PCML). -->
     <!-- The default value if none is specified is 'reference'. -->
     <xs:attribute name="passMode" type="passModeType" />
     <!-- The offset to the element within an output parameter. -->
     <!-- The default value if none is specified is 0. -->
     <xs:attribute name="offset" type="xs:string" />
     <!-- The base location from which the 'offset' attribute is relative. -->
     <xs:attribute name="offsetFrom" type="xs:string" />
     <!-- The number of bytes to reserve for output data for the element. -->
     <xs:attribute name="outputSize" type="xs:string" />
     <!-- The lowest version of IBM i to which this field applies. -->
     <!-- If not specified, we assume this field applies to all versions. -->
     <xs:attribute name="minvrm" type="string10" />
     <!-- The highest version of IBM i to which this field applies. -->
     <!-- If not specified, we assume this field applies to all versions. -->
     <xs:attribute name="maxvrm" type="string10" />
  </xs:attributeGroup>

  <xs:simpleType name="passDirectionType">
    <xs:restriction base="xs:string">
       <xs:enumeration value="in"/>
       <xs:enumeration value="inout"/>
       <xs:enumeration value="out"/>
       <xs:enumeration value="inherit"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="passModeType">
    <xs:restriction base="xs:string">
       <xs:enumeration value="value"/>
       <xs:enumeration value="reference"/>
     </xs:restriction>
  </xs:simpleType>

  <!-- Following types are to maintain compatibility with PCML -->
  <xs:simpleType name="bidiStringTypeType">
    <xs:restriction base="xs:string">
       <xs:enumeration value="ST4"/>
       <xs:enumeration value="ST5"/>
       <xs:enumeration value="ST6"/>
       <xs:enumeration value="ST7"/>
       <xs:enumeration value="ST8"/>
       <xs:enumeration value="ST9"/>
       <xs:enumeration value="ST10"/>
       <xs:enumeration value="ST11"/>
       <xs:enumeration value="DEFAULT"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="charType">
    <xs:restriction base="xs:string">
       <xs:enumeration value="onebyte"/>
       <xs:enumeration value="twobyte"/>
     </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="trimType">
```

```
    <xs:restriction base="xs:string">
       <xs:enumeration value="none"/>
       <xs:enumeration value="left"/>
       <xs:enumeration value="right"/>
       <xs:enumeration value="both"/>
     </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="returnValueType">
    <xs:restriction base="xs:string">
       <xs:enumeration value="void"/>
       <xs:enumeration value="integer"/>
    </xs:restriction>
  </xs:simpleType>


  <xs:complexType name="structureParmArray">
    <xs:sequence>
       <xs:group ref="structureParm" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="name" type="string50"/>
    <xs:attribute name="passDirection" type="passDirectionType"/>
    <xs:attribute name="offset" type="xs:string" />
    <xs:attribute name="offsetFrom" type="xs:string" />
    <xs:attribute name="outputSize" type="xs:string" />
    <xs:attribute name="minvrm" type="string10" />
    <xs:attribute name="maxvrm" type="string10" />
  </xs:complexType>


<!-- A structureParm is exactly one of the following: stringParm, intParm,
     shortParm, longParm, zonedDecimalParm, packedDecimalParm, floatParm,
     doubleParm, or hexBinaryParm. -->
      <xs:group name="structureParm">
        <xs:choice>
          <xs:element ref="stringParmGroup"  />
          <xs:element ref="stringParmArrayGroup"  />
          <xs:element ref="intParmGroup"   />
          <xs:element ref="intParmArrayGroup"   />
          <xs:element ref="unsignedIntParmGroup"/>
          <xs:element ref="unsignedIntParmArrayGroup"/>
          <xs:element ref="shortParmGroup" />
          <xs:element ref="shortParmArrayGroup" />
          <xs:element ref="unsignedShortParmGroup"   />
          <xs:element ref="unsignedShortParmArrayGroup"   />
          <xs:element ref="longParmGroup"  />
          <xs:element ref="longParmArrayGroup"  />
          <xs:element ref="zonedDecimalParmGroup"  />
          <xs:element ref="zonedDecimalParmArrayGroup"  />
          <xs:element ref="packedDecimalParmGroup" />
          <xs:element ref="packedDecimalParmArrayGroup"/>
          <xs:element ref="floatParmGroup" />
          <xs:element ref="floatParmArrayGroup" />
          <xs:element ref="doubleParmGroup" />
          <xs:element ref="doubleParmArrayGroup" />
          <xs:element ref="hexBinaryParmGroup" />
          <xs:element ref="hexBinaryParmArrayGroup" />
          <xs:element ref="structParmGroup"/>
          <xs:element ref="structParmArrayGroup"/>
          <xs:element ref="structArrayGroup"/>
          <xs:element ref="struct"/>
        </xs:choice>
      </xs:group>


<!-- Field Definition schema. -->
```

```
<!-- Define basic System i native data types. -->

   <xs:complexType name="zonedDecimal">
      <xs:simpleContent>
         <xs:extension base="xs:decimal">
              <xs:attribute name="totalDigits" type="xs:positiveInteger"/>
              <xs:attribute name="fractionDigits" type="xs:nonNegativeInteger"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>


   <xs:complexType name="packedDecimal">
      <xs:simpleContent>
         <xs:extension base="xs:decimal">
              <xs:attribute name="totalDigits" type="xs:positiveInteger"/>
              <xs:attribute name="fractionDigits" type="xs:nonNegativeInteger"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>


 <xs:complexType name="structureFieldArray">
  <xs:sequence>
     <xs:group ref="structureField" minOccurs="1" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="name" type="string50"/>
  <!-- 'count' is required if you want to input and/or output array data as XPCML. -->
  <xs:attribute name="count" type="xs:string"/>
</xs:complexType>



<!-- Abstract type for "struct or program". -->
<xs:element name="structOrProgram" abstract="true" />


<!-- Abstract type for all data field types. -->
<xs:element name="stringFieldGroup" type="stringFieldType" abstract="true" />
<xs:element name="intFieldGroup" type="intFieldType" abstract="true" />
<xs:element name="unsignedIntFieldGroup" type="unsignedIntFieldType" abstract="true" />
<xs:element name="shortFieldGroup" type="shortFieldType" abstract="true" />
<xs:element name="unsignedShortFieldGroup" type="unsignedShortFieldType" abstract="true" />
<xs:element name="longFieldGroup" type="longFieldType" abstract="true" />
<xs:element name="zonedDecimalFieldGroup" type="zonedDecimalFieldType" abstract="true" />
<xs:element name="packedDecimalFieldGroup" type="packedDecimalFieldType" abstract="true" />
<xs:element name="floatFieldGroup" type="floatFieldType" abstract="true" />
<xs:element name="doubleFieldGroup" type="doubleFieldType" abstract="true" />
<xs:element name="hexBinaryFieldGroup" type="hexBinaryFieldType" abstract="true" />
<xs:element name="structFieldGroup" type="structFieldType" abstract="true" />



<!-- Declare each field element to be a specific field type. -->
<xs:element name="stringField" type="stringFieldType" substitutionGroup="stringFieldGroup"
            nillable="true"/>
<xs:element name="intField" type="intFieldType" nillable="true"
            substitutionGroup="intFieldGroup" />
<xs:element name="unsignedIntField" type="unsignedIntFieldType"
            substitutionGroup="unsignedIntFieldGroup" nillable="true"/>
<xs:element name="shortField" type="shortFieldType" nillable="true"
            substitutionGroup="shortFieldGroup" />
<xs:element name="unsignedShortField" type="unsignedShortFieldType" nillable="true"
            substitutionGroup="unsignedShortFieldGroup"  />
<xs:element name="longField" type="longFieldType" nillable="true"
            substitutionGroup="longFieldGroup" />
<xs:element name="hexBinaryField" type="hexBinaryFieldType" nillable="true"
```

```
              substitutionGroup="hexBinaryFieldGroup" />
  <xs:element name="zonedDecimalField" type="zonedDecimalFieldType" nillable="true"
              substitutionGroup="zonedDecimalFieldGroup"  />
  <xs:element name="packedDecimalField" type="packedDecimalFieldType" nillable="true"
              substitutionGroup="packedDecimalFieldGroup" />
  <xs:element name="doubleField" type="doubleFieldType" nillable="true"
              substitutionGroup="doubleFieldGroup" />
  <xs:element name="floatField" type="floatFieldType" nillable="true"
              substitutionGroup="floatFieldGroup" />


  <xs:element name="structField" type="structFieldType" nillable="true"
              substitutionGroup="structFieldGroup" />

<!-- A StructureField is exactly one of the following: stringField, intField,
     shortField, longField, zonedDecimalField, packedDecimalField, floatField,
     doubleField, or hexBinaryField. -->
      <xs:group name="structureField">
        <xs:choice>
          <xs:element ref="stringFieldGroup"/>
          <xs:element ref="intFieldGroup"/>
          <xs:element ref="unsignedIntFieldGroup"/>
          <xs:element ref="shortFieldGroup"/>
          <xs:element ref="unsignedShortFieldGroup"/>
          <xs:element ref="longFieldGroup"/>
          <xs:element ref="zonedDecimalFieldGroup"/>
          <xs:element ref="packedDecimalFieldGroup"/>
          <xs:element ref="floatFieldGroup"/>
          <xs:element ref="doubleFieldGroup"/>
          <xs:element ref="hexBinaryFieldGroup"/>
          <xs:element ref="structParmGroup"/>
          <xs:element ref="struct"/>
        </xs:choice>
      </xs:group>

  <!-- Character field -->
  <!-- Maps to AS400Text. -->
   <xs:complexType name="stringFieldType">
     <xs:simpleContent>
        <xs:extension base="xs:string">
          <!-- Number of characters. -->
          <xs:attribute name="length" type="xs:string"/>
          <!-- Indicates the field's encoding (CCSID) on the server. -->
          <xs:attribute name="ccsid" type="xs:string"/>
          <xs:attribute name="trim" type="trimType" />
          <xs:attribute name="bytesPerChar" type="charType" />
          <xs:attribute name="bidiStringType" type="bidiStringTypeType" />
          <xs:attributeGroup ref="commonFieldAttrs"/>
        </xs:extension>
     </xs:simpleContent>
   </xs:complexType>


  <!-- hexBinary field -->
  <!-- Maps to AS400ByteArray. -->
   <xs:complexType name="hexBinaryFieldType">
     <xs:simpleContent>
        <xs:extension base="xs:hexBinary">
          <xs:attributeGroup ref="commonFieldAttrs"/>
        </xs:extension>
     </xs:simpleContent>
   </xs:complexType>


  <!-- Float field -->
  <!-- Maps to AS400Float4. -->
   <xs:complexType name="floatFieldType">
```

```
    <xs:simpleContent>
        <xs:extension base="xs:float">
            <xs:attributeGroup ref="commonFieldAttrs"/>
        </xs:extension>
    </xs:simpleContent>
   </xs:complexType>

<!-- zonedDecimal field -->
<!-- Maps to AS400ZonedDecimal. -->
 <xs:complexType name="zonedDecimalFieldType">
    <xs:simpleContent>
        <xs:extension base="zonedDecimal">
            <xs:attributeGroup ref="commonFieldAttrs"/>
        </xs:extension>
    </xs:simpleContent>
 </xs:complexType>

<!-- packedDecimal field -->
<!-- Maps to AS400PackedDecimal. -->
 <xs:complexType name="packedDecimalFieldType">
    <xs:simpleContent>
        <!-- In DDS, "binary" values are 1-18 digits; if field length is
             greater than 9, then decimal positions value must be 0. -->
        <xs:extension base="packedDecimal">
            <xs:attributeGroup ref="commonFieldAttrs"/>
        </xs:extension>
    </xs:simpleContent>
 </xs:complexType>

<!-- int field -->
<!-- Maps to AS400Bin4. -->
 <xs:complexType name="intFieldType">
    <xs:simpleContent>
        <xs:extension base="xs:int">
           <xs:attributeGroup ref="commonFieldAttrs"/>
        </xs:extension>
    </xs:simpleContent>
 </xs:complexType>

<!-- unsigned int field -->
<!-- Maps to AS400Bin4. -->
 <xs:complexType name="unsignedIntFieldType">
    <xs:simpleContent>
        <xs:extension base="xs:unsignedInt">
           <xs:attributeGroup ref="commonFieldAttrs"/>
        </xs:extension>
    </xs:simpleContent>
 </xs:complexType>

<!-- short field -->
<!-- Maps to AS400Bin2. -->
 <xs:complexType name="shortFieldType">
    <xs:simpleContent>
        <xs:extension base="xs:short">
           <xs:attributeGroup ref="commonFieldAttrs"/>
        </xs:extension>
    </xs:simpleContent>
 </xs:complexType>

<!-- unsigned short field -->
<!-- Maps to AS400Bin2. -->
 <xs:complexType name="unsignedShortFieldType">
    <xs:simpleContent>
        <xs:extension base="xs:unsignedShort">
           <xs:attributeGroup ref="commonFieldAttrs"/>
        </xs:extension>
    </xs:simpleContent>
```

```
   </xs:complexType>


  <!-- long field -->
  <!-- Maps to AS400Bin8. -->
   <xs:complexType name="longFieldType">
     <xs:simpleContent>
         <xs:extension base="xs:long">
            <xs:attributeGroup ref="commonFieldAttrs"/>
         </xs:extension>
     </xs:simpleContent>
   </xs:complexType>

  <!-- double field -->
  <!-- Maps to AS400Float8. -->
   <xs:complexType name="doubleFieldType">
     <xs:simpleContent>
         <xs:extension base="xs:double">
            <xs:attributeGroup ref="commonFieldAttrs"/>
         </xs:extension>
     </xs:simpleContent>
   </xs:complexType>



  <!-- struct Field -->
   <xs:complexType name="structFieldType">
     <xs:simpleContent>
       <xs:extension base="xs:string">
         <xs:attribute name="struct" type="string50"/>
         <xs:attributeGroup ref="commonFieldAttrs"/>
       </xs:extension>
     </xs:simpleContent>
   </xs:complexType>


<!-- Attributes that are common to all data field types. -->
<xs:attributeGroup name="commonFieldAttrs">
   <xs:attribute name="name" type="string50"/>
   <xs:attribute name="columnHeading1" type="string20"/>
   <xs:attribute name="columnHeading2" type="string20"/>
   <xs:attribute name="columnHeading3" type="string20"/>
   <xs:attribute name="description" type="string50"/>
   <xs:attribute name="defaultValue" type="xs:string"/><!-- Max length of string is 65535 characters. -->
   <xs:attribute name="nullable" type="xs:boolean"/>
   <xs:attribute name="isEmptyString" type="xs:boolean"/><!-- Indicate this is an empty string. -->
</xs:attributeGroup>

<!-- Utility types.  -->

<xs:simpleType name="ccsidType">
                     <xs:restriction base="xs:nonNegativeInteger">
                      <xs:maxInclusive value="65535"/>
                     </xs:restriction>
                    </xs:simpleType>

<xs:simpleType name="string10">
                     <xs:restriction base="xs:string">
                      <xs:maxLength value="10"/>
                     </xs:restriction>
                    </xs:simpleType>

<xs:simpleType name="string20">
                     <xs:restriction base="xs:string">
                      <xs:maxLength value="20"/>
                     </xs:restriction>
                    </xs:simpleType>
```

```
<xs:simpleType name="string50">
                      <xs:restriction base="xs:string">
                       <xs:maxLength value="50"/>
                      </xs:restriction>
                     </xs:simpleType>
</xs:schema>
```

XPCML の構文:

XPCML スキーマは複数のエレメント・タグを定義し、個々のエレメント・タグには属性タグが含まれています。

以下の表には、 XPCML ソース・ファイル中で宣言して定義できるさまざまなエレメントがリストされています。 1 列目の個々のエントリーは、 XPCML スキーマの該当するセクションにリンクしています。

XPCML タグ	説明	同等の PCML タグ
doubleParm	倍精度パラメーターを定義します。	データ (タイプ = float、長さ = 8)
arrayOfDoubleParm	倍精度の配列になっているパラメーターを定義します。	
floatParm	浮動小数点パラメーターを定義します。	データ (タイプ = float、長さ = 4)
arrayOfFloatParm	浮動小数点の配列になっているパラメーターを定義します。	
hexBinaryParm	16 進数で表されるバイト・パラメーターを定義します。	バイト (おおよそ同等、16 進表記)
arrayOfHexBinaryParm	16 進数バイトの配列になっているパラメーターを定義します。	
intParm	整数パラメーターを定義します。	データ (タイプ = int、長さ = 4)
arrayOfIntParm	整数の配列になっているパラメーターを定義します。	
longParm	長形式パラメーターを定義します。	データ (タイプ = int、長さ = 8)
arrayOfLongParm	長形式の配列になっているパラメーターを定義します。	
packedDecimalParm	パック 10 進数パラメーターを定義します。	データ (タイプ = packed)
arrayOfPackedDecimalParm	パック 10 進数の配列になっているパラメーターを定義します。	
parameterList	囲みタグがプログラムのパラメーター定義すべてを表していることをシグナル通知します。	
program	1 つのプログラム呼び出しを記述した XML を開始および終了します。	program
shortParm	短形式パラメーターを定義します。	データ (タイプ int、長さ 2)
arrayOfShortParm	短形式の配列になっているパラメーターを定義します。	
stringParm	ストリング・パラメーターを定義します。	

XPCML タグ	説明	同等の PCML タグ
arrayOfStringParm	ストリングの配列になっているパラメーターを定義します。	
struct	プログラムに対する引数として、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
arrayOfStruct	構造体の配列を定義します。	
structParm	XPCML 文書中の他の場所にあり、この文書中の特定の場所に組み込む struct タグに対する参照を表します。	データ (タイプ = struct)
arrayOfStructParm	構造体パラメーターの配列になっているパラメーターを定義します。	
unsignedIntParm	符号なし整数パラメーターを定義します。	データ (タイプ = int、長さ = 4、精度 = 32)
arrayOfUnsignedIntParm	符号なし整数の配列になっているパラメーターを定義します。	
unsignedShortParm	符号なし短形式パラメーターを定義します。	データ (タイプ = int、長さ = 2、精度 = 16)
arrayOfUnsignedShortParm	符号なし短形式の配列になっているパラメーターを定義します。	
xpcml	プログラム呼び出しの形式を記述した XPCML ソース・ファイルを開始および終了します。	
zonedDecimalParm	ゾーン 10 進数パラメーターを定義します。	データ (タイプ zoned)
arrayOfZonedDecimalParm	ゾーン 10 進数の配列になっているパラメーターを定義します。	

XPCML タグ属性:

XPCML スキーマは複数のエレメント・タグを定義し、個々のエレメント・タグには属性タグが含まれています。以下の表では、エレメントごとにさまざまな属性をリストし、説明しています。

XPCML タグとそれらの属性に関する固有で詳細な説明については、 XPCML のスキーマを参照してください。

XPCML タグ	属性	説明
hexBinaryParm	末尾の 2 列をデータで埋め、形式を決めます。	
arrayOfHexBinaryParm		
doubleParm	倍精度パラメーターを定義します。	float (長さ 8)
arrayOfDoubleParm	倍精度の配列になっているパラメーターを定義します。	
floatParm	浮動小数点パラメーターを定義します。	データ (タイプ float、長さ 4)

XPCML タグ	属性	説明
arrayOfFloatParm	浮動小数点の配列になっているパラメーターを定義します。	
intParm	整数パラメーターを定義します。	データ (タイプ int、長さ 4)
arrayOfIntParm	整数の配列になっているパラメーターを定義します。	
longParm	長形式パラメーターを定義します。	データ (タイプ int、長さ 8)
arrayOfLongParm	長形式の配列になっているパラメーターを定義します。	
packedDecimalParm	パック 10 進数パラメーターを定義します。	データ (タイプ packed)
arrayOfPackedDecimalParm	パック 10 進数の配列になっているパラメーターを定義します。	
parameterList	囲みタグがプログラムのパラメーター定義すべてを表していることをシグナル通知します。	
program	1 つのプログラム呼び出しを記述した XML を開始および終了します。	
shortParm	短形式パラメーターを定義します。	データ (タイプ int、長さ 2)
arrayOfShortParm	短形式の配列になっているパラメーターを定義します。	
stringParm	ストリング・パラメーターを定義します。	
arrayOfStringParm	ストリングの配列になっているパラメーターを定義します。	
struct	プログラムに対する引数として、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。	
arrayOfStruct	構造体の配列を定義します。	
structParm	XPCML 文書中の他の場所にあり、この文書中の特定の場所に組み込む struct タグに対する参照を表します。	データ (タイプ struct)
arrayOfStructParm	構造体パラメーターの配列になっているパラメーターを定義します。	
unsignedIntParm	符号なし整数パラメーターを定義します。	データ (タイプ int、長さ 4、精度 32)
arrayOfUnsignedIntParm	符号なし整数の配列になっているパラメーターを定義します。	
unsignedShortParm	符号なし短形式パラメーターを定義します。	データ (タイプ int、長さ 2、精度 16)
arrayOfUnsignedShortParm	符号なし短形式の配列になっているパラメーターを定義します。	
xpcml	プログラム呼び出しの形式を記述した XPCML ソース・ファイルを開始および終了します。	

XPCML タグ	属性	説明
zonedDecimalParm	ゾーン 10 進数パラメーターを定義します。	データ (タイプ zoned)
arrayOfZonedDecimalParm	ゾーン 10 進数の配列になっているパラメーターを定義します。	

XPCML の使用

XPCML の使用法は PCML の使用法に似ています。以下のステップは、 XPCML を使用するために実行する必要のあるアクションの一般的な概要を示します。

1. XPCML を使用して、プログラム呼び出しの仕様を記述します。
2. ProgramCallDocument オブジェクトを作成します。
3. ProgramCallDocument.callProgram() を使用してプログラムを実行します。

XPCML の使用法は PCML の使用法に似ているものの、以下の拡張機能が備えられています。

- 自動的にパーサーにパラメーター値を妥当性検査させる。
- プログラム・パラメーターの値を指定して渡す。
- サーバーへのプログラム呼び出しの結果を XPCML で取り出す。
- 既存の PCML 文書を同等の XPCML 文書に変換する。
- XPCML スキーマを拡張し、単純および複雑なエレメントおよび属性を新しく定義する。

例えば、IBM Toolbox for Java を使用して、 XPCML スキーマを拡張して新しいパラメーターおよびデータ型を作成できます。この XPCML 能力を使用して、XPCML ソース・ファイルを圧縮できます。圧縮すると、ファイルを読みやすくなり、コードを使いやすくなります。

XPCML の使用についての詳細は、以下のページを参照してください。

479 ページの『既存の PCML から XPCML への変換』
ProgramCallDocument クラスには transformPCMLToXPCML メソッドが含まれており、 このメソッドを使用すると既存の PCML 文書を同等の XPCML 文書に変換できます。

480 ページの『XPCML を使用してサーバー上のプログラムを呼び出す』
XPCML ファイルの作成後に、 XPCML 仕様およびデータ値を使用してIBM i サーバー上のプログラムを呼び出せる ProgramCallDocument オブジェクトを作成する必要があります。

481 ページの『プログラム呼び出しの結果を XPCML として取得する』
サーバー・プログラムを呼び出した後に、 ProgramCallDocument.getValue メソッドを使用して、 プログラム・パラメーター値を表す Java オブジェクトを検索できます。

481 ページの『パラメーター値を XPCML として渡す』
XPCML ソース・ファイル中でプログラム・パラメーターの値を設定し、 そのパラメーター値を XPCML として渡すことができます。

482 ページの『圧縮された XPCML の使用』
XPCML は拡張可能なので、 XPCML スキーマで指定されているパラメーター・タイプを拡張する新しいパラメーター・タイプを定義できます。 XPCML を圧縮することにより、XPCML スキーマを拡張して新しいデータ型定義を作成し、 XPCML 文書の読み易さおよび使用可能度を単純化して改善できます。

483 ページの『XPCML の解析エラーの識別』
XPCML スキーマ文書の妥当性検査時に、 完全な妥当性検査を行う XML パーサーは警告、致命的でない解析エラー、 および致命的な解析エラーを生成する場合があります。

既存の PCML から XPCML への変換:

ProgramCallDocument クラスには transformPCMLToXPCML メソッドが含まれており、このメソッドを使用すると既存の PCML 文書を同等の XPCML 文書に変換できます。

XPCML には、PCML で定義できる、すべてのエレメントおよび属性の比較可能な定義があります。transformPCMLToXPCML() を使用すると、エレメントおよび属性の PCML 表記が同等の XPCML に変換されます。

XPCML と PCML で同等な属性の名前が違う場合があることに注意してください。例えば、PCML の属性「usage」は XPCML では属性「passDirection」です。 PCML と比較した XPCML の使用法について詳しくは、XPCML のスキーマおよび構文を参照してください。

このメソッドは既存の PCML 文書 (InputStream オブジェクトとしてメソッドに渡す) を使用して、同等の XPCML を OutputStream オブジェクトとして生成します。 transformPCMLToXPCML() は静的メソッドなので、事前に ProgramCallDocument オブジェクトを作成せずにこのメソッドを呼び出せます。

例: PCML 文書を XPCML 文書に変換する

以下の例は、 PCML 文書 (名前 myPCML.pcml) を XPCML 文書 (名前 myXPCML.xpcml) に変換する方法を示しています。

注: XPCML ファイルのファイル拡張子として .xpcml を指定しなければなりません。ファイル拡張子として .xpcml を使用すると、 ProgramCallDocument クラスがファイルを XPCML として認識することが保証されます。拡張子を指定しないと、 ProgramCallDocument はファイルが PCML であると想定します。

PCML 文書 myPCML.pcml

```
<!-- myPCML.pcml -->
<pcml version="4.0">
   <program name="prog1" path="/QSYS.LIB/W95LIB.LIB/PROG1.PGM">
      <data type="char" name="parm1" usage="in" passby="reference"
            minvrm="V5R2M0" ccsid="37" length="10" init="Value 1"/>
    </program>
</pcml>
```

myPCML.pcml を myPCML.xpcml に変換する Java コード

```
try {
  InputStream pcmlStream = new FileInputStream("myPCML.pcml");
  OutputStream xpcmlStream = new FileOutputStream("myXPCML.xpcml");
  ProgramCallDocument.transformPCMLToXPCML(pcmlStream, xpcmlStream);
}
catch (Exception e) {
  System.out.println("error:  - "+e.getMessage());
  e.printStackTrace();
}
```

結果の XPCML 文書 myXPCML.xpcml

```
<?xml version="1.0" encoding="UTF-8"?>
   <!-- myXPCML.xpcml -->
   <xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="xpcml.xsd" version="4.0">
   <program name="prog1" path="/QSYS.LIB/W95LIB.LIB/PROG1.PGM">
      <parameterList>
         <stringParm name="parm1" passDirection="in" passMode="reference"
```

```
                        minvrm="V5R2M0" ccsid="37" length="10">Value 1</stringParm>
</parameterList>
          </program>
     </xpcml>
```

transformPCMLToXPCML() および ProgramCallDocument クラスについての詳細は、以下のページを参照してください。

ProgramCallDocument javadoc 情報

XPCML を使用してサーバー上のプログラムを呼び出す:

XPCML ファイルの作成後に、 XPCML 仕様およびデータ値を使用してIBM i サーバー上のプログラムを呼び出せる ProgramCallDocument オブジェクトを作成する必要があります。

ProgramCallDocument コンストラクター上に XPCML ファイルの名前を渡して、 XPCML の ProgramCallDocument を作成してください。この方法で XPCML の ProgramCallDocument を作成するには、まず XPCML 文書を解析して妥当性検査してから、 ProgramCallDocument オブジェクトを作成します。

XPCML 文書を解析して妥当性検査するには、完全な妥当性検査を行う XML パーサーが CLASSPATH に組み込まれていることを確認してください。 XPCML の実行要件についての詳細は、以下のページを参照してください。

457 ページの『XPCML を使用するための要件』

以下の例は、 XPCML ファイル myXPCML.xpcml の ProgramCallDocument オブジェクトを作成する方法を示しています。

```
system = new AS400();
// Create a ProgramCallDocument into which to parse the file.
ProgramCallDocument xpcmlDoc =
   new ProgramCallDocument(system, "myXPCML.xpcml");
```

XPCML の ProgramCallDocument を作成する場合と PCML の ProgramCallDocument を作成する場合の違いは、 PCML 文書の代わりに XPCML 文書をコンストラクターに渡すことのみです。

注: XPCML ファイルのファイル拡張子として .xpcml を指定しなければなりません。ファイル拡張子として .xpcml を使用すると、 ProgramCallDocument クラスがファイルを XPCML として認識することが保証されます。拡張子を指定しないと、 ProgramCallDocument はファイルが PCML であると想定します。

XPCML を使用してサーバー上のプログラムを呼び出す

ProgramCallDocument オブジェクトの作成後に、 ProgramCallDocument クラスのいずれかのメソッドを使用して、XPCML 文書を処理してください。例えば、ProgramCallDocument.callProgram() を使用して IBM i プログラムを呼び出したり、 XPCML 入力パラメーターの値を変更してから該当する ProgramCallDocument.setValue メソッドを使用してサーバー・プログラムを呼び出したりしてください。

以下の例は、 XPCML ファイル (名前 myXPCML.xpcml) の ProgramCallDocument オブジェクトを作成する方法を示しています。この例は、ProgramCallDocument オブジェクトの作成後に、 XPCML 文書中に指定されているプログラム (PROG1) を呼び出します。この例の場合、XPCML の使用法と PCML の使用法の違いは、 XPCML ファイルを ProgramCallDocument コンストラクターに渡すことのみです。

アプリケーションが XPCML 文書を読み取って解析した後に、 XPCML 文書は PCML 文書と全く同様の働きをします。この時点で、PCML で使用される既存のメソッドを XPCML で使用できます。

```
system = new AS400();

// Create a ProgramCallDocument into which to parse the file.
ProgramCallDocument xpcmlDoc = new ProgramCallDocument(system, "myXPCML.xpcml");

// Call PROG1
boolean rc = xpcmlDoc.callProgram("PROG1");
```

プログラム呼び出しの結果を XPCML として取得する:

サーバー・プログラムを呼び出した後に、 ProgramCallDocument.getValue メソッドを使用して、プログラム・パラメーター値を表す Java オブジェクトを検索できます。

さらに、以下の generateXPCML メソッドを使用すると、 ProgramCallDocument はプログラム呼び出しの結果を XPCML として戻すことができます。

- generateXPCML(String fileName): ProgramCallDocument オブジェクトの構成に使用する XPCML ソース・ファイル全体の結果を、 XPCML 形式で生成する。指定されたファイル名のファイル中にこの XPCML を保管します。
- generateXPCML(String pgmName, String fileName): 指定されたプログラムおよびパラメーターのみの結果を XPCML 形式で生成する。指定されたファイル名のファイル中にこの XPCML を保管します。
- generateXPCML(java.io.OutputStream outputStream): XPCML ソース・ファイル全体の結果を XPCML 形式で生成する。指定された OutputStream オブジェクト中にこの XPCML を保管します。
- generateXPCML(String pgmName, java.io.OutputStream outputStream): 指定されたプログラムおよびパラメーターのみの結果を XPCML 形式で生成する。指定された OutputStream オブジェクト中にこの XPCML を保管します。

ProgramCallDocument クラスの詳細は、ProgramCallDocument Javadoc の解説を参照してください。

以下の例は、XPCML ProgramCallDocument を構成し、IBM i プログラムを呼び出し、プログラム呼び出しの結果を XPCML として取り出す方法を示しています。

798 ページの『例: プログラム呼び出しの結果を XPCML として取り出す』

関連情報

ProgramCallDocument Javadoc

パラメーター値を XPCML として渡す:

XPCML ソース・ファイル中でプログラム・パラメーターの値を設定し、そのパラメーター値を XPCML として渡すことができます。

ProgramCallDocument オブジェクトが XPCML 文書を読み取って解析する際に、 XPCML に指定されたパラメーターごとに該当する setValue メソッドが自動的に呼び出されます。

XPCML を使用してパラメーター値を渡すと、複雑な構造および配列の値を設定した Java コードを作成するという負担が軽減されます。

以下の例では、配列を構成してパラメーター値を XPCML として渡すさまざまな方法を示します。

801 ページの『例: パラメーター値を XPCML として渡す』

802 ページの『例: パラメーター値の配列を XPCML として渡す』

圧縮された XPCML の使用:

XPCML は拡張可能なので、 XPCML スキーマで指定されているパラメーター・タイプを拡張する新しいパラメーター・タイプを定義できます。 XPCML を圧縮することにより、XPCML スキーマを拡張して新しいデータ型定義を作成し、 XPCML 文書の読み易さおよび使用可能度を単純化して改善できます。

以下の説明は、XPCML スキーマについて理解していることを前提としています。 XPCML のスキーマについての詳細は、以下のページを参照してください。

458 ページの『XPCML のスキーマおよび構文』

既存の XPCML ソースを圧縮するには、 ProgramCallDocument.condenseXPCML メソッドを使用します。このメソッドは以下のものを生成します。

- 既存の XPCML ソース中のパラメーターごとに新しいタイプ定義を含む、拡張されたスキーマ
- 拡張されたスキーマ中のタイプ定義を使用する、新しい XPCML ソース

XPCML の圧縮についての詳細は、以下のページを参照してください。

『既存の XPCML 文書の圧縮』

807 ページの『例: 圧縮された XPCML を使用して ProgramCallDocument オブジェクトを作成する』

807 ページの『例: プログラム呼び出しの結果を圧縮された XPCML として取得する』

既存の XPCML 文書の圧縮:

既存の XPCML 文書を圧縮すると、 XPCML ソースの読みやすさと使いやすさが向上します。圧縮された XPCML を作成するには、 ProgramCallDocument.condenseXPCML メソッドを使用してください。

condenseXPCML() を呼び出すには、このメソッドに以下のパラメーターを指定してください。

- 既存の XPCML を表す入力ストリーム
- 圧縮された XPCML を表す出力ストリーム
- 新しい拡張されたスキーマを表す出力ストリーム
- 該当する形式の新しいスキーマの名前 (mySchema.xsd など)

condenseXPCML() および ProgramCallDocument クラスについての詳細は、ProgramCallDocument Javadoc の解説を参照してください。

ProgramCallDocument.condenseXPCML() は静的メソッドなので、このメソッドを呼び出すのに ProgramCallDocument オブジェクトをインスタンス化する必要はありません。

例

以下の例は、既存の XPCML 文書を圧縮する方法を示しています。

最初の例は単純なもので、元の XPCML ソース、圧縮結果の XPCML、および拡張されたスキーマが含まれています。 2 つ目の例の方が長く複雑で、 condenseXPCML() を呼び出す Java コードおよび拡張されたスキーマ中に新しく生成された数種のみのタイプ定義が含まれています。

804 ページの『例: 既存の XPCML 文書を圧縮する』

804 ページの『例: 既存の XPCML 文書を圧縮する (Java コードを含む)』

関連情報

ProgramCallDocument Javadoc

XPCML の解析エラーの識別:

XPCML スキーマ文書の妥当性検査時に、完全な妥当性検査を行う XML パーサーは警告、致命的でない解析エラー、および致命的な解析エラーを生成する場合があります。

警告および致命的でない解析エラーの場合は、解析は失敗しません。警告および致命的でない解析エラーを調べて、 XPCML ソースの問題の判別に役立てることもできます。致命的な解析エラーの場合は、解析は例外終了します。

XPCML 文書の解析時に警告および致命的でないパーサー・エラーを表示するには、アプリケーション中でトレースをオンにしてトレース・カテゴリーを PCML に設定してください。

例

完全な妥当性検査を行う XML パーサーが、値のない数値パラメーター・タイプに関するエラーを生成します。以下の例は、サンプル XPCML ソースおよび結果の致命的でない解析エラーを示しています。

XPCML ソース

```
<program name="prog1"/>
   <parameterList>
      <intParm name="parm1"/>
   </parameterList>
</program>
```

結果のエラー

```
Tue Mar 25 15:21:44 CST 2003  [Error]: cvc-complex-type.2.2: Element
'intParm' must have no element [children], and the value must be valid.
```

この種のエラーを避けるには、 nil=true 属性を intParm エレメントに追加してください。 nil=true 属性は、故意にエレメントを空のままにしていることをパーサーにシグナル通知します。前述の XPCML ソースに nil=true 属性を追加した例を以下に示します。

```
<program name="prog1"/>
   <parameterList>
      <intParm xsi:nil="true" name="parm1"/>
   </parameterList>
</program>
```

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、IBM i 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 : IBM Toolbox for Java での JDBC の使用に関する質問への回答が記載されています。

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

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必要があります。統合ファイル・システム名は、IBM i 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されるときのサーバー・オブジェクトの名前です。

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。ライブラリー名は 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。) タイプ名は 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 (/) はパス区切り文字である。
- 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
用紙定義	.FORMDF
ライブラリー	.LIB
メンバー	.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 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 (IBM i 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、IBM 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 IBM 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	IBM Toolbox for Java 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。

Java プログラムの接続の管理

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。

IBM i サーバーに接続するには、 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、 IBM i 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、システム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

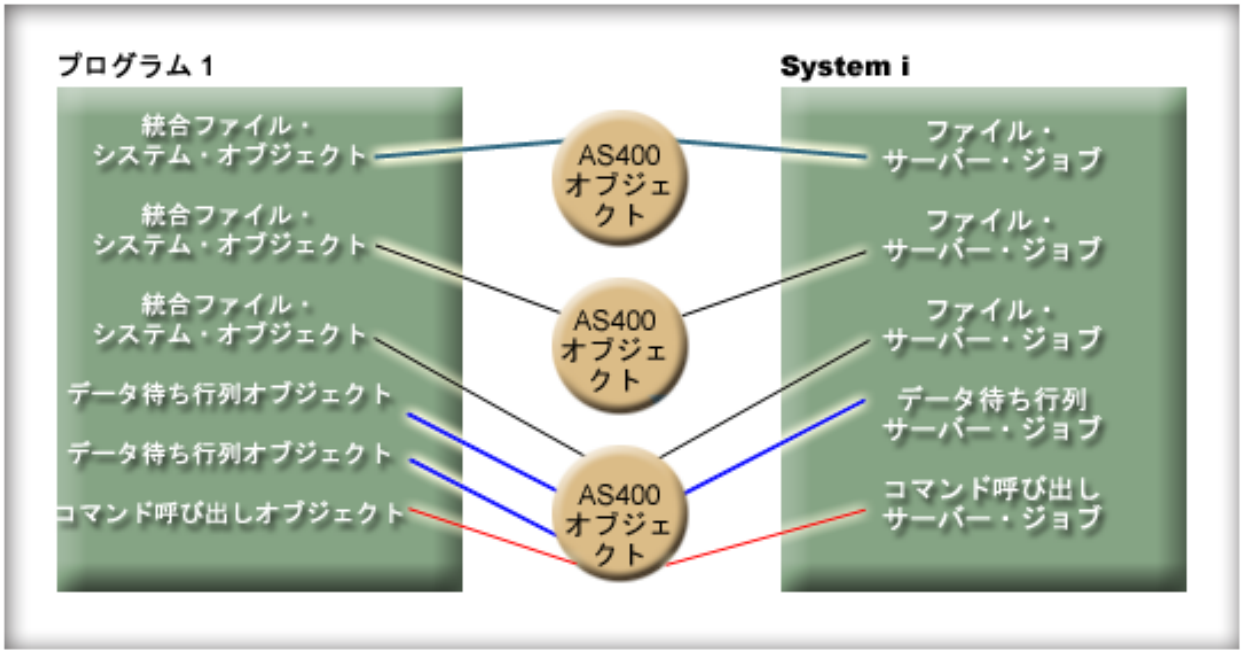
- 統合ファイル・システム
- 印刷
- データ待ち行列
- レコード・レベルでのアクセス

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。

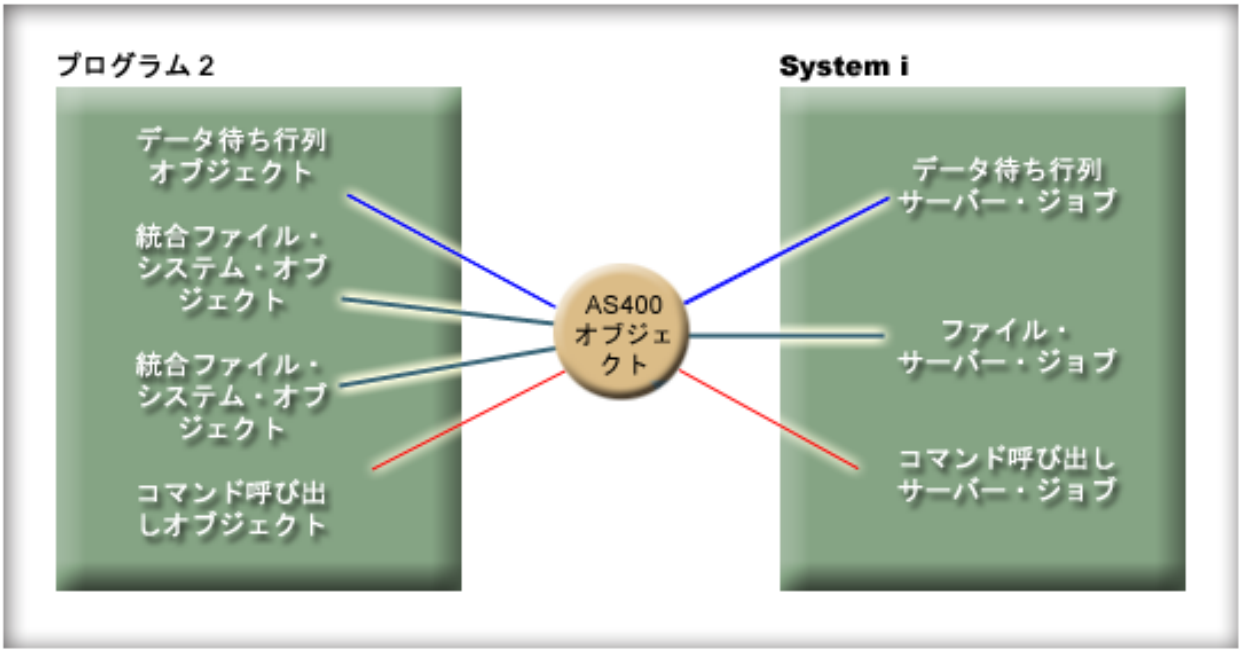
Java プログラムでは、システムへの接続の数を制御することができます。通信のパフォーマンスを最適化するために、Java プログラムでは、図 1 で示すように、同じサーバーについて複数の AS400 オブジェクトを作成することができます。これにより、システムへの複数のソケット接続が作成されます。

図 1: 同じ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



サーバー・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、サーバー上で使用されるリソースの量が減ります。

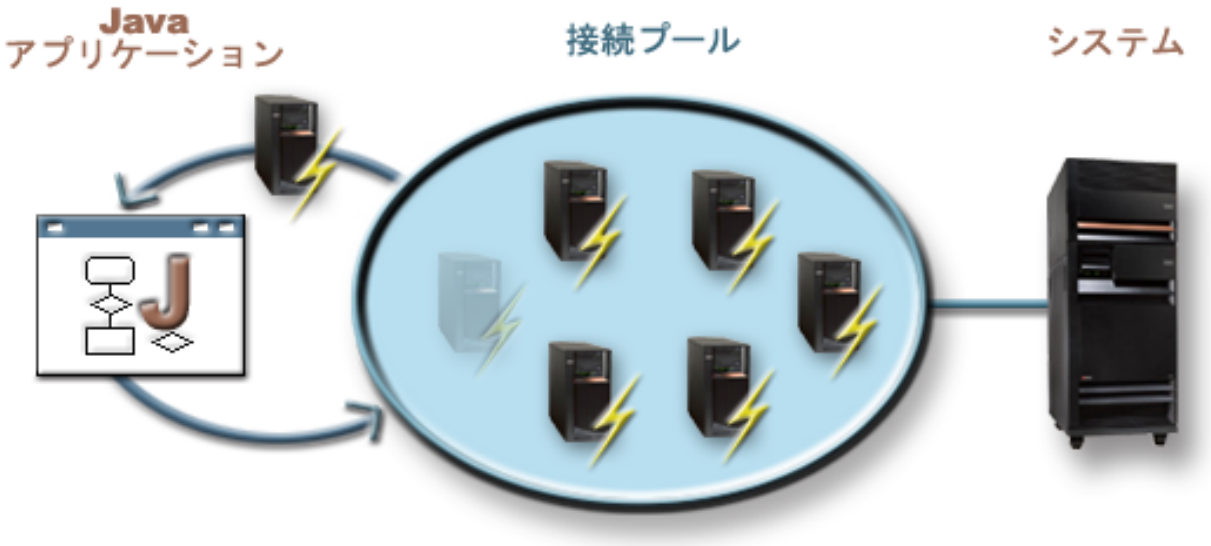
図 2: 同じシステムについて 1 つの AS400 オブジェクトとソケット接続を作成する Java プログラム



注: 多くの接続を作成するとサーバー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、接続にかかる時間を減らすことができます。

図 3: 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、同じサーバー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、サーバーへの接続は 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、サーバーへの接続が 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 オブジェクトはサーバー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 IBM i 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, and so on.))
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、システム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。

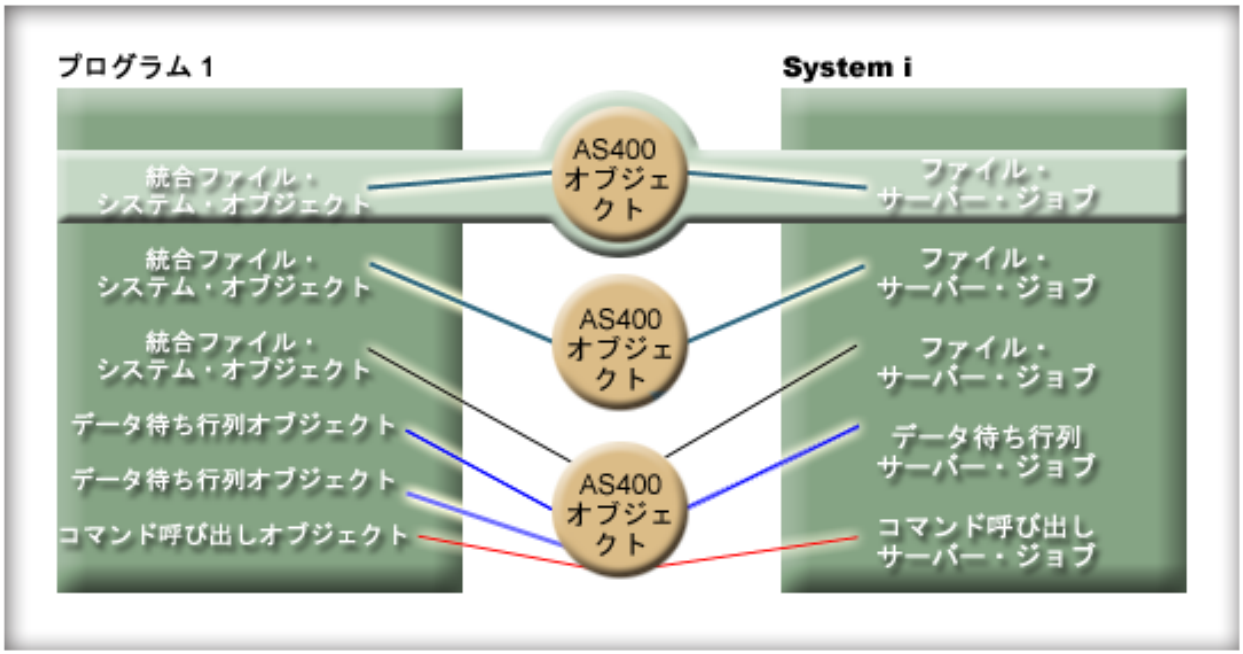
例 1: この例は、システムに事前接続する方法を示します。

```
// 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する必要があります。 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

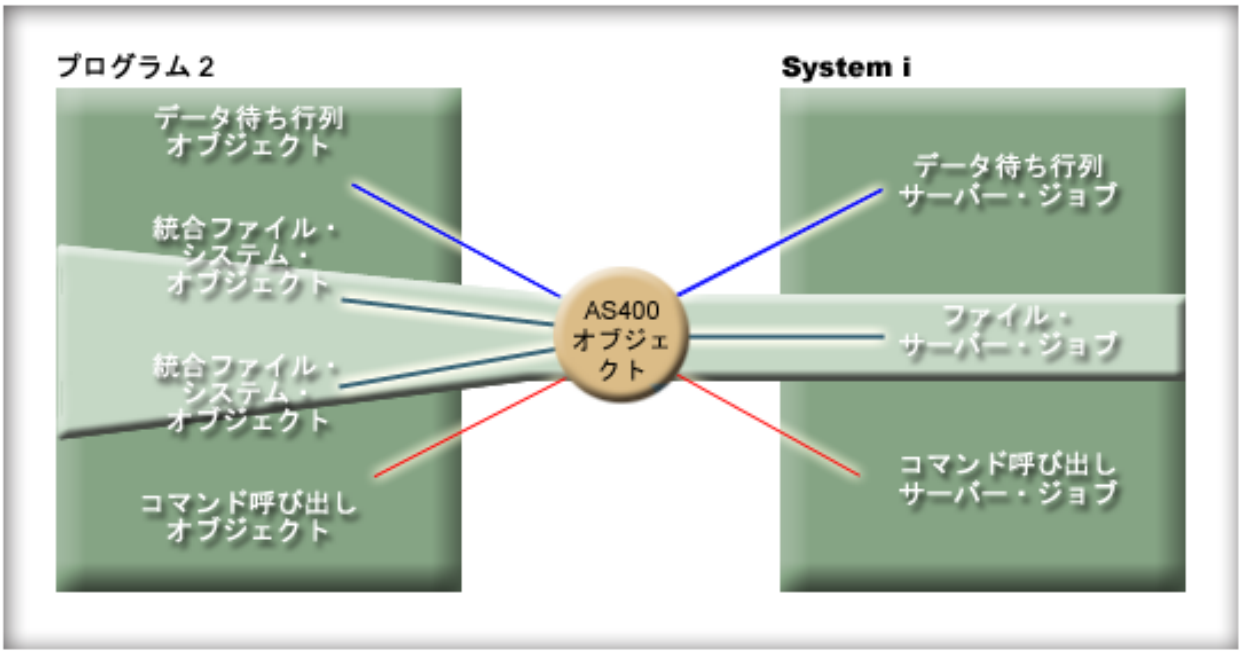


この例では、Java プログラムが接続を切断する方法を示します。

```
// 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);
```

IBM i Java 仮想マシン

IBM Toolbox for Java クラスは、IBM Developer Kit for Java (IBM i) 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 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。

さまざまな Java プラットフォーム用の IBM i ・サポートについての詳細は、複数の JDK のサポートを参照してください。

IBM i 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 (IBM i) の Java 仮想マシン (JVM) で実行されているときには、常に、サーバーの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IBM i リソースである
– IBM i JVM 上で実行しながら、別の IBM i サーバー上の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 IBM i サーバーが別の IBM i サーバーのクライアントになっている場合も含む)、IBM i 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のデータ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** - IBM i サーバー上で IBM Toolbox for Java クラス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 IBM i サーバーだけでなく他のサーバーに対しても接続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 IBM i 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は、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 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。しかし、IBM Toolbox for Java クラスを使用することによって、**サーバー**への可搬性を失うことになります。

  可搬性が重要であるか、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** - IBM i 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 IBM i サーバー** - プログラムが IBM i サーバーで実行されており、別の IBM i 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 IBM i 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 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。

IBM i Java 仮想マシン上での IBM Toolbox for Java クラスの実行

IBM Toolbox for Java クラスを、IBM Developer Kit for Java (IBM i) Java 仮想マシン (JVM) 上で実行するには、特別な考慮事項がいくつかあります。

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として、次のいずれかを使用します。

- IBM Toolbox for Java CommandCall クラス
- java.lang.Runtime.exec メソッド

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 メソッド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

IBM i 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 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 (IBM i 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 (IBM i Access for Windows など) が必要になります。

JDBC

IBM i 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 ProgramCall クラスには、任意のサーバー・プログラムを呼び出せるという利点があります。

JNI では、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。

IBM i 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 Toolbox for Java (IBM i) Java 仮想マシン (JVM) で実行されている場合、AS400 オブジェクトではシステム名、ユーザー ID、およびパスワードに対して特殊値を使用できます。

プログラムを IBM i 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。 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・パスワードを拡張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。
- 特殊値***current** は、あるサーバーの IBM i JVM で実行されている Java プログラムが、別の IBM i 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。

例

以下の例では、IBM i 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: IBM i JVM が Java プログラムを実行している場合、AS400 オブジェクトを作成する

Java プログラムが IBM i 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。

注: 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 IBM i 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
// IBM i 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: 異なるユーザー ID およびパスワードを使用して、ジョブを開始したプログラムから現在のサーバーに接続する Java プログラムは、 IBM i 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 Java 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: 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異なるサーバーに接続する

あるサーバーで実行されている Java プログラムが、他の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 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。

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")
```

AS400 オブジェクトのユーザー ID とパスワード値に関する要約:

AS400 オブジェクトのユーザー ID とパスワード値が、サーバー上で実行されている Java プログラムとクライアント上で実行されている Java プログラムを比較してどのように異なる方法で処理されるかを、次の表に要約します。

AS400 オブジェクト上の値	サーバー上で実行している Java プログラム	クライアント上で実行している Java プログラム
設定されていないシステム名、ユーザー ID、およびパスワード	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する	システム、ユーザー ID、およびパスワードを入力要求する

AS400 オブジェクト上の値	サーバー上で実行している Java プログラム	クライアント上で実行している Java プログラム
System name = localhost	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する	**エラー:** クライアント上で Java プログラムが実行している場合、 localhost は無効です。
System name = localhost User ID = *current		
System name = localhost User ID = *current Password ID = *current		
System name = "sys"	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、サーバー "sys" に接続する。 "sys" は現行のサーバーでも他のサーバーでも構いません。	ユーザー ID とパスワードが入力要求されます
System name = localhost User ID = "UID" Password ID = "PWD"	プログラムを開始したジョブのユーザー ID とパスワードではなく、 Java プログラムが指定したユーザー ID とパスワードを使用して、現行のサーバーに接続する	**エラー:** クライアント上で Java プログラムが実行していない場合、 localhost は無効です。

独立補助記憶域プール (ASP)

独立補助記憶域プール (ASP) 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。

独立 ASP 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

各独立 ASP 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、独立 ASP 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 IBM 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独立 ASP を使用します。

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、次のとおりです。

- **エラー・タイプ:** スロー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。

例: スローされる例外をキャッチする

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。

注: 法律上の重要な情報に関しては、コードに関するライセンス情報および特記事項をお読みください。

```
// 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 that you want to track...

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

エラー・イベント

多くの場合、IBM Toolbox for Java GUI コンポーネントは、例外をスローする代わりに、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スローされた例外を包むラッパーのことです。

特定のグラフィカル・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。例外がスロー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);
```

例: エラー・リスナーを定義する

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。

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、次のカスタマイズされた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);
```

関連資料

268 ページの『Vaccess クラス』
Vaccess パッケージとそのクラスは、推奨できなくなりました。代わりに、Access パッケージを Java Swing と組み合わせて使用することをお勧めします。

54 ページの『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。

関連情報

ErrorDialogAdapter Javadoc

Trace クラス

Trace クラスを使用すると、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	システムと 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 プログラムはロギングをオンにするための手段を提供します。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。

例

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

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: setTraceOn() を使用し、 log メソッドを使ってデータをログに書き込む

```
// 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: Trace を使用する

以下のコードで、トレースを使用する場合、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: コンポーネント・トレースを使用する

```
// 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 IBM Toolbox for Java and the component trace data each to separate files.
// The 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 IBM Toolbox for Java
// trace data.

Trace.setFileName("c:¥¥bit.bucket");
Trace.setFileName(myComponent1, "c:¥¥Component1.log");
```

関連情報

Trace Javadoc

IBM i 最適化

IBM Toolbox for Java ソフトウェアには 2 つのバージョンがあります。 1 つは IBM i JVM での実行用に最適化されており、もう 1 つは IBM Toolbox for Java の実行場所とは無関係に同じように稼働します。

最適化されたバージョンの IBM Toolbox for Java ソフトウェアを使って IBM i JVM 上で実行し、同じサーバーに接続する場合、サインオンの動作とパフォーマンスが改善されます。

最適化を使用可能にする

IBM Toolbox for Java は、ライセンス・プログラム 5770-SS1 (オプション 3) の一部として、以下の 2 つのディレクトリーで出荷されています。

- /QIBM/ProdData/http/public/jt400/lib には、IBM i 最適化を含まないバージョンの IBM Toolbox for Java が入っています。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。
- /QIBM/ProdData/OS400/jt400/lib には、IBM i JVM での実行用に最適化された IBM i 用のバージョンの IBM Toolbox for Java が入っています。

| **注:** IBM i 7.1 以降、ディレクトリー /QIBM/ProdData/http/public/jt400 は実際には
| /QIBM/ProdData/OS400/jt400 へのシンボリック・リンクです。 /QIBM/ProdData/http/public/jt400
| リンクは、以前のバージョンの IBM i との互換性を維持するために残されています。

| 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

最適化を含んでいるバージョンの IBM Toolbox for Java は、サーバー (システム) の名前、ユーザー ID、パスワード情報を提供する追加のオプションを IBM Toolbox for Java に追加します。

IBM i リソースへのアクセス時には、IBM Toolbox for Java 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。
- **IBM i 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。

最適化を含む IBM Toolbox for Java では、IBM i サーバーで稼働する Java プログラムが別の IBM i サーバー上のリソースにアクセスするとき、現在のジョブのユーザー ID とパスワードを使用することもできます。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。

現行のサーバーを使用するために、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");
```

例

以下の例では、最適化されたクラスを使用してサーバーにサインオンする方法を示します。

例: 異なる AS400 コンストラクターの使用時にサインオンする

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サーバー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 server 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 system 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();
```

例: 現行のジョブのユーザー ID とパスワードを使用してサインオンする

以下の例では、2 番目の IBM i サーバーを表す 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) サーバーでは、 Java プログラムを実行しているサーバー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 system using the user ID
// and password from the job on the current server.
AS400 sys = new AS400("mySystem.myCompany.com", "*current", "*current");

// Create a command call object to run a command
// on the target server.
CommandCall cmd = new CommandCall(sys,"myCommand1");


// Run the command.
cmd.run();
```

パフォーマンスの向上

IBM i によって提供される追加のクラスにより、 IBM i 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 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、 IBM i 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。 IBM i サーバーでの実行時には、IBM Toolbox for Java に付属する JDBC ドライバーではなく、 IBM i 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。

IBM i API の直接呼び出し

IBM Toolbox for Java の以下のクラスは、要求を実行するためにサーバー・プログラムを呼び出すのではなく、 IBM i API を直接呼び出すため、パフォーマンスが向上します。

- 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サーバーが判別し、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

言語特定ストリングの変更

言語特定ストリング・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。サーバーの場合、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 とシステム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 IBM i のコード・ペ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 は、ほとんどすべてのテキスト変換を Java に任せていました。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。どの時点でも、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。

関連情報

AS400 Javadoc

BinaryConverter Javadoc

Character Converter Javadoc

クライアント・インストールおよびクラスの更新

インストールや更新を目的とする場合はたいてい、サーバーの統合ファイル・システム上のロケーションにある IBM Toolbox for Java クラスを参照することができます。

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。しかし、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、 IBM i 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。 IBM i Access for Windows は、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。

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 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 unzip 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、 Javadoc の中の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

AS400ToolboxJarMaker を使用して、以下の幾つかの方法で JAR ファイルを処理することができます。

- JAR ファイルに組み込まれているファイルの中から 1 つを解凍する
- 大きな JAR ファイルを小さい JAR ファイルに分割する
- アプリケーションが実行する必要のない IBM Toolbox for Java ファイルを除外する

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 300KB 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、 400KB 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。

JarMaker Javadoc

AS400ToolboxJarMaker Javadoc

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です。

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、言語間の不一致が発生すると、 IBM Toolbox for Java ライセンス・プログラムは、**英語のエラー・メッセージを出すことがあります。**

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。

IBM Toolbox for Java のトラブルシューティング情報  IBM 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/IBM Toolbox for Java フォーラム  IBM 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てください。このフォーラムは、他の Java プログラマーおよび時には IBM 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有効です。

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、システム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 のサポート・サービスは、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 営業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 IBM iホーム・ページ  から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 API 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。

コード例

以下のリストは、IBM Toolbox for Java 情報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
Graphical Toolbox
ReportWriter クラス
セキュリティー・クラス
プログラミングに関するヒント
Vaccess クラス
Beans
HTML クラス
リソース・クラス
サーブレット・クラス
746 ページの『例: ToolboxME』
XPCML
Commtrace クラス
PCML
RFML
簡単な例
ユーティリティー・クラス

IBM は、お客様に、すべてのプログラム・コードのサンプルを使用することができる非独占的な著作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。

強行法規で除外を禁止されている場合を除き、IBM、そのプログラム開発者、および供給者は「プログラム」および「プログラム」に対する技術的サポートがある場合にはその技術的サポートについて、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。

IBM、そのプログラム開発者、または供給者は、いかなる場合においてもその予見の有無を問わず、以下に対する責任を負いません。

1. データの喪失、または損傷。
2. 直接損害、特別損害、付随的損害、間接損害、または経済上の結果的損害
3. 逸失した利益、ビジネス上の収益、あるいは節約すべかりし費用

国または地域によっては、法律の強行規定により、上記の責任の制限の一部あるいはすべてが適用されない場合があります。

例: アクセス・クラス

このセクションでは、IBM Toolbox for Java 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
- 例: ProgramCall と共に AS400DataType クラスを使用する

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
- 例: KeyedDataQueue を使用して待ち行列に項目を置く

- 例: KeyedDataQueue を使用して待ち行列から項目を取り出す

デジタル証明書

- 例: ユーザーが持っているデ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: スロー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 クラスを使用する

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: 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: SpooledFile.copy() を使用してスプール・ファイルをコピーする
- 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: スプール・ファイルを読み取り変換する
- 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

SystemStatus

- 例: SystemStatus クラスでキャッシュを使用する

SystemPool

- 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

UserGroup

- 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

UserSpace

- 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。

コードに関するライセンス情報および特記事項

IBM は、お客様に、すべてのプログラム・コードのサンプルを使用することができる非独占的な著作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。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

こ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、いかなる保証も適用されません。商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任の保証の適用も一切ありません。

例: CommandCall を使用する

この IBM Toolbox for Java プログラム例は、サーバー名と実行するコマンドをユーザーにプロンプトでたずねてから、コマンド結果を出力します。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////////////////////
//
// 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 "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

この IBM Toolbox for Java プログラム例は、AS400ConnectionPool を使用して、システムへの接続を作成します。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////
//
// AS400ConnectionPooling example.  This program uses an
// AS400ConnectionPool to create connections to an
// IBM i 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, and so on.))
    // 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 because 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: AS400JDBCManagedConnectionPoolDataSource クラスの使用

以下の例は、AS400JDBCManagedConnectionPoolDataSource クラスの使用を説明しています。AS400JDBCManagedConnectionPoolDataSource クラスは、ユーザー・アプリケーションが独自の管理コードをインプリメントする手間を省くことによって、接続プールの保守を簡素化します。

注: このコード例を使用することによって、お客様は 811 ページの『コードに関するライセンス情報および特記事項』 の条件に同意されたものとします。

例 1

この簡単な例は、AS400JDBCManagedConnectionPoolDataSource クラスの基本的な用法を示しています。

```
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.sql.DataSource;

import com.ibm.as400.access.AS400JDBCManagedConnectionPoolDataSource;
import com.ibm.as400.access.AS400JDBCManagedDataSource;


public class TestJDBCConnPoolSnippet
{
    void test()
    {
        AS400JDBCManagedConnectionPoolDataSource cpds0 = new AS400JDBCManagedConnectionPoolDataSource();

        // Set general datasource properties.  Note that both connection pool datasource (CPDS) and managed
               // datasource (MDS) have these properties, and they might have different values.
        cpds0.setServerName(host);
        cpds0.setDatabaseName(host);//iasp can be here
        cpds0.setUser(userid);
        cpds0.setPassword(password);


        cpds0.setSavePasswordWhenSerialized(true);

        // Set connection pooling-specific properties.
        cpds0.setInitialPoolSize(initialPoolSize_);
        cpds0.setMinPoolSize(minPoolSize_);
        cpds0.setMaxPoolSize(maxPoolSize_);
        cpds0.setMaxLifetime((int)(maxLifetime_/1000));  // convert to seconds
        cpds0.setMaxIdleTime((int)(maxIdleTime_/1000));  // convert to seconds
        cpds0.setPropertyCycle((int)(propertyCycle_/1000));  // convert to seconds
        //cpds0.setReuseConnections(false);  // do not re-use connections

        // Set the initial context factory to use.
        System.setProperty(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.fscontext.RefFSContextFactory");


        // Get the JNDI Initial Context.
        Context ctx = new InitialContext();

        // Note: The following is an alternative way to set context properties locally:
        //   Properties env = new Properties();
        //   env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.fscontext.RefFSContextFactory");
        //   Context ctx = new InitialContext(env);

        ctx.rebind("mydatasource", cpds0);  // We can now do lookups on cpds, by the name "mydatasource".

        // Create a standard DataSource object that references it.

        AS400JDBCManagedDataSource mds0 = new AS400JDBCManagedDataSource();
        mds0.setDescription("DataSource supporting connection pooling");
```

```
        mds0.setDataSourceName("mydatasource");
        ctx.rebind("ConnectionPoolingDataSource", mds0);

        DataSource dataSource_ = (DataSource)ctx.lookup("ConnectionPoolingDataSource");

        AS400JDBCManagedDataSource mds_ = (AS400JDBCManagedDataSource)dataSource_;

        boolean isHealthy = mds_.checkPoolHealth(false);  //check pool health

        Connection c = dataSource_.getConnection();

    }

}
```

例 2

この例は、AS400JDBCManagedConnectionPoolDataSource クラスの用法をさらに詳しく示しています。

```
import java.awt.TextArea;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.OutputStream;
import java.io.PrintStream;
import java.util.Vector;
import java.util.Properties;

import java.sql.Connection;
import javax.sql.DataSource;
import java.sql.ResultSet;
import java.sql.Statement;
import javax.naming.*;
import java.util.Date;
import java.util.ArrayList;
import java.util.Random;
import com.ibm.as400.access.AS400;
import com.ibm.as400.access.AS400JDBCManagedConnectionPoolDataSource;
import com.ibm.as400.access.AS400JDBCManagedDataSource;
import com.ibm.as400.access.Trace;


public class TestJDBCConnPool
{

    private static final boolean DEBUG = false;

    // If you turn this flag on, be sure to also turn on the following flag:
    //   AS400JDBCConnection.TESTING_THREAD_SAFETY.
    private static final boolean TESTING_THREAD_SAFETY = false;

    private static String userid;
    private static String password;
    private static String host;

    // Note: For consistency, all time values are stored units of milliseconds.
    private int initialPoolSize_;  // initial # of connections in pool
    private int minPoolSize_;      // max # of connections in pool
    private int maxPoolSize_;      // max # of connections in pool
    private long maxLifetime_;     // max lifetime (msecs) of connections in pool
    private long maxIdleTime_;     // max idle time (msecs) of available connections in pool
    private long propertyCycle_;   // pool maintenance frequency (msecs)

    private int numDaemons_;       // # of requester daemons to create
    private static long timeToRunDaemons_; // total duration (msecs) to let the daemons run
    private long daemonMaxSleepTime_;  // max time (msecs) for requester daemons to sleep each cycle
    private long daemonMinSleepTime_;  // min time (msecs) for requester daemons to sleep each cycle
    private long poolHealthCheckCycle_;  // # of msecs between calls to checkPoolHealth()

    private boolean keepDaemonsAlive_ = true;  // When this is false, the daemons shut down.

    private DataSource dataSource_;
    private AS400JDBCManagedDataSource mds_;

    private final Object daemonSleepLock_ = new Object();
```

```
    private Random random_ = new Random();

    static
    {
        try {
            Class.forName("com.ibm.as400.access.AS400JDBCDriver");
        }
        catch(Exception e){
            System.out.println("Unable to register JDBC driver.");
            System.exit(0);
        }
    }

    public static void main(String[] args)
    {
        host = args[0];
        userid = args[1];
        password = args[2];
        timeToRunDaemons_ = (new Integer(args[3])).intValue() * 1000; //milliseconds
        //args[3]=time to run in seconds
        TestJDBCConnPool cptest = new TestJDBCConnPool();

        cptest.setup();
        cptest.runTest();
    }

    public void setup()
    {
        try
        {
            if (DEBUG)
              System.out.println("TESTING_THREAD_SAFETY flag is "
                                 + (TESTING_THREAD_SAFETY ? "true" : "false"));

            if (TESTING_THREAD_SAFETY)
            {
                // Adjust values for performing thread-intensive stress testing.
                // NOTE: This assumes that the AS400JDBCConnection class has also been modified to
                             // not make actual connections to an actual server.
                // To do this, edit AS400JDBCConnection.java, changing its TESTING_THREAD_SAFETY
                             // flag to 'false', and recompile.
                minPoolSize_ = 100;
                maxPoolSize_ = 190;
                initialPoolSize_ = 150;  // this should get reset to maxPoolSize_
                numDaemons_ = 75;
                if (timeToRunDaemons_ == 0) {
                    timeToRunDaemons_ = 180*1000; // 180 seconds == 3 minutes
                }
            }
            else
            {  // Set more conservative values, as we'll be making actual connections to an
               // actual server, and we don't want to monopolize the server.
                minPoolSize_ = 5;
                maxPoolSize_ = 15;
                initialPoolSize_ = 9;
                numDaemons_ = 4;
                if (timeToRunDaemons_ == 0) {
                    timeToRunDaemons_ = 15*1000; // 15 seconds
                }
            }
            maxLifetime_ = (int)timeToRunDaemons_ / 3;
            maxIdleTime_ = (int)timeToRunDaemons_ / 4;
            propertyCycle_ = timeToRunDaemons_ / 4;
            poolHealthCheckCycle_ = Math.min(timeToRunDaemons_ / 4, 20*60*1000);
            // at least once every 20 minutes (more frequently if shorter run-time)
            daemonMaxSleepTime_ = Math.min(timeToRunDaemons_ / 3, 10*1000);
            // at most 10 seconds (less if shorter run-time)
            daemonMinSleepTime_ = 20;  // milliseconds

            if (DEBUG)
               System.out.println("setup: Constructing "
                             + "AS400JDBCManagedConnectionPoolDataSource (cpds0)");
            AS400JDBCManagedConnectionPoolDataSource cpds0 = new AS400JDBCManagedConnectionPoolDataSource();
// Set datasource properties.  Note that both CPDS and MDS have these
            // properties, and they might have different values.
            cpds0.setServerName(host);
            cpds0.setDatabaseName(host);//iasp can be here
            cpds0.setUser(userid);
```

```
        cpds0.setPassword(password);

        cpds0.setSavePasswordWhenSerialized(true);

        // Set connection pooling-specific properties.
        cpds0.setInitialPoolSize(initialPoolSize_);
        cpds0.setMinPoolSize(minPoolSize_);
        cpds0.setMaxPoolSize(maxPoolSize_);
        cpds0.setMaxLifetime((int)(maxLifetime_/1000));  // convert to seconds
        cpds0.setMaxIdleTime((int)(maxIdleTime_/1000));  // convert to seconds
        cpds0.setPropertyCycle((int)(propertyCycle_/1000));  // convert to seconds
        //cpds0.setReuseConnections(false);  // don't re-use connections

        // Set the initial context factory to use.
        System.setProperty(Context.INITIAL_CONTEXT_FACTORY,
                           "com.sun.jndi.fscontext.RefFSContextFactory");

        // Get the JNDI Initial Context.
        Context ctx = new InitialContext();

        // Note: The following is an alternative way to set context properties locally:
        //   Properties env = new Properties();
        //   env.put(Context.INITIAL_CONTEXT_FACTORY,
        //           "com.sun.jndi.fscontext.RefFSContextFactory");
        //   Context ctx = new InitialContext(env);

        ctx.rebind("mydatasource", cpds0);
                  // We can now do lookups on cpds, by the name"mydatasource".

        if (DEBUG)
          System.out.println("setup: lookup(\"mydatasource\"" + ")");
       //   AS400JDBCManagedConnectionPoolDataSource cpds1 =
       //   (AS400JDBCManagedConnectionPoolDataSource)ctx.lookup("mydatasource");
       // if (DEBUG) System.out.println("setup: cpds1.getUser() == |" + cpds1.getUser() + "|");

        // Create a standard DataSource object that references it.

        if (DEBUG)
           System.out.println("setup: Constructing AS400JDBCManagedDataSource (mds0)");
        AS400JDBCManagedDataSource mds0 = new AS400JDBCManagedDataSource();
        mds0.setDescription("DataSource supporting connection pooling");
        mds0.setDataSourceName("mydatasource");
        ctx.rebind("ConnectionPoolingDataSource", mds0);

        if (DEBUG)
          System.out.println("setup: lookup(\"ConnectionPoolingDataSource\"" + ")");
        dataSource_ = (DataSource)ctx.lookup("ConnectionPoolingDataSource");
        //dataSource_.setLogWriter(output_);
        if (DEBUG)
          System.out.println("setup: dataSource_.getUser() == |" +
                ((AS400JDBCManagedDataSource)dataSource_).getUser() + "|");

        mds_ = (AS400JDBCManagedDataSource)dataSource_;
    }
    catch (Exception e)
    {
        e.printStackTrace();
        System.out.println("Setup error during Trace file creation.");
    }
}

void displayConnectionType(Connection conn, boolean specifiedDefaultId)
{
    if (conn instanceof com.ibm.as400.access.AS400JDBCConnectionHandle)
        System.out.print("("+ (specifiedDefaultId ? "+" : "-") + "P)");
    else
        System.out.print("("+ (specifiedDefaultId ? "+" : "-") + "NP)");
}


/**
 *  Gets and returns connections from and to a connection pool for a while.
 **/
public void runTest()
{
    boolean ok = true;
    try
    {
        System.out.println("Started test run at " + new Date());
```

```
if (DEBUG)
  System.out.println("Checking health just after datasource creation "
             + "(we expect that the pool does not exist yet) ...");
if (mds_.checkPoolHealth(true)) {
    ok = false;
    System.out.println("\nERROR: Pool exists prior to first getConnection().");
}

// Verify some setters/getters for JDBC properties.
System.out.println("Verifying setters/getters ...");

mds_.setAccess("read only");
if (!mds_.getAccess().equals("read only")) {
    ok = false;
    System.out.println("\nERROR: getAccess() returned unexpected value: "
            + "|" + mds_.getAccess()+"|");
}

boolean oldBool = mds_.isBigDecimal();
boolean newBool = (oldBool ? false : true);
mds_.setBigDecimal(newBool);
if (mds_.isBigDecimal() != newBool) {
    ok = false;
    System.out.println("\nERROR: isBigDecimal() returned unexpected value: "
                        + "|"+mds_.isBigDecimal()+"|");
}
mds_.setBigDecimal(oldBool);

int oldInt = mds_.getBlockCriteria();
int newInt = (oldInt == 2 ? 1 : 2);
mds_.setBlockCriteria(newInt);
if (mds_.getBlockCriteria() != newInt) {
    ok = false;
    System.out.println("\nERROR: getBlockCriteria() returned unexpected value: "
                       + "|"+mds_.getBlockCriteria()+"|");
}
mds_.setBlockCriteria(oldInt);

// Verify some setters and getters for socket properties.

oldBool = mds_.isKeepAlive();
newBool = (oldBool ? false : true);
mds_.setKeepAlive(newBool);
if (mds_.isKeepAlive() != newBool) {
    ok = false;
    System.out.println("\nERROR: isKeepAlive() returned unexpected value: "
                       + "|"+mds_.isKeepAlive()+"|");
}
mds_.setKeepAlive(oldBool);

oldInt = mds_.getReceiveBufferSize();
newInt = (oldInt == 256 ? 512 : 256);
mds_.setReceiveBufferSize(newInt);
if (mds_.getReceiveBufferSize() != newInt) {
    ok = false;
    System.out.println("\nERROR: getReceiveBufferSize() returned unexpected value: "
                       + "|"+mds_.getReceiveBufferSize()+"|");
}
mds_.setReceiveBufferSize(oldInt);


System.out.println("CONNECTION 1");
Object o = dataSource_.getConnection();
System.out.println(o.getClass());
System.out.println("******LOOK ABOVE*******");
Connection c1 = dataSource_.getConnection();

if (DEBUG)
  displayConnectionType(c1, true);

if (DEBUG)
  System.out.println("Checking health after first getConnection() ...");
if (!mds_.checkPoolHealth(true)) {
    ok = false;
    System.out.println("\nERROR: Pool is not healthy after first getConnection().");
}

if (!TESTING_THREAD_SAFETY)
```

```
{
    try
    {
        c1.setAutoCommit(false);
        if (DEBUG)
          System.out.println("SELECT * FROM QIWS.QCUSTCDT");
        Statement s = c1.createStatement();
        ResultSet rs = s.executeQuery("SELECT * FROM QIWS.QCUSTCDT");
        while (rs.next ()) {
            if (DEBUG)
              System.out.println(rs.getString(2));
        }
        rs.close();
        s.close();
    }
    catch (Exception e) {
        e.printStackTrace();
        if (DEBUG)
          System.out.println("Checking health after fatal connection error ...");
        if (!mds_.checkPoolHealth(true)) {
          ok = false;
          System.out.println("\nERROR: Pool is not healthy after fatal connection "
                             + "error.");
        }
    }
}

System.out.println("CONNECTION 2");
Connection c2 = dataSource_.getConnection(userid, password);
if (DEBUG)
  displayConnectionType(c2, false);
System.out.println("CONNECTION 3");
Connection c3 = dataSource_.getConnection();
if (DEBUG)
  displayConnectionType(c3, true);
c1.close();

if (DEBUG)
  System.out.println("Checking health after first close() ...");
if (!mds_.checkPoolHealth(true)) {
    ok = false;
    System.out.println("\nERROR: Pool is not healthy after first close().");
}

System.out.println("CONNECTION 4");
Connection c4 = dataSource_.getConnection();
if (DEBUG) displayConnectionType(c4, true);

c1.close();  // close this one again
c2.close();
c3.close();
c4.close();

if (DEBUG) System.out.println("Checking health after last close() ...");
if (!mds_.checkPoolHealth(true)) {
    ok = false;
    System.out.println("\nERROR: Pool is not healthy after last close().");
}

// Start the test daemons.
System.out.println("Starting test daemons");
startThreads();

// Run the test daemons for a while; check pool health periodically.

long startTime = System.currentTimeMillis();
long endTime = startTime + timeToRunDaemons_;
while (System.currentTimeMillis() < endTime)
{
    System.out.print("h");
    // Let the daemons run for a while, then check pool health.
    try {
        Thread.sleep(poolHealthCheckCycle_);
    }
    catch (InterruptedException ie) {}
    if (!mds_.checkPoolHealth(true)) {
        ok = false;
        System.out.println("\nERROR: Pool is not healthy after test daemons started.");
    }
```

```
        }

        // Stop the test daemons.
        System.out.println("\nStopping test daemons");
        stopThreads();

        if (DEBUG)
          System.out.println("Checking health after connectionGetter daemons have run...");
        if (!mds_.checkPoolHealth(true)) {
            ok = false;
            System.out.println("\nERROR: Pool is not healthy after test daemons stopped.");
        }

        if (!TESTING_THREAD_SAFETY)
        {
            System.out.println("CONNECTION 5");
            Connection c = dataSource_.getConnection();
            if (DEBUG) displayConnectionType(c, true);
            c.setAutoCommit(false);
            if (DEBUG) System.out.println("SELECT * FROM QIWS.QCUSTCDT");
            Statement s = c.createStatement();
            ResultSet rs = s.executeQuery("SELECT * FROM QIWS.QCUSTCDT");
            while (rs.next ()) {
                if (DEBUG) System.out.println(rs.getString(2));
            }
            rs.close();
            s.close();
            c.close();            }

        System.out.println("\nClosing the pool...");
        mds_.closePool();

        if (DEBUG)
          System.out.println("Checking health after pool closed ...");
        Trace.setTraceJDBCOn(true);  // make sure the final stats get printed out
        Trace.setTraceOn(true);
        if (!mds_.checkPoolHealth(true)) {
            ok = false;
            System.out.println("\nERROR: Pool is not healthy after pool closed.");
        }

        System.out.println();
        if(ok==true)
            System.out.println("test ran ok");
        else
            System.out.println("test failed");
    }
    catch (Exception e)
    {
        System.out.println(e);
        e.printStackTrace();
    }
    finally {
        System.out.println("Ended test at " + new Date());
    }
}

void startThreads()
{
    // Create a bunch of threads that call getConnection().
    Thread[] threads = new Thread[numDaemons_];
    for (int i=0; i<numDaemons_; i++)
    {
        ConnectionGetter getter;
        // Flip a coin to see if this daemon will specify the default uid, or unique uid.
        if (random_.nextBoolean())
        {
            getter = new ConnectionGetter(userid,password);
            if (TESTING_THREAD_SAFETY) {  // we can use fictional userid
                getter = new ConnectionGetter("Thread"+i, "Pwd"+i);
            }
            else {  // must use a real userid
                getter = new ConnectionGetter(userid,password);
            }
        }
        else
            getter = new ConnectionGetter(null, null);

        threads[i] = new Thread(getter, "["+i+"]");
```

```
        threads[i].setDaemon(true);
    }

    // Start the threads.
    for (int i=0; i<numDaemons_; i++)
    {
        threads[i].start();
    }
}

void stopThreads()
{
    // Tell the threads to stop.
    keepDaemonsAlive_ = false;
    synchronized (daemonSleepLock_) {
        daemonSleepLock_.notifyAll();
    }

    // Wait for the daemons to stop.
    try {
        Thread.sleep(3000);
    }
    catch (InterruptedException ie) {}
}


// ConnectionGetter ---------------------------------------------------------------

/**
     Helper class. This daemon wakes up at random intervals and either
     gets another connection from the connection pool or returns a
     previously-gotten connection to the pool.
 **/
private final class ConnectionGetter implements Runnable
{
    private String uid_;
    private String pwd_;
    private boolean useDefaultUid_;
    private long maxSleepTime_;
    private String threadName_;
    private boolean firstConnection_ = true;
    ArrayList connections_ = new ArrayList();
    // list of connections that this getter currently 'owns'.

    ConnectionGetter(String uid, String pwd) {
        uid_ = uid;
        pwd_ = pwd;
        if (uid_ == null) useDefaultUid_ = true;
        else useDefaultUid_ = false;
        maxSleepTime_ = daemonMaxSleepTime_;  // our own copy that we can adjust
    }

    public void run()
    {
        threadName_ = Thread.currentThread().getName();
        if (DEBUG) System.out.println("ConnectionGetter("+threadName_+") Starting up");

        try
        {
            while (keepDaemonsAlive_)
            {
                try
                {
                    // Pick a random sleep-time, between min and max values.
                    long sleepTime = Math.max((long)(maxSleepTime_ * random_.nextFloat()),
                            daemonMinSleepTime_);
                    // Note: Must never call wait(0), because that waits forever.
                    synchronized (daemonSleepLock_) {
                        try {
                            daemonSleepLock_.wait(sleepTime);
                            System.out.print(".");
                        }
                        catch (InterruptedException ie) {}
                    }
                    if (!keepDaemonsAlive_) break;

                    // Decide by chance whether to request another connection or return a
                    // previously-obtained connection.
                    Connection conn;
```

```
                    if (random_.nextBoolean())  // Leave the decision to chance.
                    { // Request another connection.
                        if (useDefaultUid_)
                        {
                            if (DEBUG)
                              System.out.println("ConnectionGetter("+threadName_+") - get()");
                            conn = dataSource_.getConnection();
                        }
                        else
                        {
                            if (DEBUG)
                               System.out.println("ConnectionGetter("+threadName_+") - "
                                                  + "get("+uid_+",***)");
                            conn = dataSource_.getConnection(uid_, pwd_);
                        }

                        if (conn == null) {
                            System.out.println("ConnectionGetter("+threadName_+") ERROR: "
                                               + "getConnection() returned null");
                        }
                        else
                        {
                            // Occasionally "forget" that we own a connection, and neglect to
                            // close it.
                            // Orphaned connections should eventually exceed their maximum
                            // lifetime and get "reaped" by the connection manager.
                            float val = random_.nextFloat();
                            if (firstConnection_ || val < 0.1) {
                               // 'orphan' a few gotten connections
                               firstConnection_ = false;
                            }
                            else {
                                connections_.add(conn);
                            }
                            if (DEBUG) displayConnectionType(conn, useDefaultUid_);
                            if (conn instanceof com.ibm.as400.access.AS400JDBCConnectionHandle)
                            {  // We got a pooled connection.  Try speeding up our cycle time.
                                if (maxSleepTime_ > 100)
                                  maxSleepTime_--;
                                else
                                  maxSleepTime_ = 100;
                            }
                            else
                            {  // We didn't get a pooled connection.  That means that the pool
                               // must be at capacity.  Slow down our cycle time a bit.
                                maxSleepTime_ = maxSleepTime_ + 50;
                            }
                        }
                    }
                    else { // Close a connection that we currently own.
                        if (connections_.size() != 0) {
                            conn = (Connection)connections_.remove(0);
                            conn.close();
                        }
                    }
                }  // inner try
                catch (Exception e)
                {
                    e.printStackTrace();
                }
            } // outer while
        }  // outer try
        finally
        {
            if (DEBUG)
              System.out.println("ConnectionGetter("+threadName_+") Stopping");
            // Return all the connections that I still have in my list.
            for (int i=0; i<connections_.size(); i++) {
                Connection conn = (Connection)connections_.remove(0);
                try { conn.close(); } catch (Exception e) { e.printStackTrace(); }
            }

        }
    }

}  // internal class 'ConnectionGetter'
}
```

例: FieldDescription、RecordFormat、および Record クラスを使用する

以下の例は、データ待ち行列を指定してIBM Toolbox for Java FieldDescription、 RecordFormat および Record クラスを使用する方法を示しています。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

例: FieldDescription クラスを使用する

FieldDescription クラスを使って、データ待ち行列の項目を構成するさまざまなタイプのデータを記述することができます。以下の例では、データ待ち行列の項目に対して次のような形式がとられることが前提になっています。

```
Message number    Sender    Time sent      Message text     Reply required
      |             |           |              |                |
   bin(4)        char(50)    char(8)         char(1024)        char(1)
```

```
// Create field descriptions for the entry data
BinaryFieldDescription msgNumber = new BinaryFieldDescription(new AS400Bin4(),
                                   "msgnum");
CharacterFieldDescription sender = new CharacterFieldDescription(new AS400Text(50),
                                   "sender");
CharacterFieldDescription timeSent = new CharacterFieldDescription(new AS400Text(8),
                                    "timesent");
CharacterFieldDescription msgText = new CharacterFieldDescription(new AS400Text(1024),
                                    "msgtext");
CharacterFieldDescription replyRequired = new CharacterFieldDescription(new AS400Text(1),
                                          "replyreq");
```

RecordFormat クラスを使用する

RecordFormat クラスを使って、データ待ち行列の項目を構成するデータを記述することができます。

例: RecordFormat を定義して動的に使用する

以下の例では RecordFormat クラスを使ってデータ待ち行列の項目の形式を記述してから、それをレコードの検索に動的に使用します。

```
RecordFormat entryFormat = new RecordFormat();
// Describe the fields in an entry on the data queue
entryFormat.addFieldDescription(msgNumber);
entryFormat.addFieldDescription(sender);
entryFormat.addFieldDescription(timeSent);
entryFormat.addFieldDescription(msgText);
entryFormat.addFieldDescription(replyRequired);

// Get a record based on the format of the entries on the data queue
Record rec = entryFormat.getNewRecord();
```

例: RecordFormat を静的に定義する

以下の例では、レコード様式を静的に定義します。これを定義すれば、レコード様式を何回もコーディングし直さなくても 1 つの様式を多数のプログラムで使うことができます。

```
public class MessageEntryFormat extends RecordFormat
{
  // The field descriptions are contained in the class
  static BinaryFieldDescription msgNumber = new BinaryFieldDescription(new AS400Bin4(),
                                            "msgnum");
  static CharacterFieldDescription sender = new CharacterFieldDescription(new AS400Text(50),
                                            "sender");
  static CharacterFieldDescription timeSent = new CharacterFieldDescription(new AS400Text(8),
```

```
                                            "timesent");
  static CharacterFieldDescription msgText = new CharacterFieldDescription(new AS400Text(1024),
                                            "msgtext");
  static CharacterFieldDescription replyRequired = new CharacterFieldDescription(new AS400Text(1),
                                                "replyreq");

  public MessageEntryFormat()
  {
    // We will name this format for posterity
    super("MessageEntryFormat");
    // Add the field descriptions
    addFieldDescription(msgNumber);
    addFieldDescription(sender);
    addFieldDescription(timeSent);
    addFieldDescription(msgText);
    addFieldDescription(replyRequired);
  }
}
```

例: RecordFormat を静的に使用する

以下の例は、静的に定義された RecordFormat をどのように Java プログラムで使用できるかを示しています。

```
MessageEntryFormat entryFormat = new MessageEntryFormat();
// Get a record based on the format of the entries on the data queue
Record rec = entryFormat.getNewRecord();
```

Record クラスを使用する

Record クラスを使って、データ待ち行列の項目の個々のフィールドにアクセスすることができます。

例: 汎用 Record オブジェクトを使用する

```
// Instantiate our data queue object
DataQueue dq = new DataQueue(new AS400(), "/qsys.lib/mylib.lib/myq.dtaq");

// Read an entry
DataQueueEntry dqEntry = null;
try
{
  dqEntry = dq.read();
}
catch(Exception e)
{
  // Handle any exceptions
}

// Get a record object from our record format, initializing it with the data from the entry we
// just read.   Record rec = entryFormat.getNewRecord(dqEntry.getData());

// Output the complete entry as a String.  The contents of the record are converted to Java Objects
// based on the record format of the entry.
System.out.println(rec.toString());
// Get the contents of the individual fields of the entry.  Each field's contents are converted to
// a Java Object.
Integer num = (Integer)rec.getField(0);    // Retrieve contents by index
String s = (String)rec.getField("sender");// Retrieve contents by field name
String text = (String)rec.getField(3);    // Retrieve the message text
// Output the data
System.out.println(num + " " + s + " " + text);
```

例: 特定の Record オブジェクトを使用する

さらに、このデータ待ち行列の形式に固有の Record を静的に定義してからそれを使えば、 getField() や setField() よりも意味のある名前を付けられる get() および set() メソッドをフィールドに指定することができます。また、静的に定義された個々の Record を使えば、オブジェクトではなく基本の Java タイプを戻すことができ、しかも、ユーザーの戻りタイプを特定することもできます。

以下の例では正しい Java オブジェクトを明示的にキャストする必要があることに注意してください。

```
public class MessageEntryRecord extends Record
{
  static private RecordFormat format = new MessageEntryFormat();

  public MessageEntryRecord()
  {
    super(format);
  }

  public int getMessageNumber()
  {
    // Return the message number as an int.  Note: We know our record format and therefore
    // know the names of our fields.  It is safest to get the field by name in case a field
    // has been inserted into the format unbeknownst to us.
    return ((Integer)getField("msgnum")).intValue();
  }

  public String getMessageText()
  {
    // Return the text of the message
    return (String)getField("msgtext");
  }

  public String getSender()
  {
    // Return the sender of the message
    return (String)getField("sender");
  }

  public String getTimeSent()
  {
    // Return the sender of the message
    return (String)getField("timesent");
  }

  // We could add setters here
}
```

例: 新規の MessageEntryRecord を戻す

新規の MessageEntryRecord を戻すには、 MessageEntryFormat クラス (上記の例) 内の getNewRecord() メソッドをオーバーライドする必要があります。メソッドをオーバーライドするには、以下を MessageEntryFormat クラスに追加します。

```
  public Record getNewRecord(byte[] data)
  {
    Record r = new MessageEntryRecord();
    r.setContents(data);
    return r;
  }
```

新規の getNewRecord() メソッドを追加し終わったら、以下のようにして MessageEntryRecord を使ってデータ待ち行列の項目を解釈することができます。

```
// Get a record object from our record format, initializing it with the data from the entry we
// just read. Note the use of the new overridden method getNewRecord().
MessageEntryRecord rec = (MessageEntryRecord)entryFormat.getNewRecord(dqEntry.getData());
```

```
  // Output the complete entry as a String.  The contents of the record are converted to Java Objects
  // based on the record format of the entry.
  System.out.println(rec.toString());
  // Get the contents of the individual fields of the entry.  Each field's contents are converted to
  // a Java Object.
  int num = rec.getMessageNumber();    // Retrieve the message number as an int
  String s = rec.getSender();         // Retrieve the sender
  String text = rec.getMessageText(); // Retrieve the message text
  // Output the data
  System.out.println(num + " " + s + " " + text);
```

例: DataQueue クラスを使用して待ち行列にデータを書き込む

この例は、レコードおよびレコード・フォーマット・クラスを使用して、データを待ち行列に入れます。ストリング・データは、Unicode から EBCDIC に変換され、数字は、Java からシステム・フォーマットに変換されます。データはデータ待ち行列に変換されるので、サーバー・プログラム、IBM i プログラム、または別の Java プログラムが、項目を読み取ることができます。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// 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.  Because data
// is converted, entries can be read by a server program,
// an Access for Windows program, or 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

このプログラムは、データ待ち行列クラスを使用して、サーバー上のデータ待ち行列から項目を読み取ります。その項目は、DQProducer サンプル・プログラムでキューに入れられたものです。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// 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);
   }
}
```

データ型の使用例

IBM Toolbox for Java AS400DataType クラスを ProgramCall と一緒に使用して、プログラム・パラメーター用のデータの指定や、プログラム・パラメーターに戻されたデータの解釈を行うことができます。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

例: ProgramCall と共に AS400DataType クラスを使用する

以下の例は、ProgramCall を使用して QUSRMBRD "Retrieve Member Description" というシステム API を呼び出すことによって AS400DataType クラスを使用する方法を示しています。 QUSRMBRD API は、データベース・ファイル内の特定のメンバーに関する記述を検索します。この例に続いて、必要な QUSRMBRD パラメーターおよびこの例で検索する情報のタイプが表にリストされます。

```
// Create a ProgramCall object.  We will set the program name and
// parameter list later.
ProgramCall qusrmbrd = new ProgramCall(new AS400());

// Create an empty program parameter list
ProgramParameter[] parms = new ProgramParameter[6];

// Create AS400DataTypes to convert our input parameters from Java types
// to server data
AS400Bin4 bin4 = new AS400Bin4();

// We need a separate AS400Text object for each parameter with a
// different length because the AS400Text class requires the length to
// be specified.
AS400Text char8Converter = new AS400Text(8)
AS400Text char20Converter = new AS400Text(20);
AS400Text char10Converter = new AS400Text(10);
AS400Text char1Converter = new AS400Text(1);

// Populate our parameter list; we use the AS400DataType objects to
// convert our Java values to byte arrays containing server data.

// For the output parameter we need only specify how many bytes will
// be returned
parms[0] = new ProgramParameter(135);
parms[1] = new ProgramParameter(bin4.toBytes(new Integer(135)));
parms[2] = new ProgramParameter(char8Converter.toBytes("MBRD0100"));
parms[3] = new ProgramParameter(char20Converter.toBytes("MYFILE    MYLIB     "));
```

```
    parms[4] = new ProgramParameter(char10COnverter.toBytes("MYMEMBER  "));
    parms[5] = new ProgramParameter(char1Converter.toBytes("0"));

    // Set the program name and parameter list
    qusrmbrd.setProgram("/qsys.lib/qusrmbrd.pgm", parms);

    // Call the program
    try
    {
      qusrmbrd.run();
    }
    catch(Exception e)
    {
      // Handle any exceptions
    }

    // Get the information retrieved. Note that this is raw server data.
    byte[] receiverVar = parms[0].getOutputData();


// We need this to convert the time and date data
    AS400Text char13Converter = new AS400Text(13);

// We need this to convert the text description data
    AS400Text char50Converter = new AS400Text(50);

    // Create an AS400Structure to handle the returned information
    AS400DataType[] dataTypeArray = new AS400DataType[11];
    dataTypeArray[0] = bin4;
    dataTypeArray[1] = bin4;
    dataTypeArray[2] = char10Converter;
    dataTypeArray[3] = char10Converter;
    dataTypeArray[4] = char10Converter;
    dataTypeArray[5] = char10Converter;
    dataTypeArray[6] = char10Converter;
    dataTypeArray[7] = char13Converter;
    dataTypeArray[8] = char13Converter;
    dataTypeArray[9] = char50Converter;
    dataTypeArray[10] = char1Converter;
    AS400Structure returnedDataConverter = new AS400Structure(dataTypeArray);


    // Convert the data returned to an array of Java Objects using our
    // returnedDataConverter
    Object[] qusrmbrdInfo = dataConverter.toObject(receiverVar, 0);


    // Get the number of bytes returned
    Integer bytesReturned = (Integer)qusrmbrdInfo[0];
    Integer bytesAvailable = (Integer)qusrmbrdInfo[1];
    if (bytesReturned.intValue() != 135)
    {
      System.out.println("Wrong amount of information returned.");
      System.exit(0);
    }
    String fileName = (String)qusrmbrdInfo[2];
    String libName = (String)qusrmbrdInfo[3];
    String mbrName = (String)qusrmbrdInfo[4];
    String fileAttribute = (String)qusrmbrdInfo[5];
    String sourceType = (String)qusrmbrdInfo[6];
    String created = (String)qusrmbrdInfo[7];
    String lastChanged = (String)qusrmbrdInfo[8];
    String textDesc = (String)qusrmbrdInfo[9];
    String isSourceFile = (String)qusrmbrdInfo[10];

    // We will just output all the information to the screen
```

```
System.out.println(fileName + " " + libName + " " + mbrName + " " +
                   fileAttribute + sourceType + " " + created + " " +
                   lastChanged + " " + textDesc + " " + isSourceFile);
```

以下の表は、前の例で使用された QUSRMBRD API の必要パラメーターをリストします。

QUSRMBRD パラメーター	入力または出力	型	説明
レシーバー変数	出力	Char(*)	検索した情報を収容する文字バッファー
レシーバー変数の長さ	入力	Bin(4)	レシーバー変数用に用意されている文字バッファーの長さ
形式名	入力	Char(8)	検索対象の情報のタイプを指定する形式。以下のいずれかでなければなりません。 • MBRD0100 • MBRD0200 • MBRD0300 以下の例では、MBRD0100 を指定します。
データベース修飾ファイル名	入力	Char(20)	修飾ファイル名。これは、ファイル名 (10 文字になるようにブランクを埋め込まれる) の後に、ライブラリー名 (10 文字になるようにブランクが埋め込まれる) が続きます。特殊値 *CURLIB および *LIBL をライブラリー名に使うことができます。
データベース・メンバー名	入力	Char(10)	ブランクを埋め込んで 10 文字にするメンバーの名前。特殊値 *FIRST および *LAST を使うことができます。
変更処理	入力	Char(1)	変更を処理するかどうか。0 は変更を処理しないことを示します。ここではこの値を指定します。

以下の表は、この例で検索する情報のタイプをリストします (前の例の中で指定されているように、形式 MBRD0100 に基づきます)。

検索される情報	型
戻されるバイト	Bin(4)
使用可能なバイト	Bin(4)
データベース・ファイル名	Char(10)

検索される情報	型
データベース・ファイル・ライブラリー名	Char(10)
メンバー名	Char(10)
ファイル属性 (ファイルのタイプ: PF、LF、DDMF)	Char(10)
ソース・タイプ (ソース・ファイルの場合はソース・メンバーのタイプ)	Char(10)
作成日時	Char(13)
ソースの最終変更日時	Char(13)
メンバー・テキスト記述	Char(50)
ソース・ファイル (ファイルがソース・ファイルかどうか。0 = データ・ファイル、1 = ソース・ファイル)	Char(1)

例: KeyedDataQueue を使用する

このプログラムは、KeyedDataQueue クラスを使用して、レコードをデータ待ち行列に入れます。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////////////////////
//
// Data Queue example.  This program uses the KeyedDataQueue class to put
// records on a data queue.
//
// The key is a number and the data is a Unicode string.  This program
// shows one way to convert on int into a byte array and how to convert
// a Java string into a byte array so it can be written to the queue.
//
// This is the producer side of the producer/consumer example.  It puts work
// items on the queue for the consumer to process.
//
// Command syntax:
//    DQKeyedProducer system
//
////////////////////////////////////////////////////////////////////////////

import java.io.*;
import java.util.*;
import java.net.*;
import com.ibm.as400.access.*;

public class DQKeyedProducer 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)
```

```
{
   // The first parameter is the system that contains the data queue.

   String system = parameters[0];



   System.out.println("Priority is a numeric value. The value ranges are:");
   System.out.println("  0 -  49 = low priority");
   System.out.println(" 50 - 100 = medium priority");
   System.out.println("100 +     = high priority");
   System.out.println(" ");

   try
   {

       // Create an AS400 object for the server that has the data queue.

       AS400 as400 = new AS400(system);



       // Use CommandCall to create the library that contains the
       // data queue.

       CommandCall crtlib = new CommandCall(as400);
       crtlib.run("CRTLIB JAVADEMO");



       // Create the data queue object.

       QSYSObjectPathName name = new QSYSObjectPathName("JAVADEMO", "PRODCON2", "DTAQ");

       KeyedDataQueue dq = new KeyedDataQueue(as400, name.getPath());



       // 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.  The length of the key is four bytes, the length
       // of an entry is 96 bytes.

       try
       {
          dq.create(4, 96);
       }
       catch (Exception e) {};



       // Get the data from the user.

       System.out.print("Enter message: ");
       String message = inputStream.readLine();

       System.out.print("Enter priority: ");
       int priority = getInt();



       // While there is data to put on the queue.

       while (priority > 0)
       {
```

```
            // We want to write a java string as the entry to the queue.
            // Input the data queue is a byte array, however, so convert
            // the string to a byte array.

            byte [] byteData = message.getBytes("UnicodeBigUnmarked");



            // The key is a number.  Input to the data queue is a byte
            // array, however, so convert the int to a byte array;

            byte [] byteKey = new byte[4];
            byteKey[0] = (byte) (priority >>> 24);
            byteKey[1] = (byte) (priority >>> 16);
            byteKey[2] = (byte) (priority >>> 8);
            byteKey[3] = (byte) (priority);


            System.out.println("");
            System.out.println("Writing record to the server...");
            System.out.println("");



            // Write the record to the data queue.

            dq.write(byteKey, byteData);



            // Get the next value from the user.

            System.out.print("Enter message: ");
            message = inputStream.readLine();

            System.out.print("Enter priority: ");
            priority = getInt();
         }
      }
      catch (Exception e)
      {

         // If any of the above operations failed say the data queue
         // operation 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("  DQKeyedProducter 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("  DQKeyedProducer 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: KeyedDataQueue クラスを使用してデータ待ち行列から項目を読み取る

このプログラムは、KeyedDataQueue クラスを使用して、サーバー上のデータ待ち行列から項目を読み取ります。その項目は、DQKeyedProducer サンプル・プログラムでキューに入れられたものです。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// Keyed Data Queue example.  This program uses the KeyedDataQueue classes to
// read entries off a data queue on the server.  The entries were put on the
// queue with the DQKeyedProducer example program.
//
// The key is a number and the data is a unicode string.  This program
// shows one way to convert the byte array to a Java int and to read
// a byte array and convert it into a Java string.
//
// This is the consumer side of the producer/consumer example.  It reads
// entries off the queue and process them.
//
// Command syntax:
//    DQKeyedConsumer system
//
///////////////////////////////////////////////////////////////////////////

import java.io.*;
```

```
import java.util.*;
import java.net.*;
import com.ibm.as400.access.*;

public class DQKeyedConsumer 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)
      {

         // The first parameter is the system that contains the data queue.
         String system = parameters[0];

         // Create byte arrays for the priority boundries:
         // 100 +    = high priority
         // 50 - 100 = medium priority
         //  0 -  49 = low priority

         byte [] key0 = new byte[4];
         key0[0] = 0;
         key0[1] = 0;
         key0[2] = 0;
         key0[3] = 0;

         byte [] key50  = new byte[4];
         key50[0] = (byte) (50 >>> 24);
         key50[1] = (byte) (50 >>> 16);
         key50[2] = (byte) (50 >>> 8);
         key50[3] = (byte) (50);

         byte [] key100 = new byte[4];
         key100[0] = (byte) (100 >>> 24);
         key100[1] = (byte) (100 >>> 16);
         key100[2] = (byte) (100 >>> 8);
         key100[3] = (byte) (100);

         try
         {
             // Create an AS400 object for the server that has the data queue.
             AS400 as400 = new AS400(system);

             // Create the data queue object that represents the data queue
             // on the server.

             QSYSObjectPathName name = new QSYSObjectPathName("JAVADEMO",
                                                              "PRODCON2",
                                                              "DTAQ");
             KeyedDataQueue dq = new KeyedDataQueue(as400, name.getPath());
             KeyedDataQueueEntry DQData = null;

             try
             {
                boolean Continue = true;

                // Go until the user ends us.
                while (Continue)
                {
                   // Look for a high priority item on the queue. If one is
                   // found process that item.  Note the peek method does not
                   // remove the item if one is found. Also note the timeout
                   // is 0. If an item is not found we get control back with
                   // a null data queue entry.
                   DQData = dq.read(key100, 0, "GE");
```

```
            if (DQData != null)
            {
               processEntry(DQData);
            }

            // else no high priority item was found.  Look for a medium
            // priority item.
            else
            {
               DQData = dq.read(key50, 0, "GE");

               if (DQData != null)
               {
                  processEntry(DQData);
               }

               // else no medium priority item was found, look for a low
               // priority item.
               else
               {
                  DQData = dq.read(key0, 0, "GE");

                  if (DQData != null)
                  {
                     processEntry(DQData);
                  }

                  else
                  {
                    System.out.println("Nothing to process, will check again in 30 seconds");
                    Thread.sleep(30000);
                  }
               }
            }
         }
      }
      catch (Exception e)
      {
         // If any of the above operations failed say the data queue
         // operation failed and output the exception.
         System.out.println("Could not read from the data queue.");
         System.out.println(e);
      };

   }
   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(" DQKeyedConsumer 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("");
            System.out.println(" DQKeyedConsumer mySystem");
            System.out.println("");
            System.out.println("");
        }

        System.exit(0);
    }

    static void processEntry(KeyedDataQueueEntry DQData)
    {
        try
        {
            // The data is a string.  Get the string out of the data queue entry.
            // In the data queue entry the data is a byte array so convert the
            // entry from a byte array into a string.
            String message  = new String(DQData.getData(), "UnicodeBig");

            // The key is a byte array.  Get the key out of the data queue entry
            // and convert it into a number.
            byte [] keyData = DQData.getKey();

            int keyValue = ((keyData[0] & 0xFF) << 24) +
                           ((keyData[1] & 0xFF) << 16) +
                           ((keyData[2] & 0xFF) <<  8) +
                            (keyData[3] & 0xFF);

            // Output the entry.
            System.out.println("Priority: " + keyValue + "   message: " + message);

        }
        catch (Exception e)
        {
            // If any of the above operations failed say the data queue operation
            // failed and output the exception.

            System.out.println("Could not read from the data queue");
            System.out.println(e);
        }
    }
}
```

例: IFSFile を使用する

以下の例では、IBM Toolbox for Java IFSFile クラスを使用する方法を示します。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

- 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
- 546 ページの『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 system.
    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.
   // 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 例外をスローします。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 クラスが例外をスローしない場合でも、この IFSFile クラスは例外をスローします。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 がスローされます。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();

   System.out.println("Delete successful ");
}
catch (ExtendedIOException e)
{
   // The delete failed. Get the return code out of
   // the exception and display why the delete failed.
   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

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

このプログラム例は、IBM Toolbox for Java IFS クラスを使って、サーバー上のディレクトリーの内容をリストします。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////////////////////////
//
// 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 about 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 about 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

このプログラムは、インストール可能ファイル・システム・クラスを使って、サーバー上のディレクトリーから別のディレクトリーにファイルをコピーします。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////////////////////////
//
// 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

このプログラムは、統合ファイル・システム・クラスを使って、サーバー上のディレクトリーの内容をリストします。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////////////////////////
//
// 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

このプログラムは、IBM Toolbox for Java JDBC ドライバーを使用して、テーブルを作成し、そこにデータを取り込みます。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////////////////////////
//
// 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 SQL 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: AS400JDBCManagedConnectionPoolDataSource クラスの使用

以下の例は、AS400JDBCManagedConnectionPoolDataSource クラスの使用を説明しています。AS400JDBCManagedConnectionPoolDataSource クラスは、ユーザー・アプリケーションが独自の管理コードをインプリメントする手間を省くことによって、接続プールの保守を簡素化します。

注: このコード例を使用することによって、お客様は 811 ページの『コードに関するライセンス情報および特記事項』 の条件に同意されたものとします。

例 1

この簡単な例は、AS400JDBCManagedConnectionPoolDataSource クラスの基本的な用法を示しています。

```
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.sql.DataSource;

import com.ibm.as400.access.AS400JDBCManagedConnectionPoolDataSource;
import com.ibm.as400.access.AS400JDBCManagedDataSource;


public class TestJDBCConnPoolSnippet
{
    void test()
    {
        AS400JDBCManagedConnectionPoolDataSource cpds0 = new AS400JDBCManagedConnectionPoolDataSource();

        // Set general datasource properties.  Note that both connection pool datasource (CPDS) and managed
               // datasource (MDS) have these properties, and they might have different values.
        cpds0.setServerName(host);
        cpds0.setDatabaseName(host);//iasp can be here
        cpds0.setUser(userid);
        cpds0.setPassword(password);


        cpds0.setSavePasswordWhenSerialized(true);

        // Set connection pooling-specific properties.
        cpds0.setInitialPoolSize(initialPoolSize_);
        cpds0.setMinPoolSize(minPoolSize_);
        cpds0.setMaxPoolSize(maxPoolSize_);
        cpds0.setMaxLifetime((int)(maxLifetime_/1000));  // convert to seconds
        cpds0.setMaxIdleTime((int)(maxIdleTime_/1000));  // convert to seconds
        cpds0.setPropertyCycle((int)(propertyCycle_/1000));  // convert to seconds
        //cpds0.setReuseConnections(false);  // do not re-use connections

        // Set the initial context factory to use.
        System.setProperty(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.fscontext.RefFSContextFactory");
```

```
        // Get the JNDI Initial Context.
        Context ctx = new InitialContext();

        // Note: The following is an alternative way to set context properties locally:
        //   Properties env = new Properties();
        //   env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.fscontext.RefFSContextFactory");
        //   Context ctx = new InitialContext(env);

        ctx.rebind("mydatasource", cpds0);  // We can now do lookups on cpds, by the name "mydatasource".

        // Create a standard DataSource object that references it.

        AS400JDBCManagedDataSource mds0 = new AS400JDBCManagedDataSource();
        mds0.setDescription("DataSource supporting connection pooling");
        mds0.setDataSourceName("mydatasource");
        ctx.rebind("ConnectionPoolingDataSource", mds0);

        DataSource dataSource_ = (DataSource)ctx.lookup("ConnectionPoolingDataSource");

        AS400JDBCManagedDataSource mds_ = (AS400JDBCManagedDataSource)dataSource_;

        boolean isHealthy = mds_.checkPoolHealth(false);  //check pool health

        Connection c = dataSource_.getConnection();

    }

}
```

例 2

この例は、AS400JDBCManagedConnectionPoolDataSource クラスの用法をさらに詳しく示しています。

```
import java.awt.TextArea;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.OutputStream;
import java.io.PrintStream;
import java.util.Vector;
import java.util.Properties;

import java.sql.Connection;
import javax.sql.DataSource;
import java.sql.ResultSet;
import java.sql.Statement;
import javax.naming.*;
import java.util.Date;
import java.util.ArrayList;
import java.util.Random;
import com.ibm.as400.access.AS400;
import com.ibm.as400.access.AS400JDBCManagedConnectionPoolDataSource;
import com.ibm.as400.access.AS400JDBCManagedDataSource;
import com.ibm.as400.access.Trace;


public class TestJDBCConnPool
{

    private static final boolean DEBUG = false;

    // If you turn this flag on, be sure to also turn on the following flag:
    //   AS400JDBCConnection.TESTING_THREAD_SAFETY.
    private static final boolean TESTING_THREAD_SAFETY = false;

    private static String userid;
    private static String password;
    private static String host;

    // Note: For consistency, all time values are stored units of milliseconds.
    private int initialPoolSize_;  // initial # of connections in pool
    private int minPoolSize_;      // max # of connections in pool
    private int maxPoolSize_;      // max # of connections in pool
    private long maxLifetime_;     // max lifetime (msecs) of connections in pool
    private long maxIdleTime_;     // max idle time (msecs) of available connections in pool
```

```
private long propertyCycle_;   // pool maintenance frequency (msecs)

private int numDaemons_;       // # of requester daemons to create
private static long timeToRunDaemons_; // total duration (msecs) to let the daemons run
private long daemonMaxSleepTime_;  // max time (msecs) for requester daemons to sleep each cycle
private long daemonMinSleepTime_;  // min time (msecs) for requester daemons to sleep each cycle
private long poolHealthCheckCycle_;  // # of msecs between calls to checkPoolHealth()

private boolean keepDaemonsAlive_ = true;  // When this is false, the daemons shut down.

private DataSource dataSource_;
private AS400JDBCManagedDataSource mds_;

private final Object daemonSleepLock_ = new Object();

private Random random_ = new Random();

static
{
    try {
        Class.forName("com.ibm.as400.access.AS400JDBCDriver");
    }
    catch(Exception e){
        System.out.println("Unable to register JDBC driver.");
        System.exit(0);
    }
}

public static void main(String[] args)
{
    host = args[0];
    userid = args[1];
    password = args[2];
    timeToRunDaemons_ = (new Integer(args[3])).intValue() * 1000; //milliseconds
    //args[3]=time to run in seconds
    TestJDBCConnPool cptest = new TestJDBCConnPool();

    cptest.setup();
    cptest.runTest();
}

public void setup()
{
    try
    {
        if (DEBUG)
          System.out.println("TESTING_THREAD_SAFETY flag is "
                           + (TESTING_THREAD_SAFETY ? "true" : "false"));

        if (TESTING_THREAD_SAFETY)
        {
            // Adjust values for performing thread-intensive stress testing.
            // NOTE: This assumes that the AS400JDBCConnection class has also been modified to
                        // not make actual connections to an actual server.
            // To do this, edit AS400JDBCConnection.java, changing its TESTING_THREAD_SAFETY
                        // flag to 'false', and recompile.
            minPoolSize_ = 100;
            maxPoolSize_ = 190;
            initialPoolSize_ = 150;  // this should get reset to maxPoolSize_
            numDaemons_ = 75;
            if (timeToRunDaemons_ == 0) {
                timeToRunDaemons_ = 180*1000; // 180 seconds == 3 minutes
            }
        }
        else
        {  // Set more conservative values, as we'll be making actual connections to an
           // actual server, and we don't want to monopolize the server.
            minPoolSize_ = 5;
            maxPoolSize_ = 15;
            initialPoolSize_ = 9;
            numDaemons_ = 4;
            if (timeToRunDaemons_ == 0) {
                timeToRunDaemons_ = 15*1000; // 15 seconds
            }
        }
        maxLifetime_ = (int)timeToRunDaemons_ / 3;
        maxIdleTime_ = (int)timeToRunDaemons_ / 4;
        propertyCycle_ = timeToRunDaemons_ / 4;
        poolHealthCheckCycle_ = Math.min(timeToRunDaemons_ / 4, 20*60*1000);
```

```
            // at least once every 20 minutes (more frequently if shorter run-time)
            daemonMaxSleepTime_ = Math.min(timeToRunDaemons_ / 3, 10*1000);
            // at most 10 seconds (less if shorter run-time)
            daemonMinSleepTime_ = 20;  // milliseconds

            if (DEBUG)
               System.out.println("setup: Constructing "
                               + "AS400JDBCManagedConnectionPoolDataSource (cpds0)");
            AS400JDBCManagedConnectionPoolDataSource cpds0 = new AS400JDBCManagedConnectionPoolDataSource();
// Set datasource properties.  Note that both CPDS and MDS have these
            // properties, and they might have different values.
            cpds0.setServerName(host);
            cpds0.setDatabaseName(host);//iasp can be here
            cpds0.setUser(userid);
            cpds0.setPassword(password);

            cpds0.setSavePasswordWhenSerialized(true);

            // Set connection pooling-specific properties.
            cpds0.setInitialPoolSize(initialPoolSize_);
            cpds0.setMinPoolSize(minPoolSize_);
            cpds0.setMaxPoolSize(maxPoolSize_);
            cpds0.setMaxLifetime((int)(maxLifetime_/1000));  // convert to seconds
            cpds0.setMaxIdleTime((int)(maxIdleTime_/1000));  // convert to seconds
            cpds0.setPropertyCycle((int)(propertyCycle_/1000));  // convert to seconds
            //cpds0.setReuseConnections(false);  // don't re-use connections

            // Set the initial context factory to use.
            System.setProperty(Context.INITIAL_CONTEXT_FACTORY,
                               "com.sun.jndi.fscontext.RefFSContextFactory");

            // Get the JNDI Initial Context.
            Context ctx = new InitialContext();

            // Note: The following is an alternative way to set context properties locally:
            //   Properties env = new Properties();
            //   env.put(Context.INITIAL_CONTEXT_FACTORY,
            //           "com.sun.jndi.fscontext.RefFSContextFactory");
            //   Context ctx = new InitialContext(env);

            ctx.rebind("mydatasource", cpds0);
                      // We can now do lookups on cpds, by the name"mydatasource".

            if (DEBUG)
              System.out.println("setup: lookup(\"mydatasource\"" + ")");
           //   AS400JDBCManagedConnectionPoolDataSource cpds1 =
           //   (AS400JDBCManagedConnectionPoolDataSource)ctx.lookup("mydatasource");
           // if (DEBUG) System.out.println("setup: cpds1.getUser() == |" + cpds1.getUser() + "|");

            // Create a standard DataSource object that references it.

            if (DEBUG)
               System.out.println("setup: Constructing AS400JDBCManagedDataSource (mds0)");
            AS400JDBCManagedDataSource mds0 = new AS400JDBCManagedDataSource();
            mds0.setDescription("DataSource supporting connection pooling");
            mds0.setDataSourceName("mydatasource");
            ctx.rebind("ConnectionPoolingDataSource", mds0);

            if (DEBUG)
              System.out.println("setup: lookup(\"ConnectionPoolingDataSource\"" + ")");
            dataSource_ = (DataSource)ctx.lookup("ConnectionPoolingDataSource");
            //dataSource_.setLogWriter(output_);
            if (DEBUG)
              System.out.println("setup: dataSource_.getUser() == |" +
                    ((AS400JDBCManagedDataSource)dataSource_).getUser() + "|");

            mds_ = (AS400JDBCManagedDataSource)dataSource_;
        }
        catch (Exception e)
        {
            e.printStackTrace();
            System.out.println("Setup error during Trace file creation.");
        }
    }

    void displayConnectionType(Connection conn, boolean specifiedDefaultId)
    {
        if (conn instanceof com.ibm.as400.access.AS400JDBCConnectionHandle)
            System.out.print("("+ (specifiedDefaultId ? "+" : "-") + "P)");
```

```
    else
        System.out.print("("+ (specifiedDefaultId ? "+" : "-") + "NP)");
}


/**
 *  Gets and returns connections from and to a connection pool for a while.
 **/
public void runTest()
{
    boolean ok = true;
    try
    {
        System.out.println("Started test run at " + new Date());

        if (DEBUG)
          System.out.println("Checking health just after datasource creation "
                     + "(we expect that the pool does not exist yet) ...");
        if (mds_.checkPoolHealth(true)) {
            ok = false;
            System.out.println("\nERROR: Pool exists prior to first getConnection().");
        }

        // Verify some setters/getters for JDBC properties.
        System.out.println("Verifying setters/getters ...");

        mds_.setAccess("read only");
        if (!mds_.getAccess().equals("read only")) {
            ok = false;
            System.out.println("\nERROR: getAccess() returned unexpected value: "
                    + "|" + mds_.getAccess()+"|");
        }

        boolean oldBool = mds_.isBigDecimal();
        boolean newBool = (oldBool ? false : true);
        mds_.setBigDecimal(newBool);
        if (mds_.isBigDecimal() != newBool) {
            ok = false;
            System.out.println("\nERROR: isBigDecimal() returned unexpected value: "
                               + "|"+mds_.isBigDecimal()+"|");
        }
        mds_.setBigDecimal(oldBool);

        int oldInt = mds_.getBlockCriteria();
        int newInt = (oldInt == 2 ? 1 : 2);
        mds_.setBlockCriteria(newInt);
        if (mds_.getBlockCriteria() != newInt) {
            ok = false;
            System.out.println("\nERROR: getBlockCriteria() returned unexpected value: "
                              + "|"+mds_.getBlockCriteria()+"|");
        }
        mds_.setBlockCriteria(oldInt);

        // Verify some setters and getters for socket properties.

        oldBool = mds_.isKeepAlive();
        newBool = (oldBool ? false : true);
        mds_.setKeepAlive(newBool);
        if (mds_.isKeepAlive() != newBool) {
            ok = false;
            System.out.println("\nERROR: isKeepAlive() returned unexpected value: "
                              + "|"+mds_.isKeepAlive()+"|");
        }
        mds_.setKeepAlive(oldBool);

        oldInt = mds_.getReceiveBufferSize();
        newInt = (oldInt == 256 ? 512 : 256);
        mds_.setReceiveBufferSize(newInt);
        if (mds_.getReceiveBufferSize() != newInt) {
            ok = false;
            System.out.println("\nERROR: getReceiveBufferSize() returned unexpected value: "
                              + "|"+mds_.getReceiveBufferSize()+"|");
        }
        mds_.setReceiveBufferSize(oldInt);


        System.out.println("CONNECTION 1");
        Object o = dataSource_.getConnection();
        System.out.println(o.getClass());
```

```
System.out.println("******LOOK ABOVE*******");
Connection c1 = dataSource_.getConnection();

if (DEBUG)
  displayConnectionType(c1, true);

if (DEBUG)
  System.out.println("Checking health after first getConnection() ...");
if (!mds_.checkPoolHealth(true)) {
    ok = false;
    System.out.println("\nERROR: Pool is not healthy after first getConnection().");
}

if (!TESTING_THREAD_SAFETY)
{
    try
    {
        c1.setAutoCommit(false);
        if (DEBUG)
          System.out.println("SELECT * FROM QIWS.QCUSTCDT");
        Statement s = c1.createStatement();
        ResultSet rs = s.executeQuery("SELECT * FROM QIWS.QCUSTCDT");
        while (rs.next ()) {
            if (DEBUG)
              System.out.println(rs.getString(2));
        }
        rs.close();
        s.close();
    }
    catch (Exception e) {
        e.printStackTrace();
        if (DEBUG)
          System.out.println("Checking health after fatal connection error ...");
        if (!mds_.checkPoolHealth(true)) {
          ok = false;
          System.out.println("\nERROR: Pool is not healthy after fatal connection "
                             + "error.");
        }
    }
}

System.out.println("CONNECTION 2");
Connection c2 = dataSource_.getConnection(userid, password);
if (DEBUG)
  displayConnectionType(c2, false);
System.out.println("CONNECTION 3");
Connection c3 = dataSource_.getConnection();
if (DEBUG)
  displayConnectionType(c3, true);
c1.close();

if (DEBUG)
  System.out.println("Checking health after first close() ...");
if (!mds_.checkPoolHealth(true)) {
    ok = false;
    System.out.println("\nERROR: Pool is not healthy after first close().");
}

System.out.println("CONNECTION 4");
Connection c4 = dataSource_.getConnection();
if (DEBUG) displayConnectionType(c4, true);

c1.close();  // close this one again
c2.close();
c3.close();
c4.close();

if (DEBUG) System.out.println("Checking health after last close() ...");
if (!mds_.checkPoolHealth(true)) {
    ok = false;
    System.out.println("\nERROR: Pool is not healthy after last close().");
}

// Start the test daemons.
System.out.println("Starting test daemons");
startThreads();

// Run the test daemons for a while; check pool health periodically.
```

```
        long startTime = System.currentTimeMillis();
        long endTime = startTime + timeToRunDaemons_;
        while (System.currentTimeMillis() < endTime)
        {
            System.out.print("h");
            // Let the daemons run for a while, then check pool health.
            try {
                Thread.sleep(poolHealthCheckCycle_);
            }
            catch (InterruptedException ie) {}
            if (!mds_.checkPoolHealth(true)) {
                ok = false;
                System.out.println("\nERROR: Pool is not healthy after test daemons started.");
            }
        }

        // Stop the test daemons.
        System.out.println("\nStopping test daemons");
        stopThreads();

        if (DEBUG)
          System.out.println("Checking health after connectionGetter daemons have run...");
        if (!mds_.checkPoolHealth(true)) {
            ok = false;
            System.out.println("\nERROR: Pool is not healthy after test daemons stopped.");
        }

        if (!TESTING_THREAD_SAFETY)
        {
            System.out.println("CONNECTION 5");
            Connection c = dataSource_.getConnection();
            if (DEBUG) displayConnectionType(c, true);
            c.setAutoCommit(false);
            if (DEBUG) System.out.println("SELECT * FROM QIWS.QCUSTCDT");
            Statement s = c.createStatement();
            ResultSet rs = s.executeQuery("SELECT * FROM QIWS.QCUSTCDT");
            while (rs.next ()) {
                if (DEBUG) System.out.println(rs.getString(2));
            }
            rs.close();
            s.close();
            c.close();            }

        System.out.println("\nClosing the pool...");
        mds_.closePool();

        if (DEBUG)
          System.out.println("Checking health after pool closed ...");
        Trace.setTraceJDBCOn(true);  // make sure the final stats get printed out
        Trace.setTraceOn(true);
        if (!mds_.checkPoolHealth(true)) {
            ok = false;
            System.out.println("\nERROR: Pool is not healthy after pool closed.");
        }

        System.out.println();
        if(ok==true)
            System.out.println("test ran ok");
        else
            System.out.println("test failed");
    }
    catch (Exception e)
    {
        System.out.println(e);
        e.printStackTrace();
    }
    finally {
        System.out.println("Ended test at " + new Date());
    }
}

void startThreads()
{
    // Create a bunch of threads that call getConnection().
    Thread[] threads = new Thread[numDaemons_];
    for (int i=0; i<numDaemons_; i++)
    {
        ConnectionGetter getter;
        // Flip a coin to see if this daemon will specify the default uid, or unique uid.
```

```
        if (random_.nextBoolean())
        {
            getter = new ConnectionGetter(userid,password);
            if (TESTING_THREAD_SAFETY) {  // we can use fictional userid
                getter = new ConnectionGetter("Thread"+i, "Pwd"+i);
            }
            else {  // must use a real userid
                getter = new ConnectionGetter(userid,password);
            }
        }
        else
            getter = new ConnectionGetter(null, null);

        threads[i] = new Thread(getter, "["+i+"]");
        threads[i].setDaemon(true);
    }

    // Start the threads.
    for (int i=0; i<numDaemons_; i++)
    {
        threads[i].start();
    }
}

void stopThreads()
{
    // Tell the threads to stop.
    keepDaemonsAlive_ = false;
    synchronized (daemonSleepLock_) {
        daemonSleepLock_.notifyAll();
    }

    // Wait for the daemons to stop.
    try {
        Thread.sleep(3000);
    }
    catch (InterruptedException ie) {}
}


// ConnectionGetter ---------------------------------------------------------------

/**
     Helper class. This daemon wakes up at random intervals and either
     gets another connection from the connection pool or returns a
     previously-gotten connection to the pool.
 **/
private final class ConnectionGetter implements Runnable
{
    private String uid_;
    private String pwd_;
    private boolean useDefaultUid_;
    private long maxSleepTime_;
    private String threadName_;
    private boolean firstConnection_ = true;
    ArrayList connections_ = new ArrayList();
    // list of connections that this getter currently 'owns'.

    ConnectionGetter(String uid, String pwd) {
        uid_ = uid;
        pwd_ = pwd;
        if (uid_ == null) useDefaultUid_ = true;
        else useDefaultUid_ = false;
        maxSleepTime_ = daemonMaxSleepTime_;  // our own copy that we can adjust
    }

    public void run()
    {
        threadName_ = Thread.currentThread().getName();
        if (DEBUG) System.out.println("ConnectionGetter("+threadName_+") Starting up");

        try
        {
            while (keepDaemonsAlive_)
            {
                try
                {
                    // Pick a random sleep-time, between min and max values.
                    long sleepTime = Math.max((long)(maxSleepTime_ * random_.nextFloat()),
```

```
                daemonMinSleepTime_);
            // Note: Must never call wait(0), because that waits forever.
            synchronized (daemonSleepLock_) {
                try {
                    daemonSleepLock_.wait(sleepTime);
                    System.out.print(".");
                }
                catch (InterruptedException ie) {}
            }
            if (!keepDaemonsAlive_) break;

            // Decide by chance whether to request another connection or return a
            // previously-obtained connection.
            Connection conn;
            if (random_.nextBoolean())  // Leave the decision to chance.
            { // Request another connection.
                if (useDefaultUid_)
                {
                    if (DEBUG)
                      System.out.println("ConnectionGetter("+threadName_+") - get()");
                    conn = dataSource_.getConnection();
                }
                else
                {
                    if (DEBUG)
                       System.out.println("ConnectionGetter("+threadName_+") - "
                                          + "get("+uid_+",***)");
                    conn = dataSource_.getConnection(uid_, pwd_);
                }

                if (conn == null) {
                    System.out.println("ConnectionGetter("+threadName_+") ERROR: "
                                       + "getConnection() returned null");
                }
                else
                {
                    // Occasionally "forget" that we own a connection, and neglect to
                    // close it.
                    // Orphaned connections should eventually exceed their maximum
                    // lifetime and get "reaped" by the connection manager.
                    float val = random_.nextFloat();
                    if (firstConnection_ || val < 0.1) {
                       // 'orphan' a few gotten connections
                       firstConnection_ = false;
                    }
                    else {
                        connections_.add(conn);
                    }
                    if (DEBUG) displayConnectionType(conn, useDefaultUid_);
                    if (conn instanceof com.ibm.as400.access.AS400JDBCConnectionHandle)
                    {  // We got a pooled connection.  Try speeding up our cycle time.
                        if (maxSleepTime_ > 100)
                          maxSleepTime_--;
                        else
                          maxSleepTime_ = 100;
                    }
                    else
                    {  // We didn't get a pooled connection.  That means that the pool
                       // must be at capacity.  Slow down our cycle time a bit.
                        maxSleepTime_ = maxSleepTime_ + 50;
                    }
                }
            }
            else { // Close a connection that we currently own.
                if (connections_.size() != 0) {
                    conn = (Connection)connections_.remove(0);
                    conn.close();
                }
            }
        }  // inner try
        catch (Exception e)
        {
            e.printStackTrace();
        }
    } // outer while
}  // outer try
finally
{
    if (DEBUG)
```

```
                System.out.println("ConnectionGetter("+threadName_+") Stopping");
              // Return all the connections that I still have in my list.
              for (int i=0; i<connections_.size(); i++) {
                  Connection conn = (Connection)connections_.remove(0);
                  try { conn.close(); } catch (Exception e) { e.printStackTrace(); }
              }

          }
       }

   }  // internal class 'ConnectionGetter'
}
```

例: JDBCQuery を使用してテーブルを照会する

このプログラムは、IBM Toolbox for Java JDBC ドライバーを使用して、テーブルを照会し、その内容を出力します。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////////////////////////
//
// 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

このプログラムは、IBM Toolbox for Java のジョブ・サポートの例です。ここでは、システム上の特定のユーザーに関するジョブ ID 情報がリストされます。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////////////////
//
// 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

この例は、IBM Toolbox for Java Job クラスを使用して、サーバー上のジョブのリストを取得し、ジョブ状況とその後に続くジョブ ID を出力します。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////////////////
//
// 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

このプログラムは、IBM Toolbox for Java のジョブ・ログ機能の例です。これは、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します。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////////////////
//
// 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

以下の例には、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。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
///////////////////////////////////////////////////////////////////////
//
// 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 server, 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, 0, bytesRead);
            }
        } 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、 コードに関するライセンス情報および特記事項をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// 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

以下の例は、PrintObjectInputStream クラスを使用して、既存のスプール・ファイルを読み取る方法を示しています。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
///////////////////////////////////////////////////////////////////////
//
// 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・ファイルを読み取り変換する

以下の例は、スプール・ファイル・データを読み取る際に、異なる変換を取得できるように PrintParameterList をセットアップする方法を示しています。以下のコード・セグメントでは、スプール・ファイルがすでにサーバーに存在しており、 *createSpooledFile()* メソッドが、スプール・ファイルを表す SpooledFile クラスのインスタンスを作成することを想定しています。

PrintObjectPageInputStream の例

注: 法律上の重要な情報に関しては、コードに関するライセンス情報および特記事項をお読みください。

以下の例は、GIF イメージとしてフォーマットされるデータのページを読み取るために、PrintObjectPageInputStream オブジェクトを作成する方法を示しています。この場合、スプール・ファイルからの各ページが GIF イメージに変換されます。データ変換を指定するために GIF ワークステーション・カスタマイズ・オブジェクトが使用されます。

```
// Create a spooled file
SpooledFile splF = createSpooledFile();

// Set up print parameter list
PrintParameterList printParms = new PrintParameterList();
printParms.setParameter(PrintObject.ATTR_WORKSTATION_CUST_OBJECT, "/QSYS.LIB/QWPGIF.WSCST");
printParms.setParameter(PrintObject.ATTR_MFGTYPE, "*WSCST");

// Create a page input stream from the spooled file
PrintObjectPageInputStream is = splF.getPageInputStream(printParms);
```

PrintObjectTransformedInputStream の例

注: 法律上の重要な情報に関しては、コードに関するライセンス情報および特記事項をお読みください。

以下の例は、TIFF としてフォーマットされるデータを読み取るために、PrintObjectTransformedInputStream オブジェクトを作成する方法を示しています。データ変換を指定するために TIFF (G4 圧縮) ワークステーション・カスタマイズ・オブジェクトが使用されます。

```
// Create a spooled file
SpooledFile splF = createSpooledFile();

// Set up print parameter list
PrintParameterList printParms = new PrintParameterList();
printParms.setParameter(PrintObject.ATTR_WORKSTATION_CUST_OBJECT, "/QSYS.LIB/QWPTIFFG4.WSCST");
printParms.setParameter(PrintObject.ATTR_MFGTYPE, "*WSCST");

// Create a transformed input stream from the spooled file
PrintObjectTransformedInputStream is = splF.getTransformedInputStream(printParms);
```

メーカーの機種および型式を使用した PrintObjectTransformedInputStream の例

注: 法律上の重要な情報に関しては、コードに関するライセンス情報および特記事項をお読みください。

以下の例は、ASCII プリンターへの出力用にフォーマットされたデータを読み取るために、PrintObjectTransformedInputStream オブジェクトを作成する方法を示しています。データ変換を指定するために *HP4 のメーカーの機種および型式が使用されます。

```
// Create a spooled file
SpooledFile splF = createSpooledFile();

// Set up print parameter list
PrintParameterList printParms = new PrintParameterList();
printParms.setParameter(PrintObject.ATTR_MFGTYPE, "*HP4");

// Create a transformed input stream from the spooled file
PrintObjectTransformedInputStream is = splF.getTransformedInputStream(printParms);
```

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)

以下の例は、リストの構築時にフィードバックを取得するために、PrintObjectListListener インターフェースを使用して、サーバー上のスプール・ファイルをすべて非同期でリストする方法を示します。非同期でのリスト表示を用いれば、リスト全体が作成される前にリスト・オブジェクトの処理を呼び出し元が開始できるので、ユーザーが感じる応答時間が高速化されます。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// Example that shows listing all spooled files on a server 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 server 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)

以下の例は、PrintObjectListListener インターフェースを使用しないで、システム上ですべてのスプール・ファイルを非同期でリストする方法を示します。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// This example lists 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

以下の例は、サーバー上のすべてのスプール・ファイルを同期してリストする方法を示します。同期リスト表示を行うと、完全なリストが作成されるまで、操作は呼び出し元には戻りません。ユーザーは、非同期のリスト表示と比較して、応答時間が遅いと感じます。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// Example that shows listing all spooled files on a server 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

このプログラムは、QWCRSSTS サーバー・プログラムを呼び出して、システムの状況を検索します。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// 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

この IBM Toolbox for Java プログラム例は、サーバーの名前と表示するファイルをユーザーにプロンプトでたずねます。そのファイルは、存在していなければならず、レコードが入っていなければなりません。ファイル内の各レコードは、System.out に表示されます。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
///////////////////////////////////////////////////////////////////////////////
//
// 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 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

このプログラムは、レコード・レベルの access クラスを使用して、サーバー上のファイルからレコードを読み取ります。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// 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, 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 must 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

このプログラムは、レコード・レベルの access クラスを使用して、サーバー上のファイルからレコードをキー別に読み取ります。ファイル QCUSTCDTKY を実行するサーバー名と、このファイルを作成するライブラリーが、ユーザーにプロンプトでたずねられます。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// 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 IBM i 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

このソースは、IBM Toolbox for Java UserList の例です。このプログラムは、特定のグループ内のすべてのユーザーをリストします。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
///////////////////////////////////////////////////////////////////////
//
// 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

この項では、IBM Toolbox for Java bean に関する情報全体で取り上げられているコード例をリストします。

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。

コードに関するライセンス情報および特記事項

IBM は、お客様に、すべてのプログラム・コードのサンプルを使用することができる非独占的な著作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。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

こ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、いかなる保証も適用されません。商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任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、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////////////////////////
//
// 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()
```

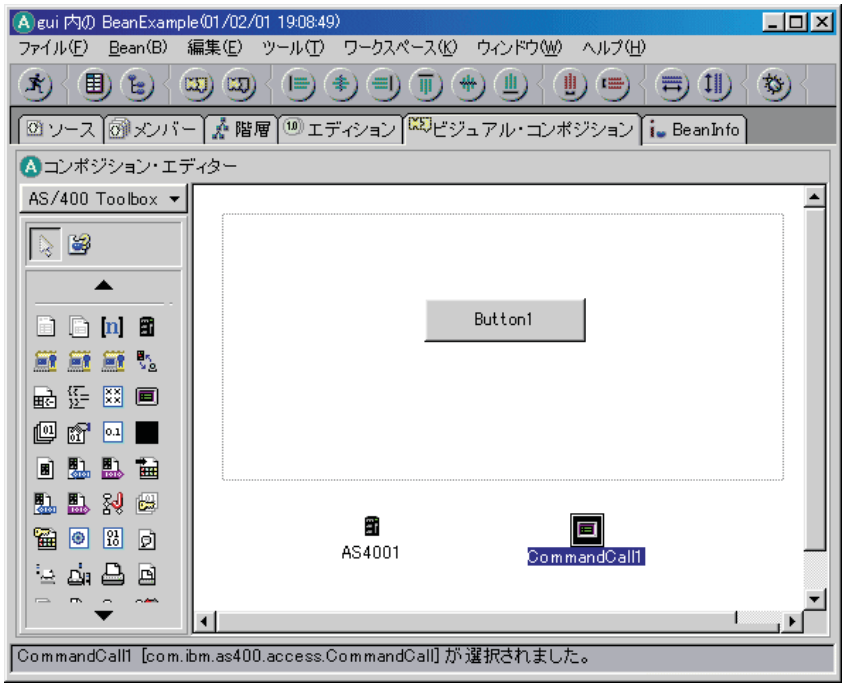
```
          {
             public void connected(ConnectionEvent event)
             {
                 System.out.println( "System connected." );
             }
             public void disconnected(ConnectionEvent event)
             {
                 System.out.println( "System disconnected." );
             }
          }
        );

        // 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);
    }
}
```

例: visual bean ビルダーを使用して bean を作成する

この例では IBM VisualAge for Java エンタープライズ版 V2.0 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。

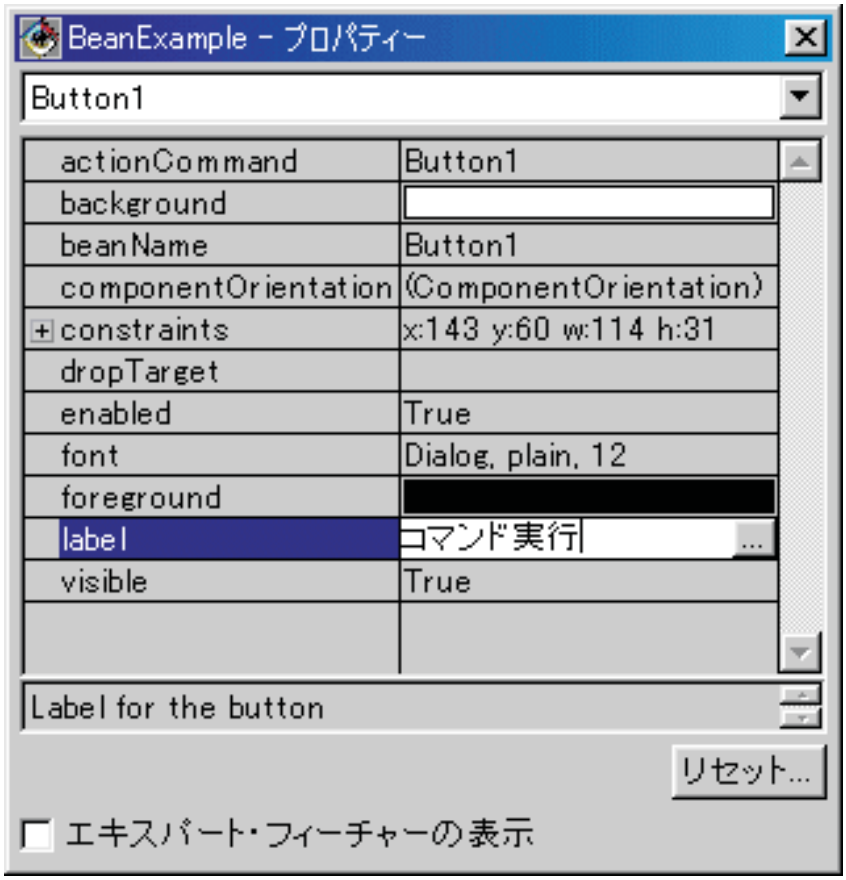
- ボタンをそのアプレットにドラッグ・アンド・ドロップします。 (このボタンは、図 1 に示されている bean ビルダーの「ビジュアル・コンポジション」タブの左にあります。)
- アプレットの外に CommandCall bean と AS400 bean をドロップします。 (この Bean は、図 1 に示されている bean ビルダーの「ビジュアル・コンポジション」タブの左にあります。)

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



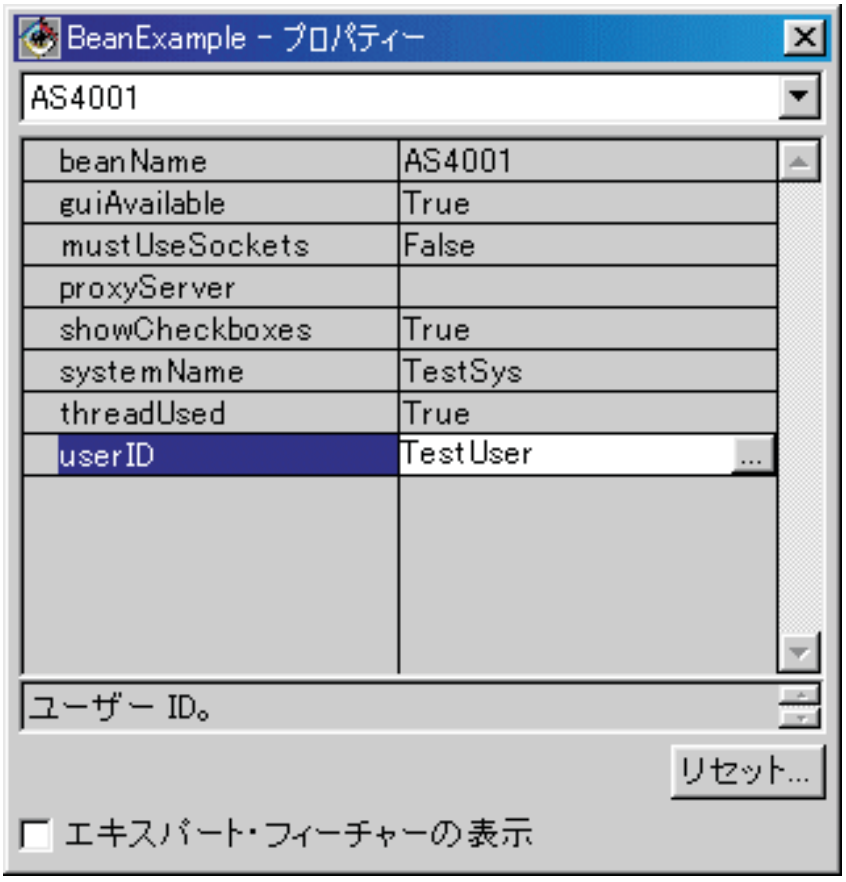
- bean のプロパティーを編集する。 (編集する際には、bean を選択してから右クリックし、「プロパティー」オプションのあるウィンドウを表示します。)
  - ボタンのラベルを図 2 に示されているように「**コマンド実行**」に変更します。

図 2: ボタン・ラベルの「コマンド実行」への変更



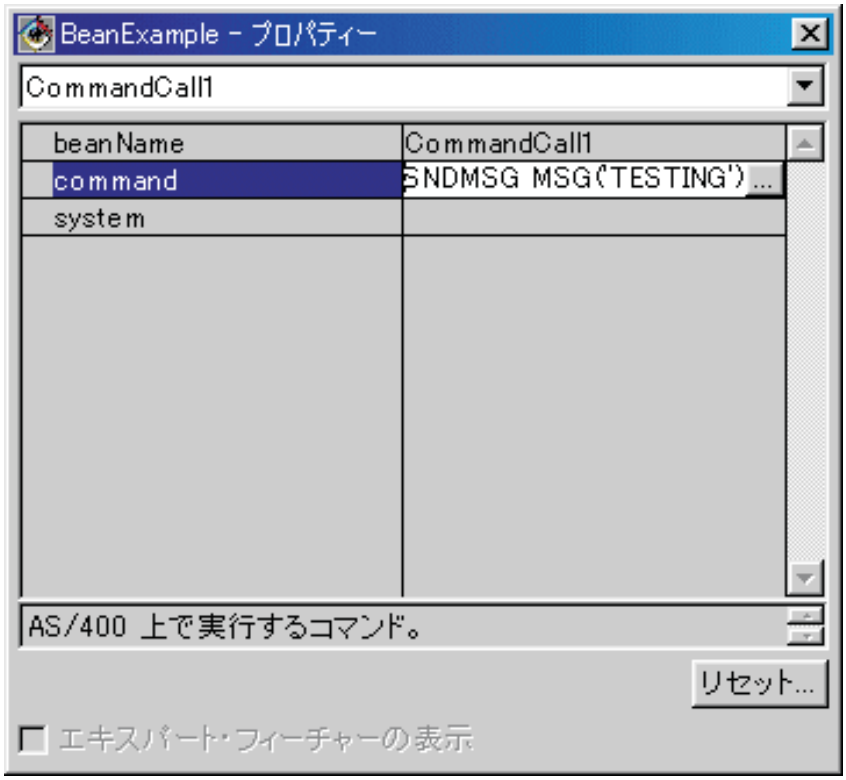
– AS400 bean のシステム名を「**TestSys**」に変更します。
– AS400 bean のユーザー ID を図 3 に示されているように「**TestUser**」に変更します。

図 3: ユーザー ID の名前の「TestUser」への変更



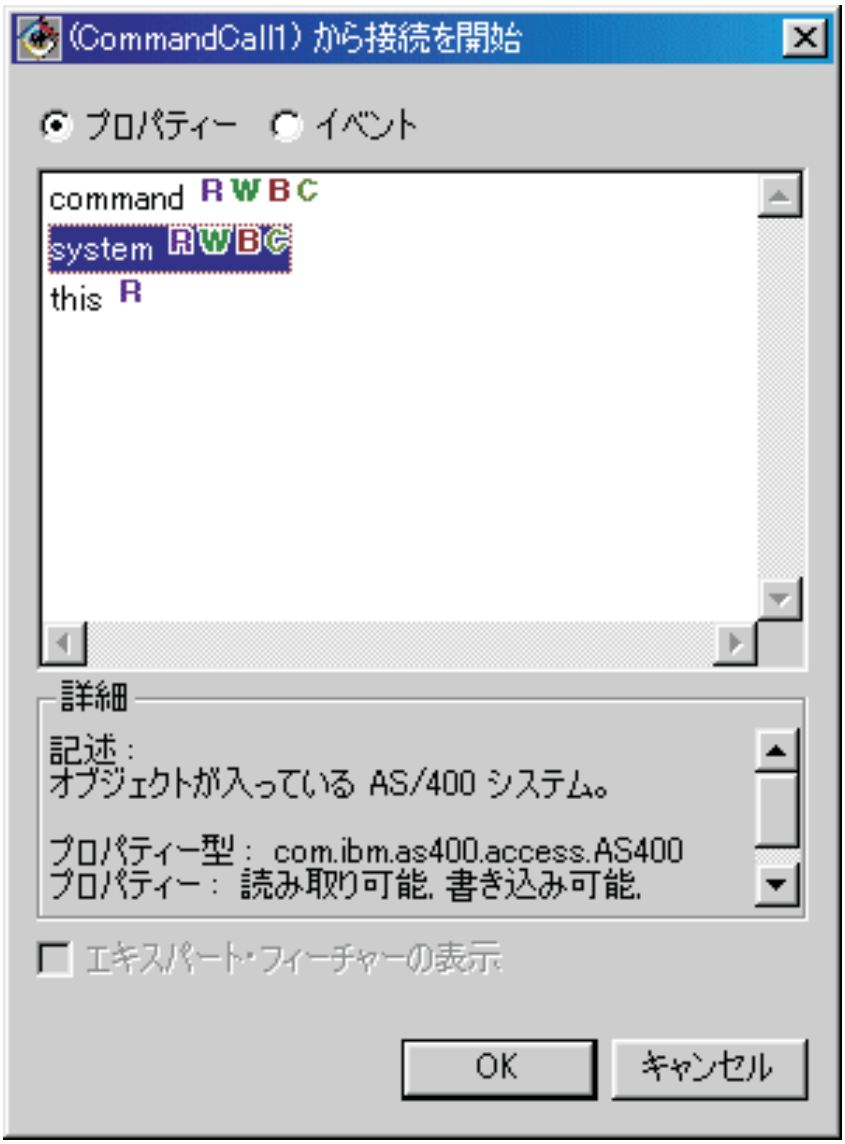
– CommandCall bean のコマンドを図 4 に示されているように、 **SNDMSG MSG('Testing') TOUSR('TESTUSER')** に変更します。

図 4: CommandCall bean のコマンドの変更



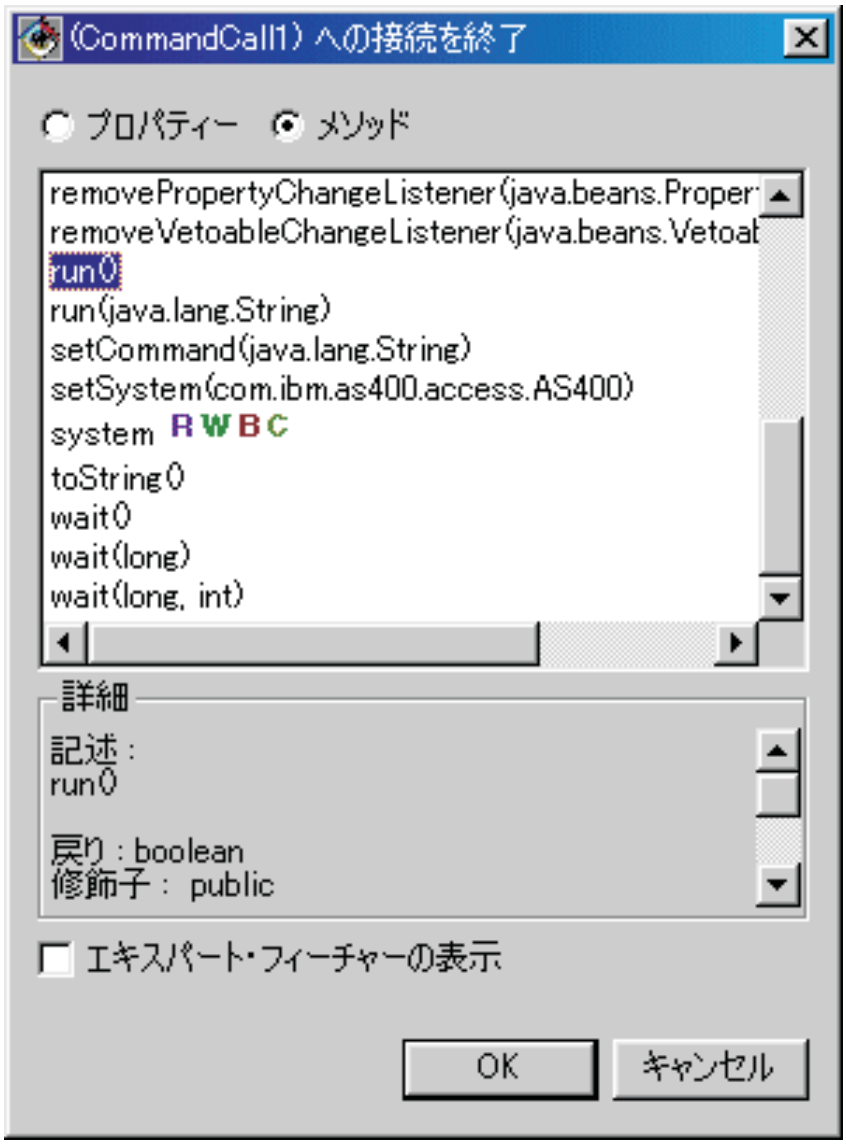
- AS400 bean を CommandCall bean に接続します。このときに使用するメソッドは、各 bean ビルダー間で異なります。この例では、以下を行います。
  - CommandCall bean を選択してから右マウス・ボタンをクリックする。
  - 「**接続**」を選択する。
  - 「**接続可能なフィーチャー**」を選択する。
  - 図 5 に示されているように、機能のリストから「**system**」を選択する。
  - AS400 bean を選択する。
  - その AS400 bean 上に表示されるポップアップ・メニューから「**this**」を選択する。

図 5: CommandCall bean への AS400 bean の接続



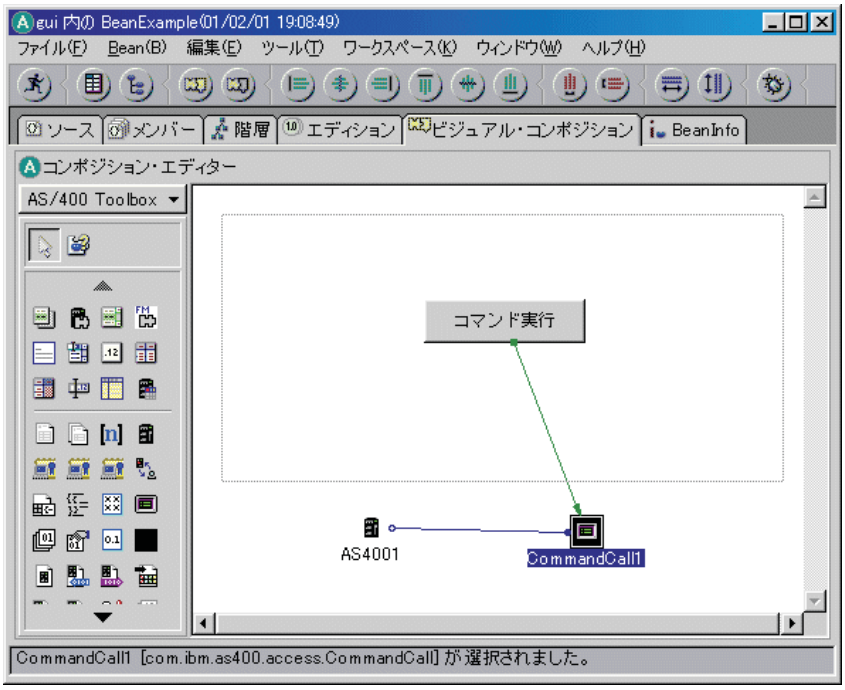
- ボタンを CommandCall bean に接続します。
  - Button bean を選択してから右マウス・ボタンをクリックする。
  - 「**接続**」を選択する。
  - 「**actionPerformed**」を選択する。
  - CommandCall bean を選択する。
  - 表示されるポップアップ・メニューから「**接続可能なフィーチャー**」を選択する。
  - 図 6 に示されているように、メソッドのリストで「**run**」を選択します。

図 6: ボタンへのメソッドの接続



上記の終了後、「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 - 完了した bean の例



例: Commtrace クラス

このトピックは、IBM Toolbox for Java commtrace クラスの資料で提供されているサンプル・コードにリンクしています。

- 203 ページの『例: Commtrace 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。

コードに関するライセンス情報および特記事項

IBM は、お客様に、すべてのプログラム・コードのサンプルを使用することができる非独占的な著作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。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

こ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、いかなる保証も適用されません。商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任の保証の適用も一切ありません。

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これらの例を使用してください。

- パネルの構成と表示: Graphical Toolbox 環境の基本機能および操作を全体的に示す、簡単なパネルの構成。

- 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。
- 完全に機能するダイアログの構成: 完全に機能するダイアログの構成 (データをパネルに供給する DataBeans を実装し、PDML 内の属性を識別した後)
- 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が動的に変更する方法。
- 編集可能コンボ・ボックス: 編集可能コンボ・ボックスのデータ bean のコーディング。

以下の例は、さまざまな GUI エレメントを作成するために 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。
- デック・ペイン: デック・ペインの作成、および完成したデック・ペインがどのようなものか。
- プロパティー・シート: プロパティー・シートの作成、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。
- 分割ペイン: 分割ペインの作成、および完成した分割ペインがどのようなものか。
- タブ形式ペイン: タブ形式ペインの作成、および完成したタブ形式ペインがどのようなものか。
- 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。
- ツールバー: ツールバーの作成、および完成したツールバーがどのようなものか。
- メニュー・バー: メニュー・バーの作成、および完成したメニュー・バーがどのようなものか。
- ヘルプ: ヘルプ文書の生成およびヘルプ文書のトピック・ページへの分割。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。

コードに関するライセンス情報および特記事項

IBM は、お客様に、すべてのプログラム・コードのサンプルを使用することができる非独占的な著作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。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

こ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、いかなる保証も適用されません。商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任の保証の適用も一切ありません。

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。

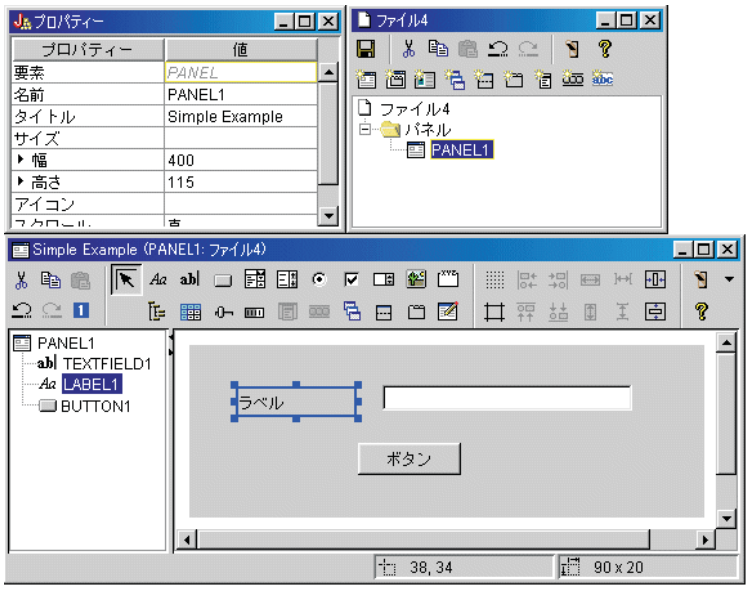
この例では、パネルの作成方法を示した後、そのパネルを表示する小さな Java アプリケーションの作成方法を示します。この例では、ユーザーがデータをテキスト・フィールドに入力して、「**閉じる**」ボタンをクリックします。アプリケーションはその後、そのデータを Java コンソールにエコー表示します。

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。

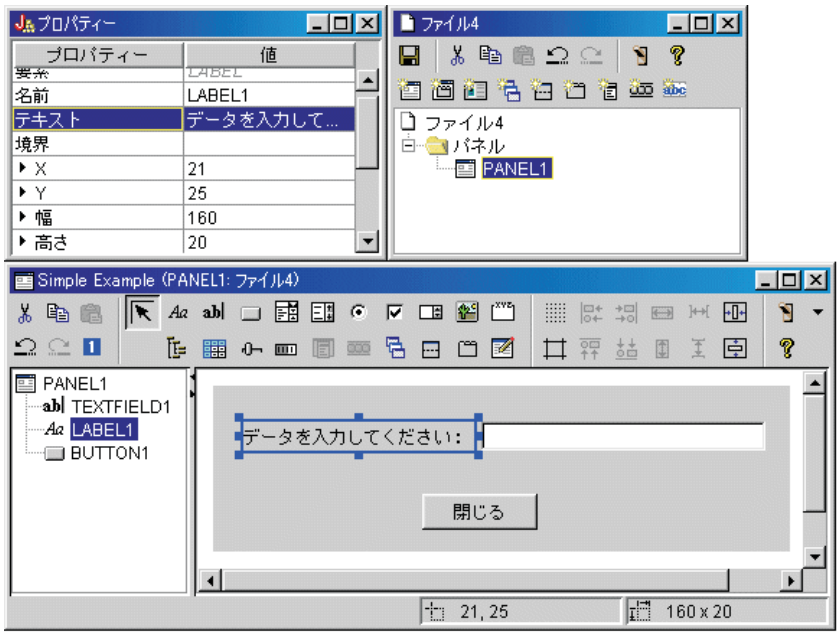
注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

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。

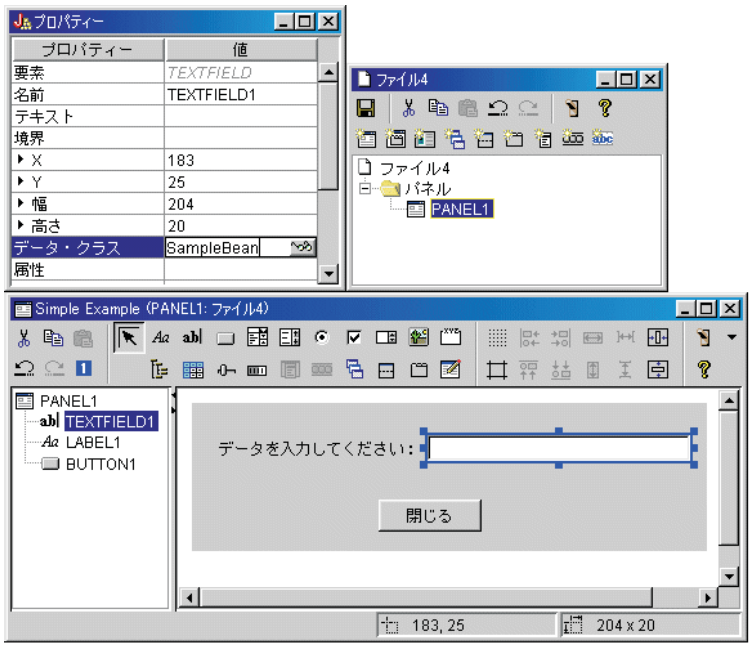
図 2: GUI ビルダーのウィンドウ: 「プロパティー」ウィンドウでテキストを変更する



テキスト・フィールド

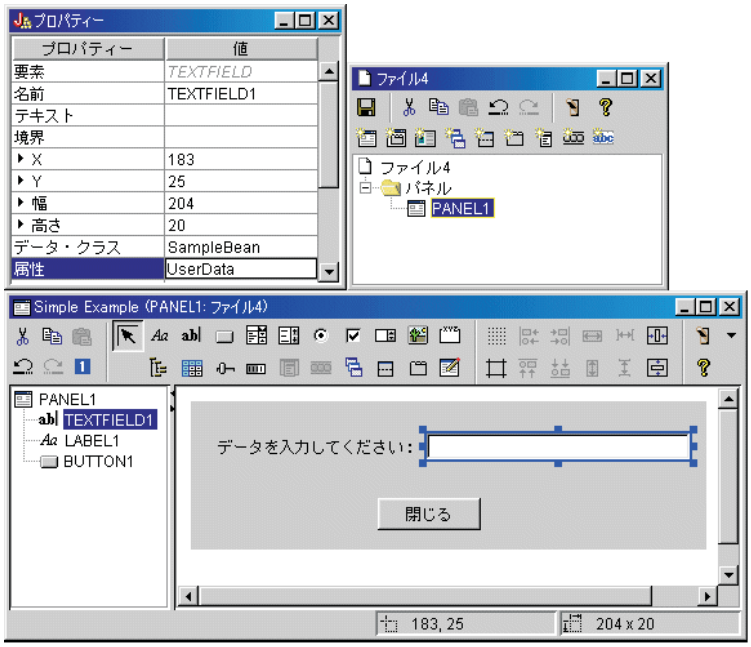
テキスト・フィールドにはデータが含まれるので、 GUI ビルダーが何種類かの追加の作業を実行するためのプロパティーを設定できます。この例では、データ・クラス・プロパティーを bean クラスの名前である **SampleBean** に設定します。この databean は、このテキスト・フィールドにデータを供給します。

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。

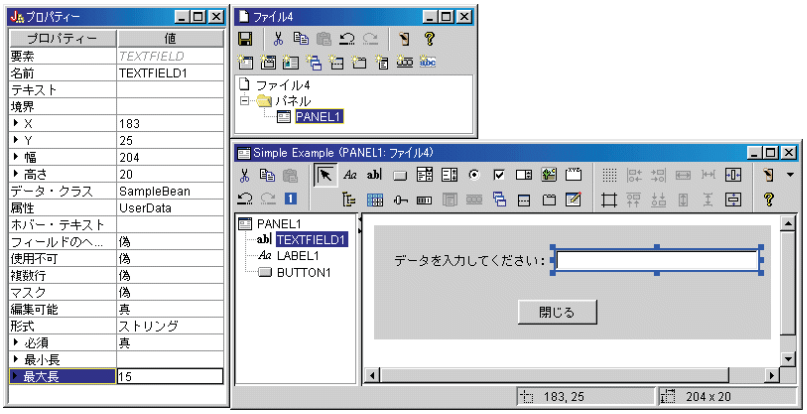
図 4: GUI ビルダーのウィンドウ: 属性プロパティーを設定する



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。

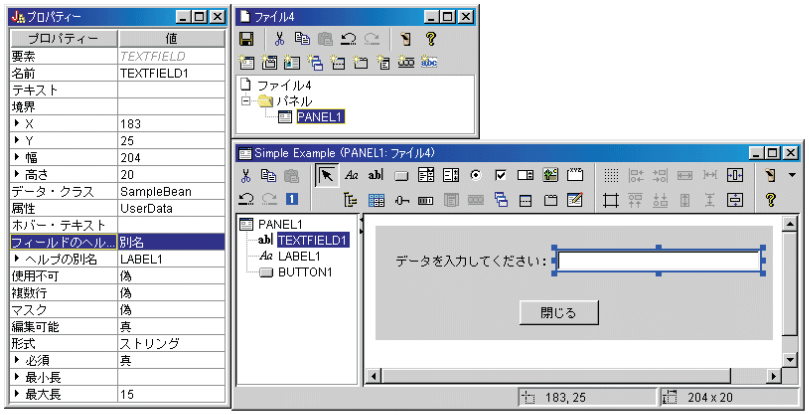
ユーザーによるデータ入力が必要であることや、そのデータは最大長 15 文字のストリングでなければならないことを指定しています。

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。

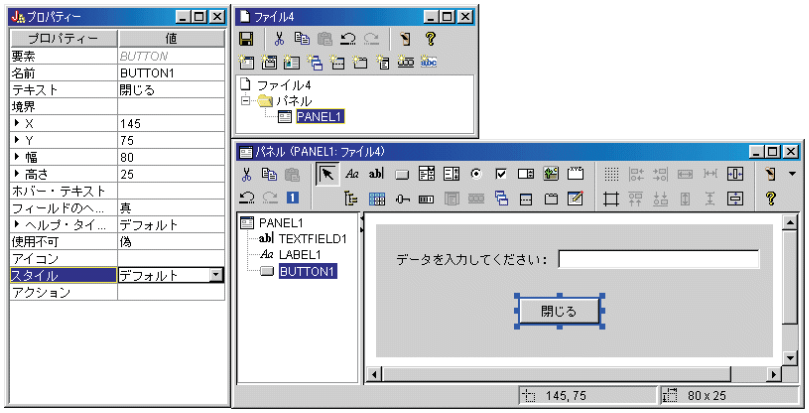
図 6: GUI ビルダーのウィンドウ: テキスト・フィールドにコンテキストに依存したヘルプを設定する



ボタン

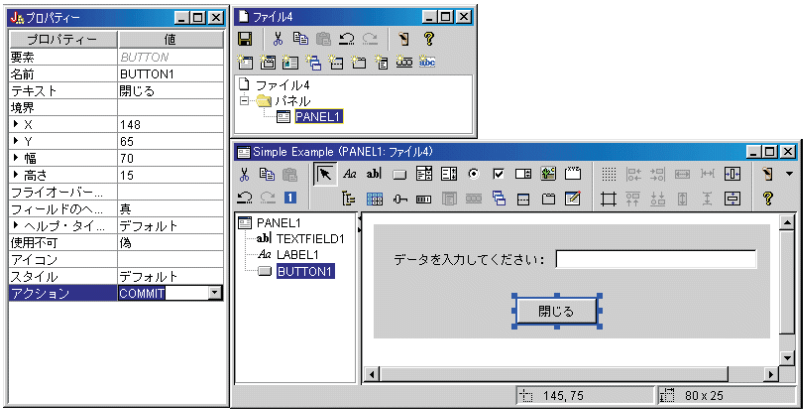
スタイル・プロパティーを変更して、ボタンのデフォルト強調を設定します。

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



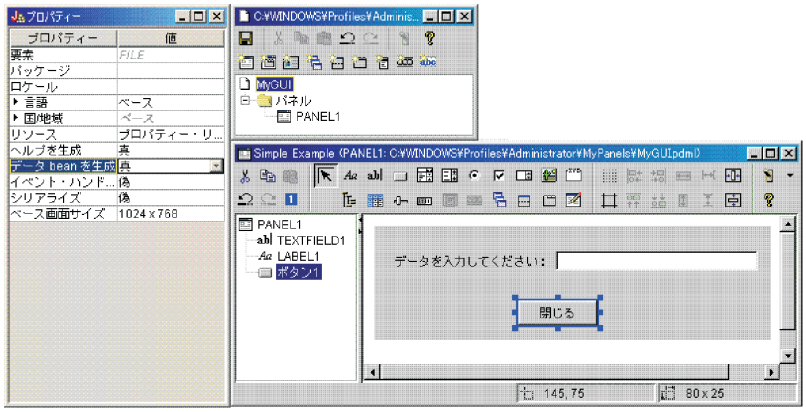
アクション・プロパティーを COMMIT に設定します。この設定により、ボタンを選択すると bean の `setUserData` メソッドが呼び出されます。

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・ヘルプのスケルトンと 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・ヘルプのスケルトンと 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はありません。

```
<!-- 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・コードのスケルトンも生成されます。**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 gettor メソッドと settor メソッドがスケルトン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 ビルダーによりスケルトンの 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);
    }
}
```

呼び出し側プログラムで **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。

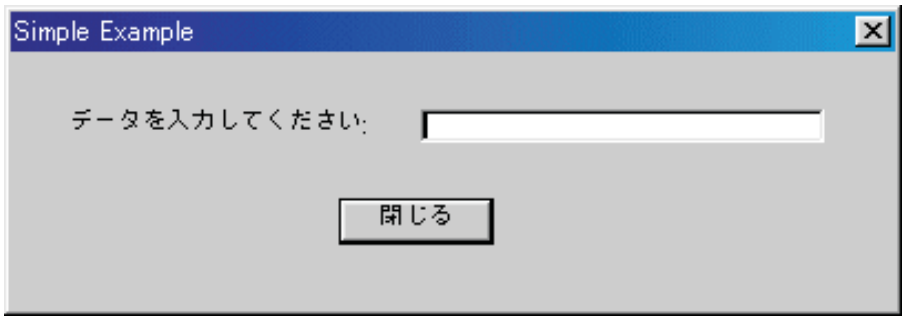
クラス **com.ibm.as400.ui.framework.java.PanelManager** には、スタンドアロン型のウィンドウとダイアログを表示するための API が備えられています。コンストラクターに備えられている PDML ファイルの名前は、 Graphical Toolbox ではリソース名として扱われます。 PDML を含むディレクトリー、ZIP ファイル、または JAR ファイルはクラスパスにより識別されます。

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、このウィンドウはモーダル・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。このウィンドウはモーダルなので、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

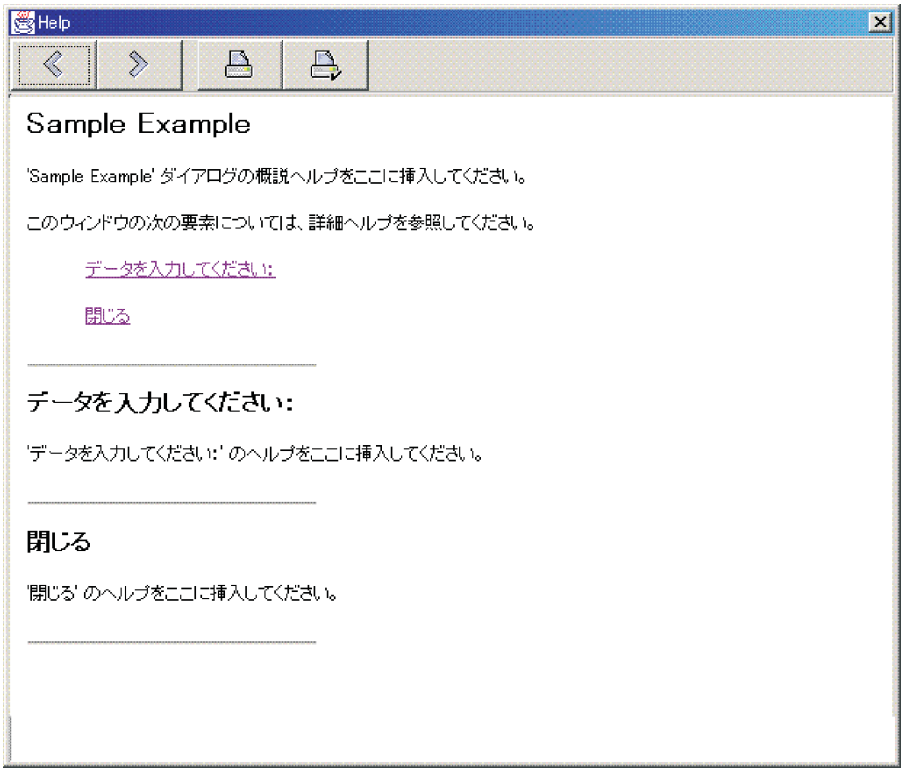
アプリケーションがコンパイルし実行すると、以下のようなウィンドウが表示されます。

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・ヘルプのスケルトンが入ります。

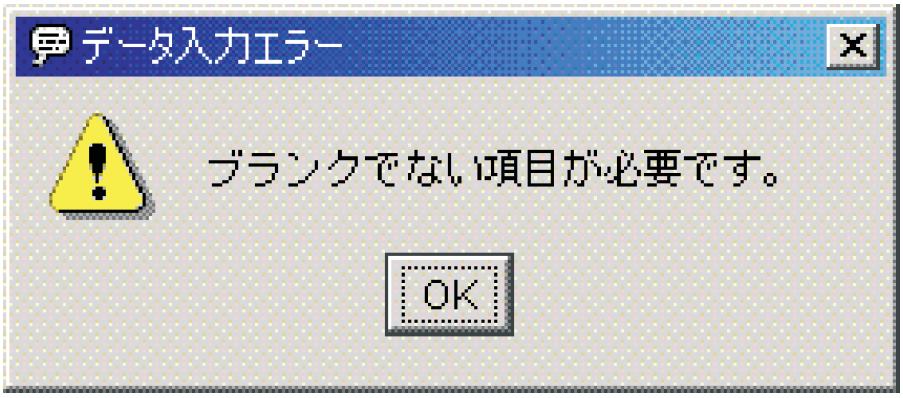
図 11: 「Simple Example」オンライン・ヘルプのスケルトン



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・エラー (Data 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。

関連情報

パッケージ com.ibm.as400.ui.framework.java の要約

編集可能コンボ・ボックス

bean 生成プログラムが編集可能コンボ・ボックスの 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 型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 型が検出される場合、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 型を受け取ることを予期します。

以下の例では、Editable が 編集可能なコンボ・ボックスで、 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 型が検出される場合、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 型です。

```
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 */ }
    }
```

例: RecordListFormPane を使用する

このプログラム例は、サーバー上でファイル内容を収容するフォームを示します。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// RecordListFormPane example.  This program presents a form that contains
// the contents of a file on the server.
//
// Command syntax:
//    RecordListFormPaneExample system fileName
//
// This source is an example of IBM Toolbox for Java "RecordListFormPane".
//
///////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class RecordListFormPaneExample
{


    public static void main (String[] args)
    {
        // If a system and fileName was not specified, then display
        // help text and exit.
        if (args.length != 2)
        {
            System.out.println("Usage:  RecordListFormPaneExample system fileName");
            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.
            JFrame f = new JFrame ("RecordListFormPane example");

            // Create an error dialog adapter.  This will display
            // any errors to the user.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

            // Create a record list form pane to present the contents
            // of the database.  Note we create the form pane, add
            // the error listener, then set the system and file name.
            // Creating the form pane and setting its parameters
            // can be done in one step as follows:
            //    RecordListFormPane formPane = new RecordListFormPane (system, args[1]);
```

```
            // The potential problem is there is no error listener yet
            // so if the file name is not correct, there is no place
            // to display the error.
            RecordListFormPane formPane = new RecordListFormPane();
            formPane.addErrorListener (errorHandler);
            formPane.setSystem(system);
            formPane.setFileName(args[1]);

            // Retrieve the information from the system.
            formPane.load ();

            // When the frame closes, exit.
            f.addWindowListener (new WindowAdapter () {
                public void windowClosing (WindowEvent event)
                {
                    System.exit (0);
                }
            });

            // Layout the frame with the form pane.
            f.getContentPane ().setLayout (new BorderLayout ());
            f.getContentPane ().add ("Center", formPane);
            f.pack ();
            f.show ();
        }
        catch (Exception e)
        {
           System.out.println ("Error: " + e.getMessage ());
           System.exit (0);
        }
    }


}
```

GUI ビルダーによるパネルの作成

以下の解説を参考にして、GUI ビルダーを使用してパネルを作成します。

メニューを作成するには、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

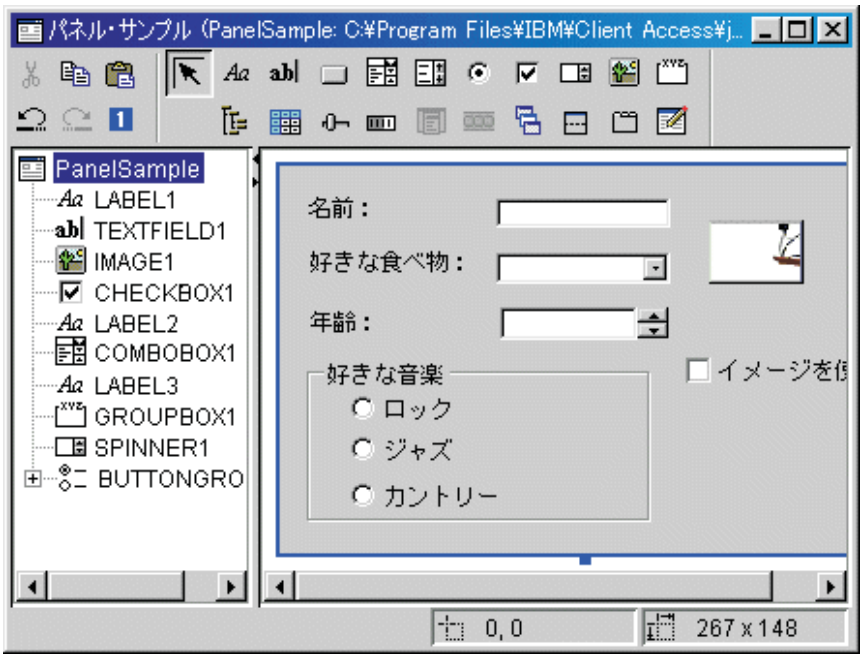


図 1 のサンプル・パネルは、以下のような DataBean コードを使用して、さまざまなコンポーネントを組み合わせます。

```
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。

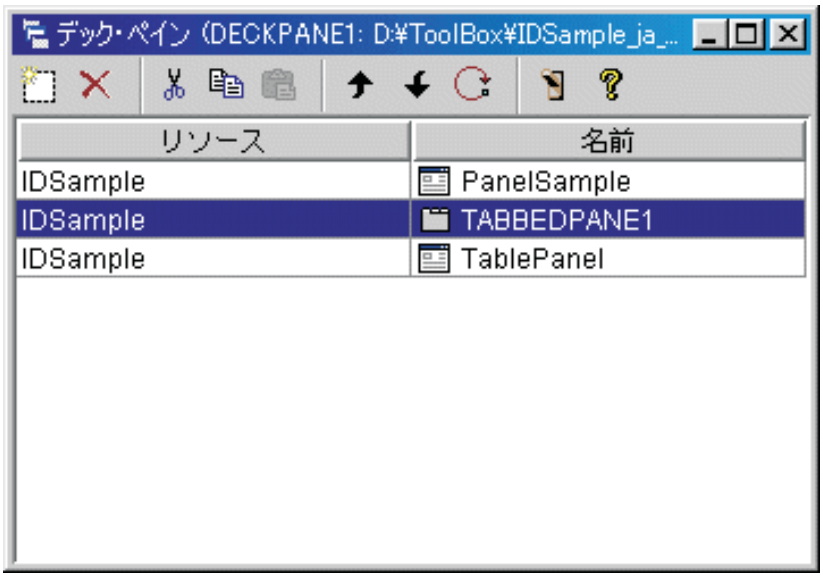
GUI ビルダーを使用してデック・ペインを作成する

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 ビルダーの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。

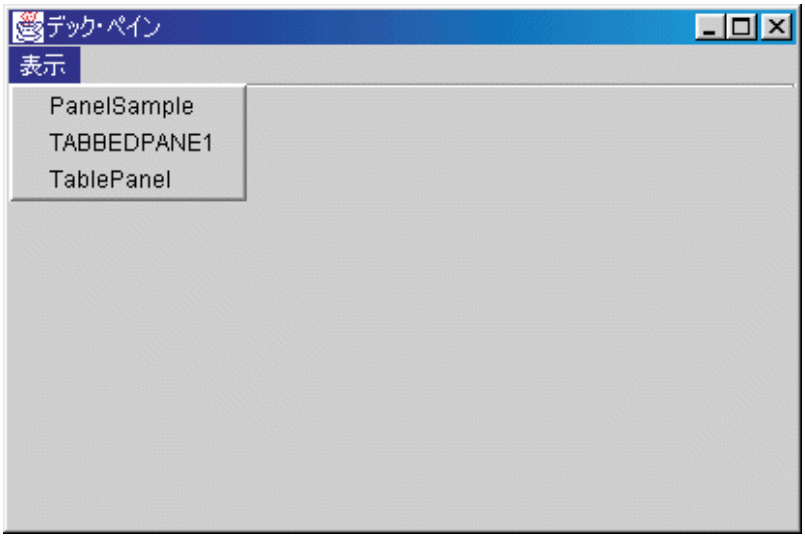
GUI ビルダーの「**ファイル**」ウィンドウのメニュー・バーから、「**デック・ペインの挿入 (Insert Deck Pane)**」ツール・ボタン  をクリックし、デック・ペイン用のコンポーネントを挿入できるパネル・ビルダーを表示します。以下の例では、3 つのコンポーネントが追加されます。

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・ペインには、「**表示**」メニューを選択するまでは何も表示されません。

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

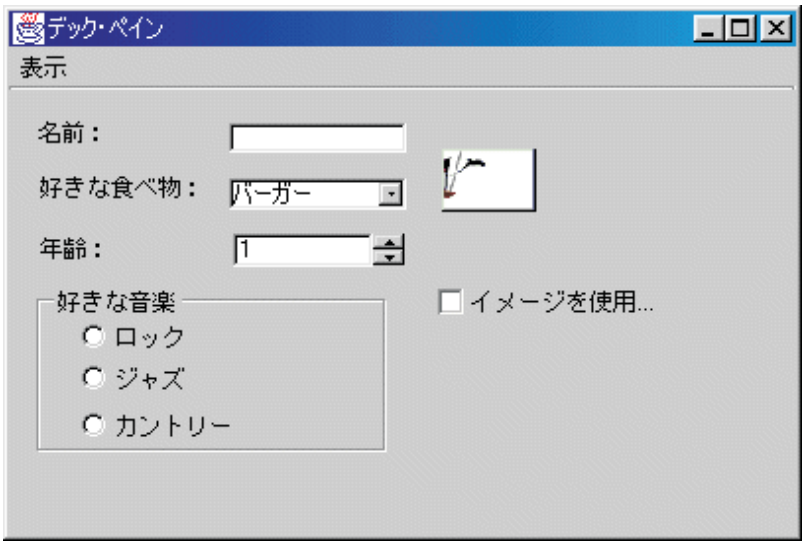
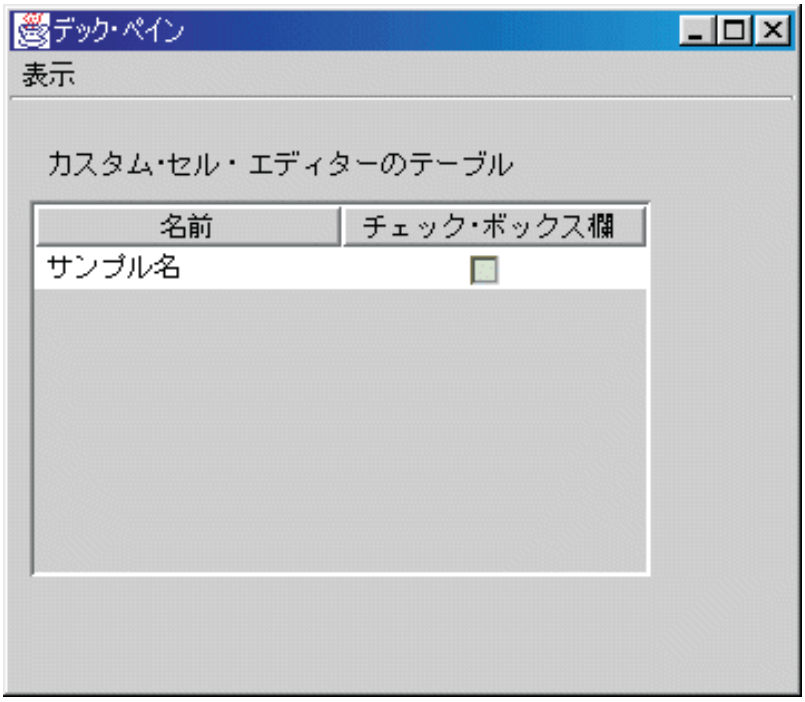


図 4: GUI ビルダーを使用して TABBEDPANE1 を表示する

図 5: GUI ビルダーを使用して TablePanel を表示する



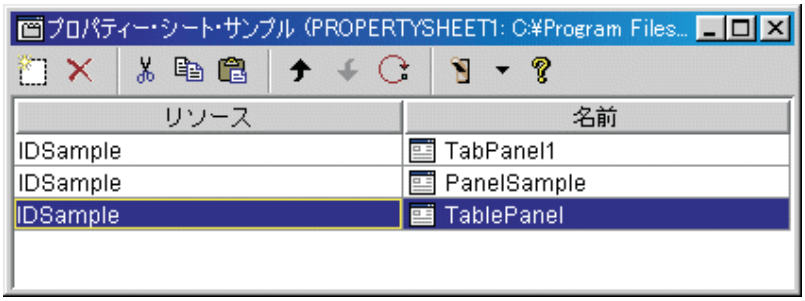
GUI ビルダーによるプロパティー・シートの作成

以下のステップを行って、GUI ビルダーを使用してプロパティー・シートを作成します。

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。

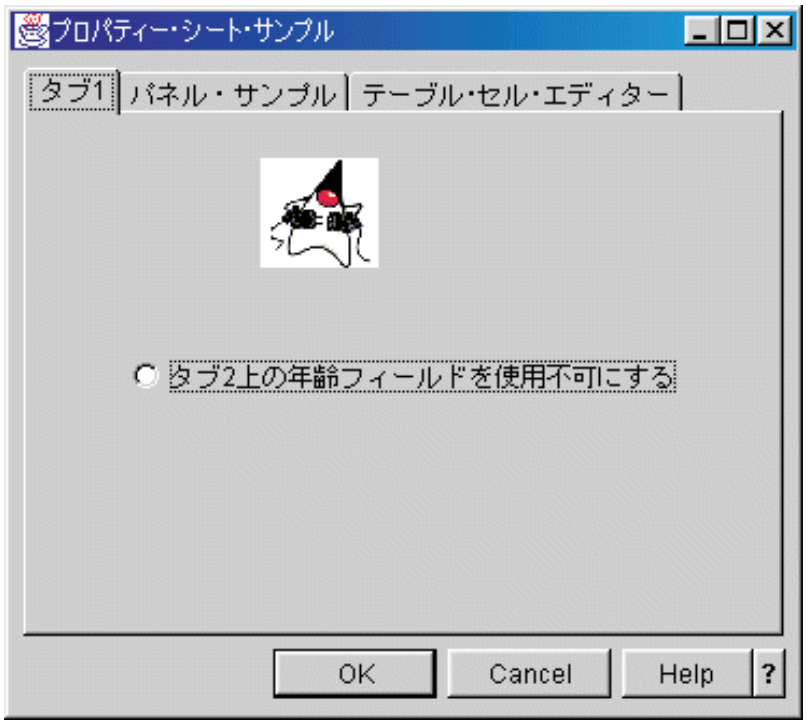
GUI ビルダーの「**ファイル**」ウィンドウのメニュー・バーから、「プロパティー・シートの挿入」アイコン  をクリックして、プロパティー・シートのコンポーネントを挿入できるパネル・ビルダーを表示させます。

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



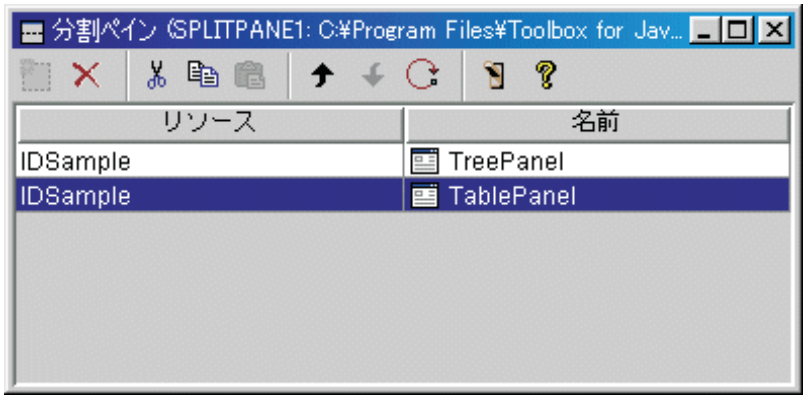
GUI ビルダーを使用して分割ペインを作成する

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。

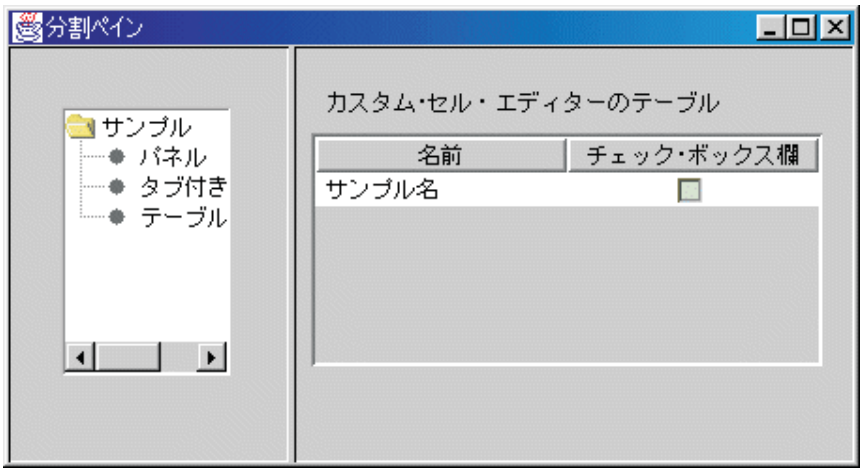
GUI ビルダーの「**ファイル**」ウィンドウのメニュー・バーから、「分割ペインの挿入 (Insert Split Pane)」ツール・ボタン  をクリックして、分割ペインに含めたいコンポーネントを挿入できるパネル・ビルダーを表示させます。以下の例では、2 つのコンポーネントが追加されています。

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



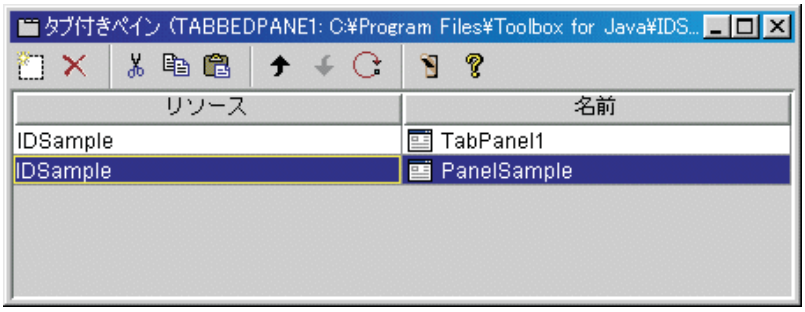
GUI ビルダーを使用してタブ付きペインを作成する

以下の解説を参考にして、タブ付きペインを作成します。

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。

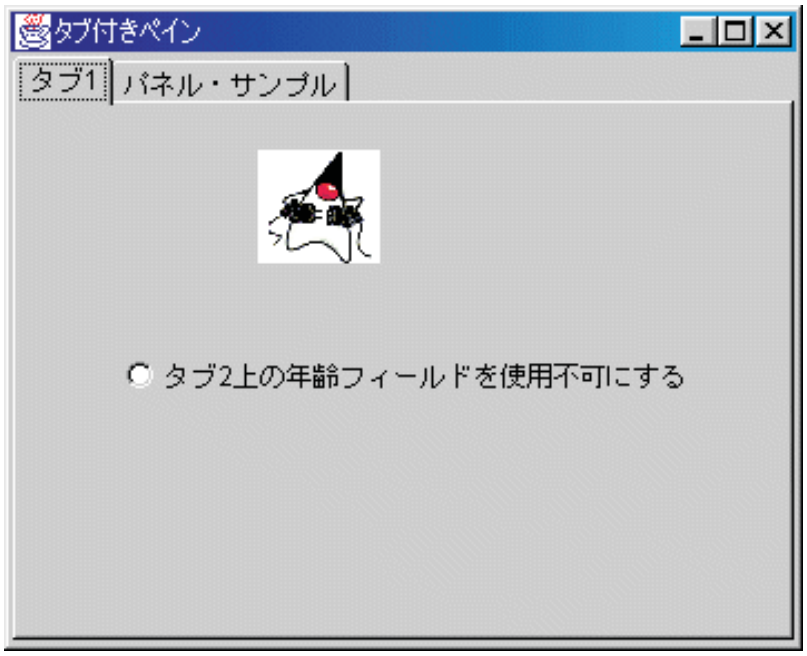
GUI ビルダーの「**ファイル**」ウィンドウのメニュー・バーから、「タブ付きペインの挿入 (Insert Tabbed Pane)」アイコン  をクリックして、タブ付きペインのコンポーネントを挿入できるパネル・ビルダーを表示させます。以下の例では、2 つのコンポーネントが追加されています。

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



GUI ビルダーでウィザードを作成する

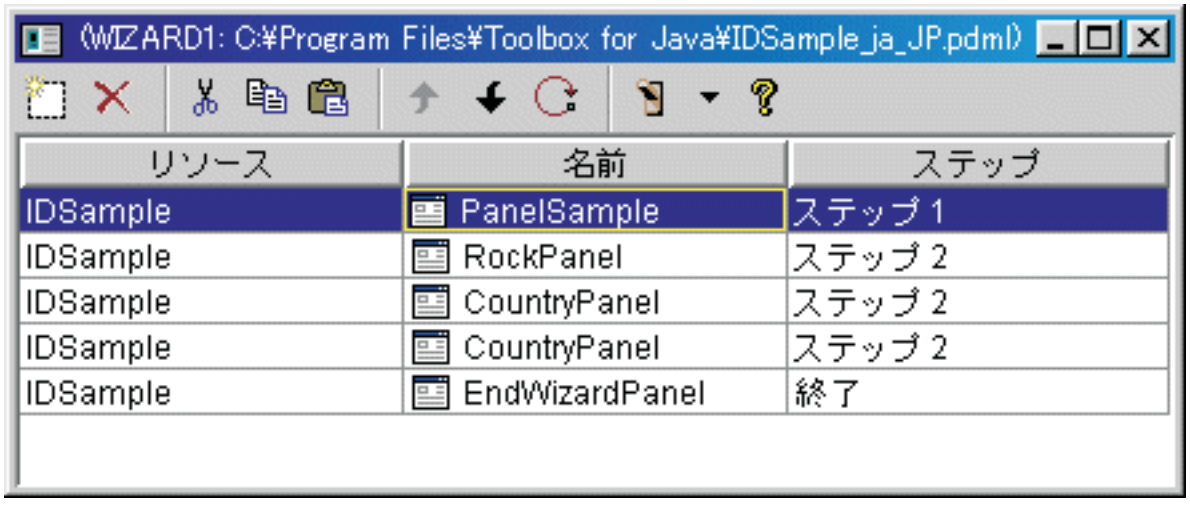
以下の解説を参考にして、ウィザードを作成します。

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

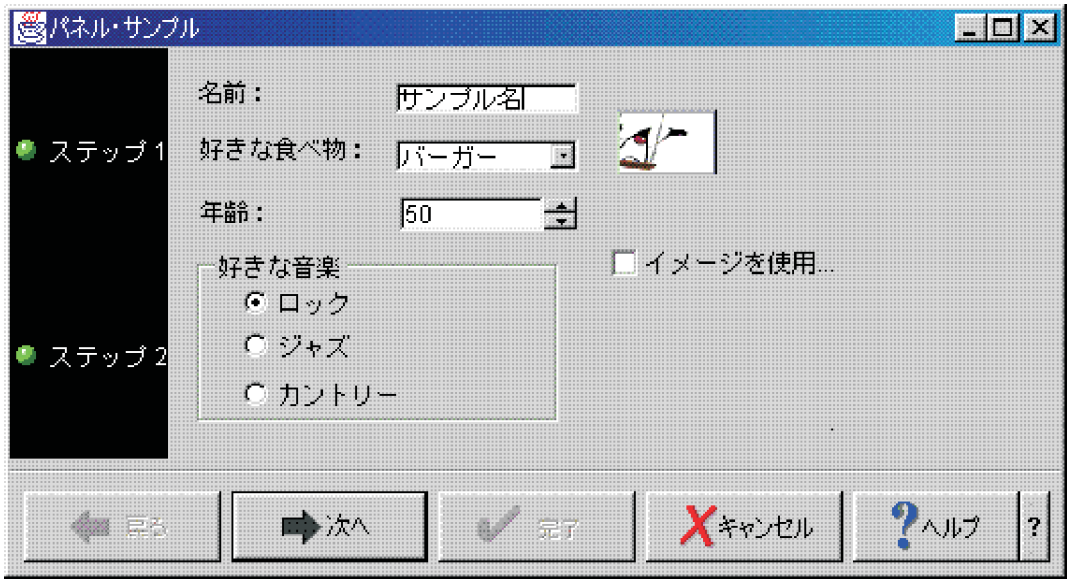
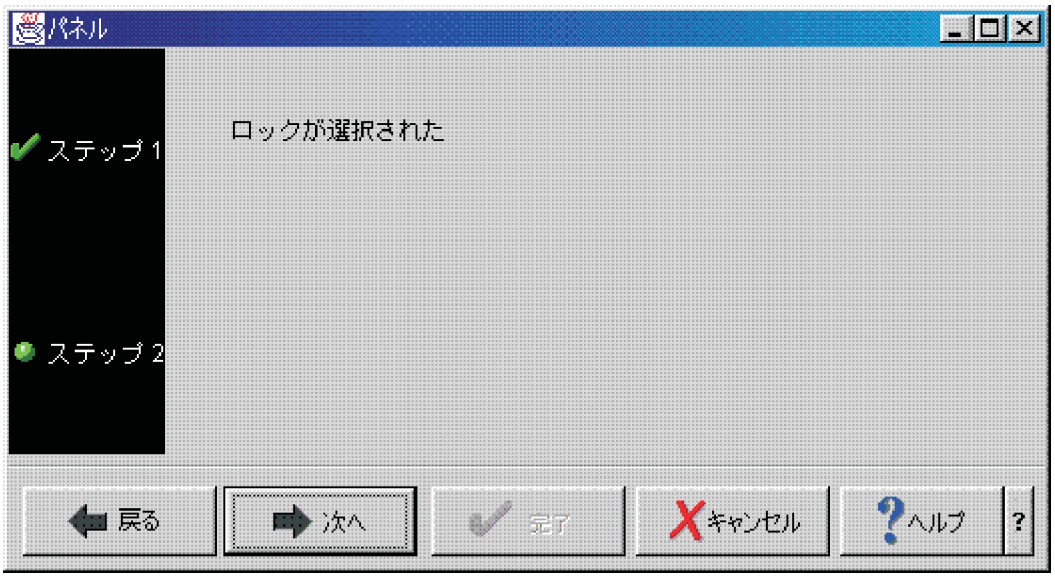


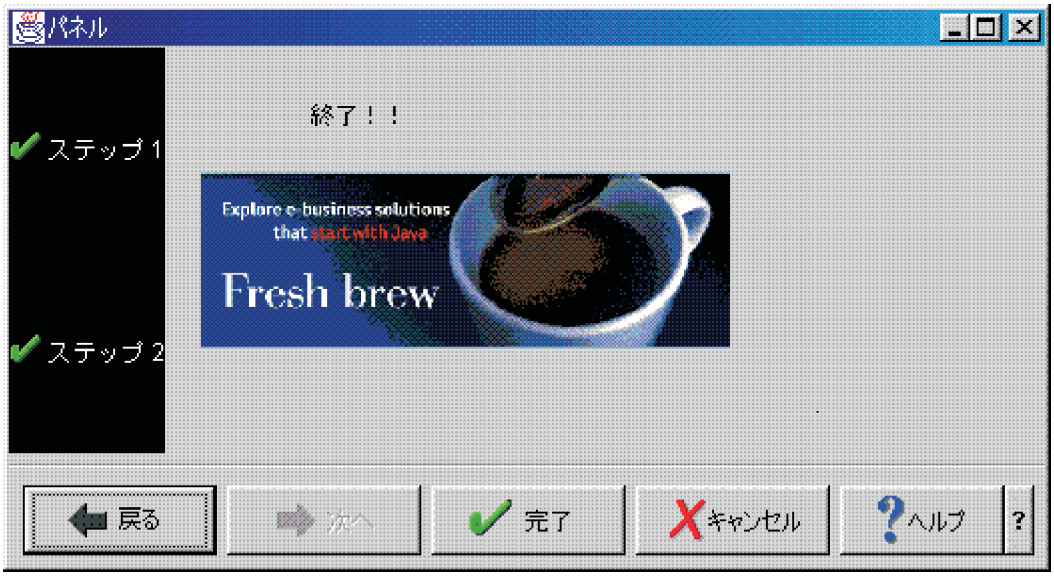
図 2 は、ユーザーが「**ロック**」を選択して、「**次へ**」をクリックした時に表示される 2 番目のパネルを示しています。

図 3: 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**」をクリックすると、図 4 に示されているように、最後のパネルが表示されます。

図 4: 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



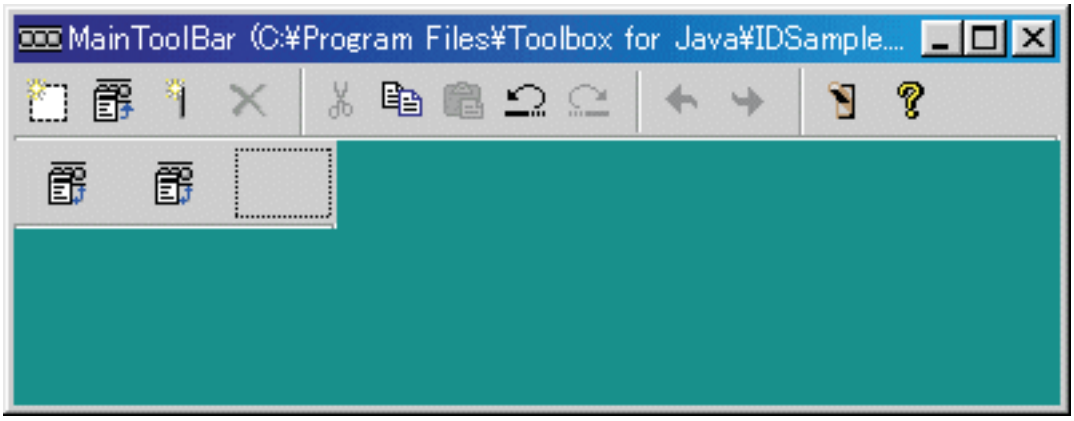
GUI ビルダーを使用してツールバーを作成する

以下の解説を参考にして、GUI ビルダーを使用してツールバーを作成します。

GUI ビルダーの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。

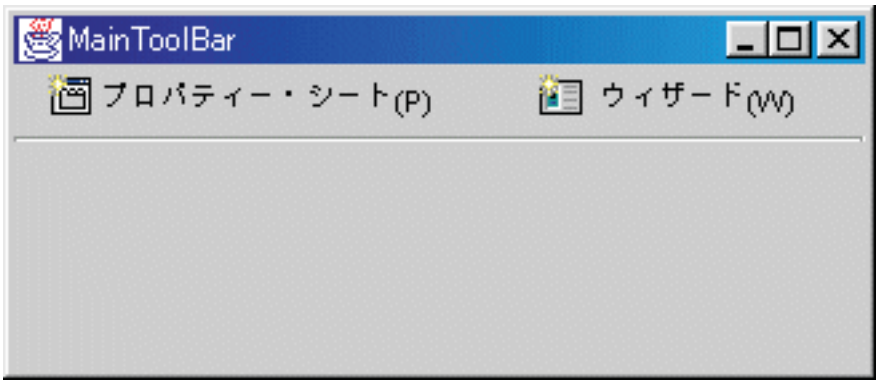
GUI ビルダーの「**ファイル**」ウィンドウのメニュー・バーから、「**ツールバーの挿入 (Insert Tool Bar)**」ツール・ボタンをクリックし、ツールバーのコンポーネントを挿入できるパネル・ビルダーを表示させます。

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



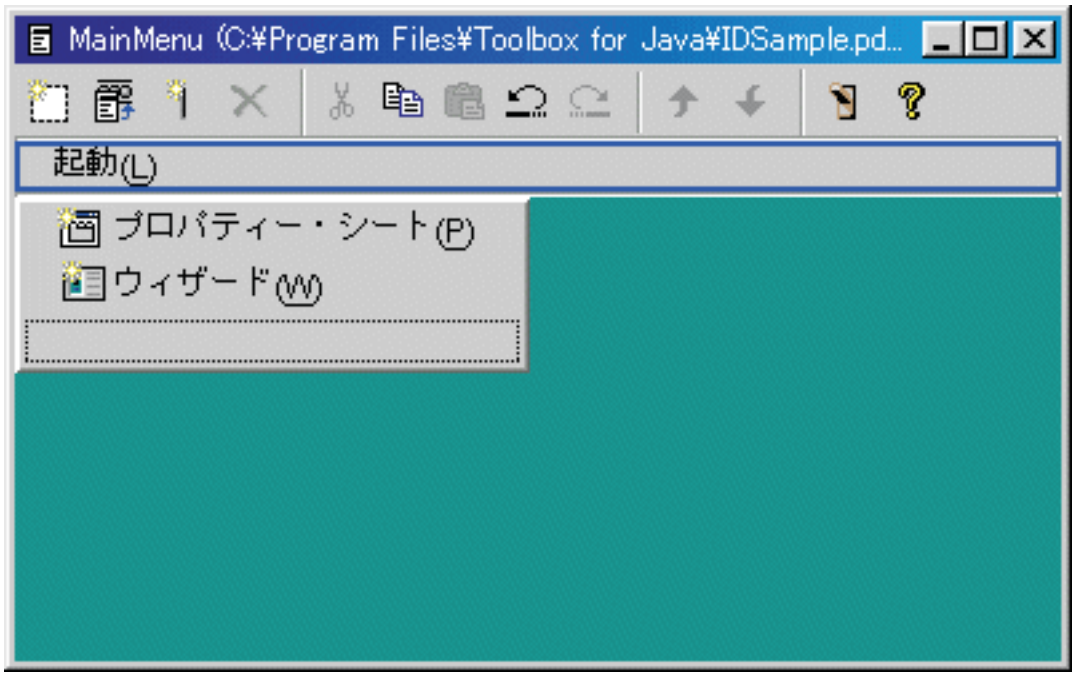
GUI ビルダーでメニュー・バーを作成する

以下の解説を参考にして、メニュー・バーを作成します。

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

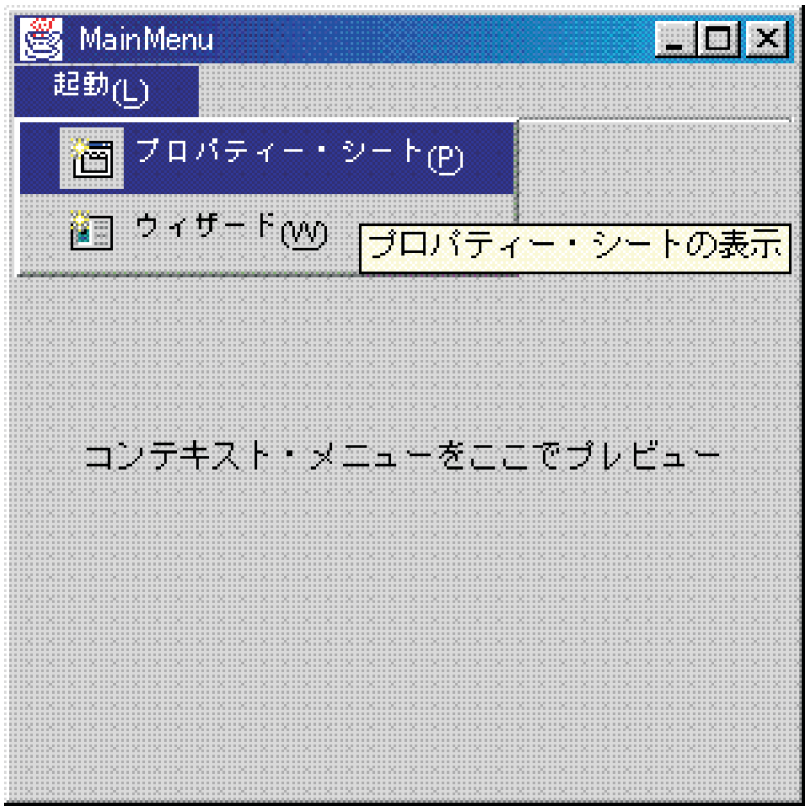
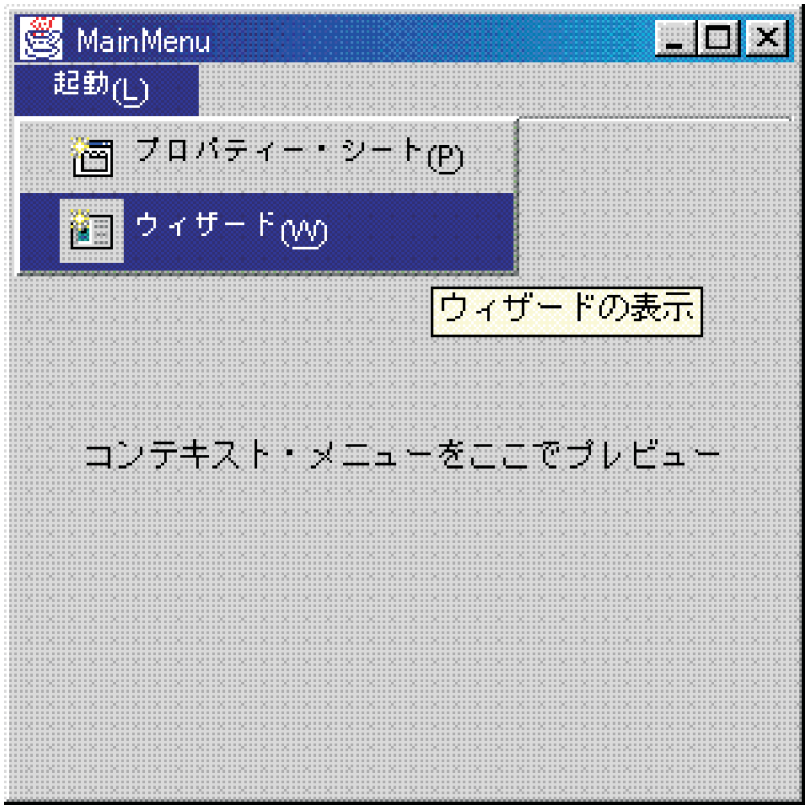


図 3: GUI ビルダー: 「起動」メニューの「ウィザード」を表示する

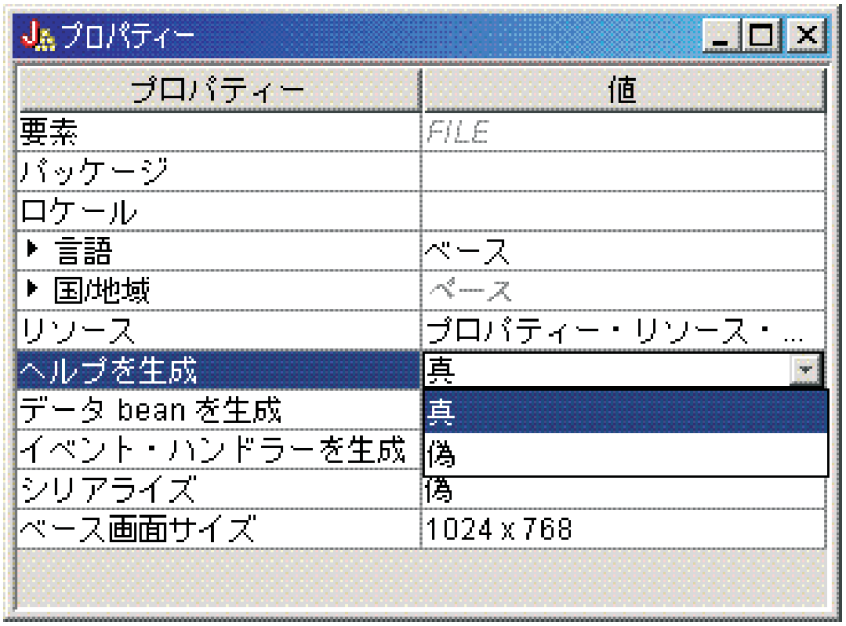


例: ヘルプ文書の作成

このトピックでは、GUI ビルダーを使ったヘルプ・ファイルの作成方法を説明します。

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。

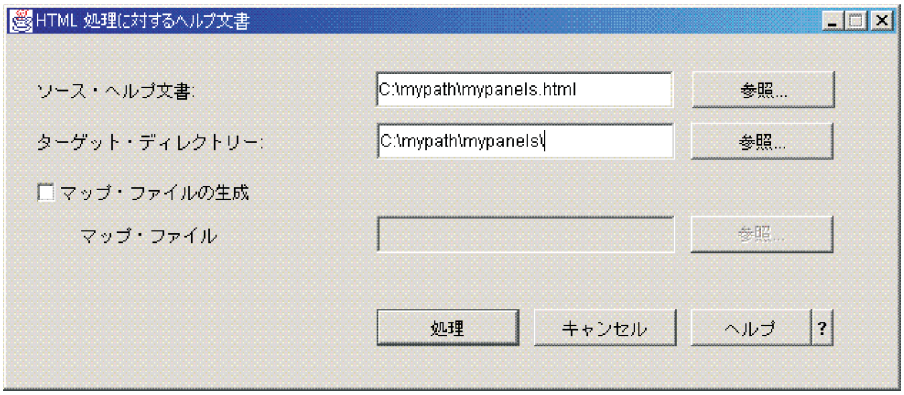
図 1: GUI ビルダーの「プロパティー」パネルの「ヘルプを生成」プロパティーを設定する



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。

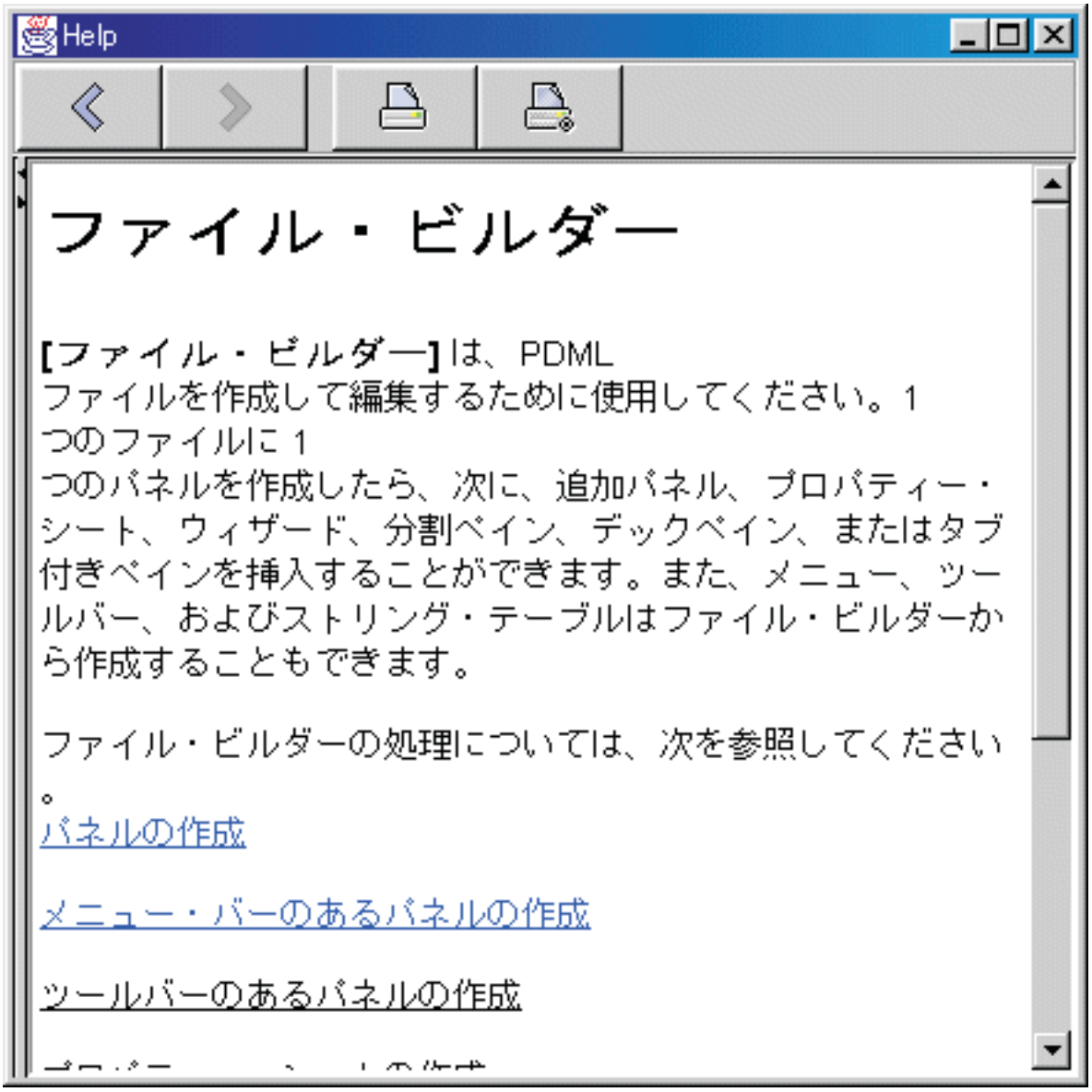
ヘルプ文書を分割するには、コマンド行で以下のようなコマンドを使います。

```
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 (オプション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。

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。

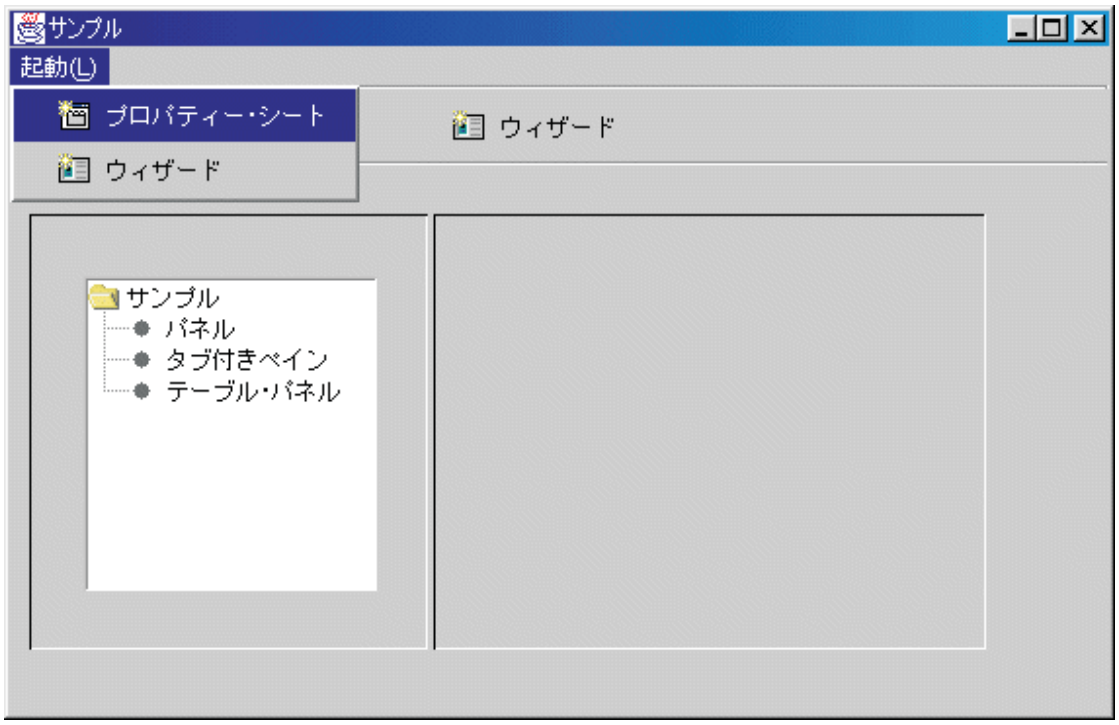
このパネルを使用すると、以下の処理を行うことが可能になります。

- 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

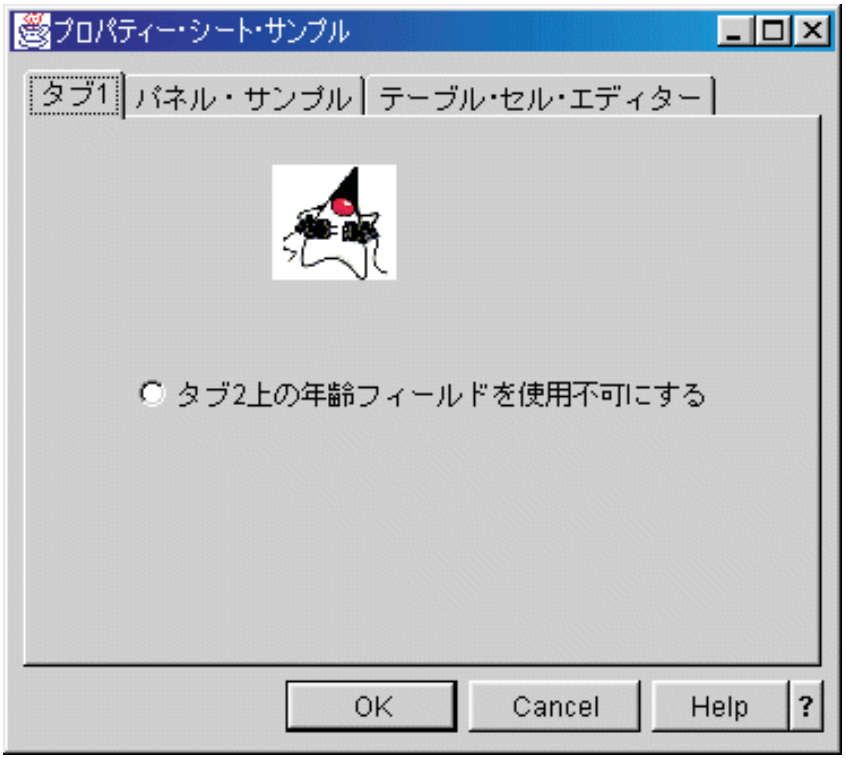
ツールバーの「プロパティー・シート」ボタンをクリックするか、「**起動**」メニューを使用することにより、プロパティー・シートを起動できます。ツールバーかメニューのどちらかを選択できるということは、メニュー項目をリンクするということを意味しています。図 4 は、GUI ビルダーのサンプル・メイン・ウィンドウの「**起動**」メニューから、「**プロパティー・シート**」が選択されている様子を示しています。

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



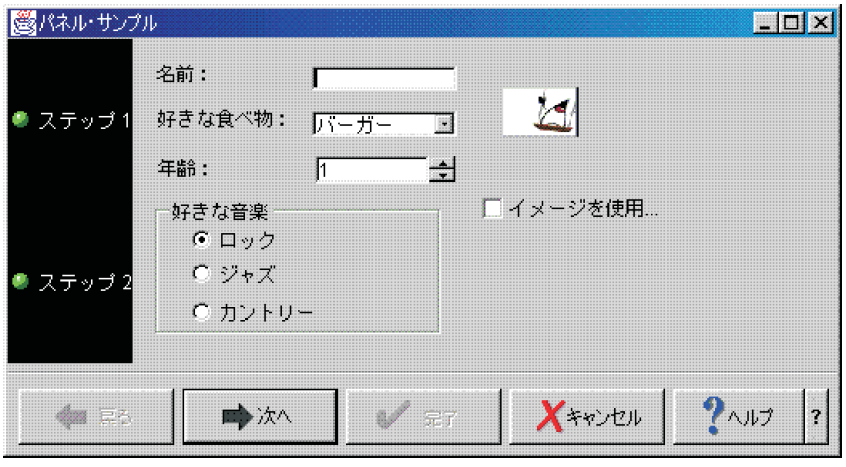
「プロパティー・シート・サンプル」が最初に表示される時、「タブ 1」がデフォルトで表示されます。

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。

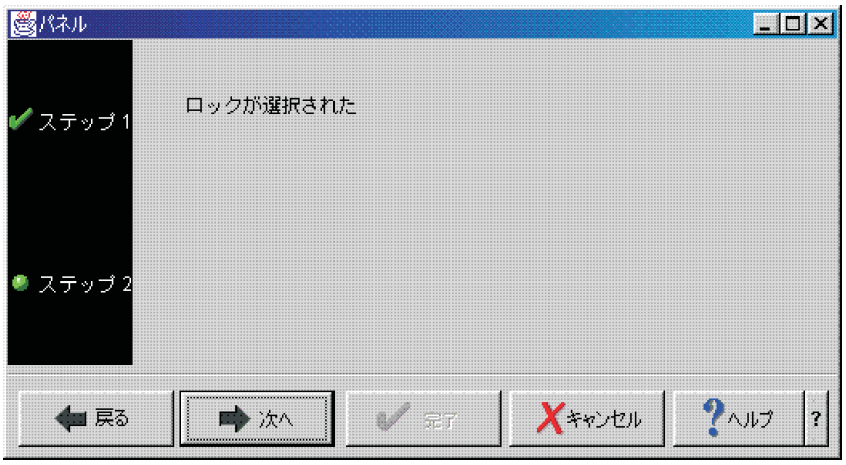
図 9 は、どのように最初のウィザード・ダイアログで多くのオプションが提供されるかを示しています。

図 9: 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、「**次へ**」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、「**次へ**」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、「**次へ**」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「次へ」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

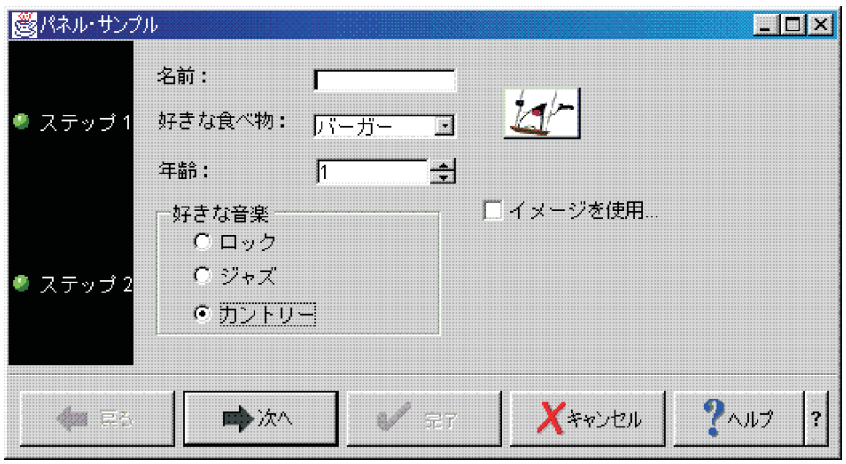


図 13: 2 番目のウィザード・ダイアログ (「カントリー」を選択した後)

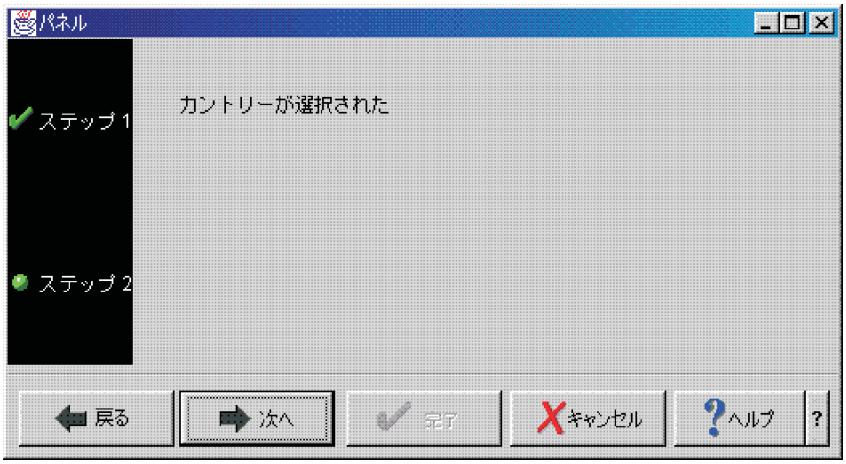
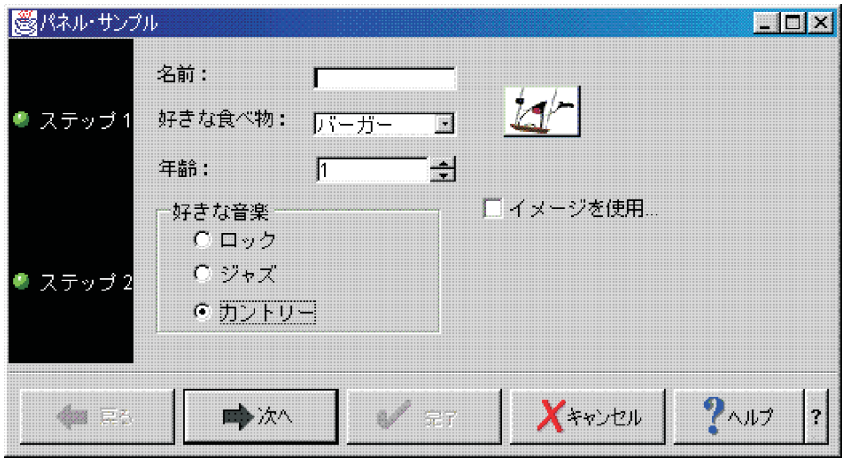


図 14: 最初のウィザード・ダイアログへのループバック

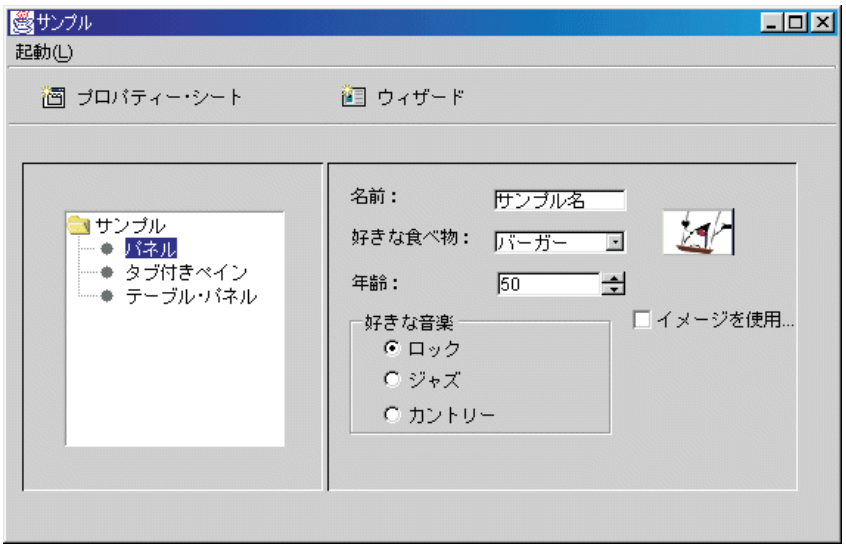


つまり、お気に入りの音楽ジャンルとしてカントリーを選択できないようになっています。

サンプルの表示

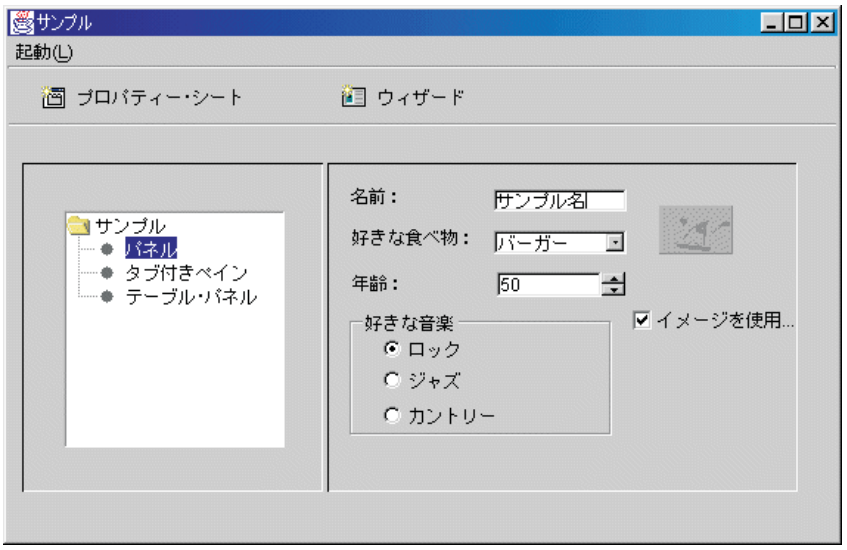
GUI ビルダーのサンプル・メイン・ウィンドウで、ツールバーの下の左側のペインから、その他の機能を選択することもできます。図 15 は、左側のペインで「**パネル**」を選択すると、どのように右側のペインにパネル・サンプルが表示されるかを示しています。

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、図 16 に示されているように、「**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

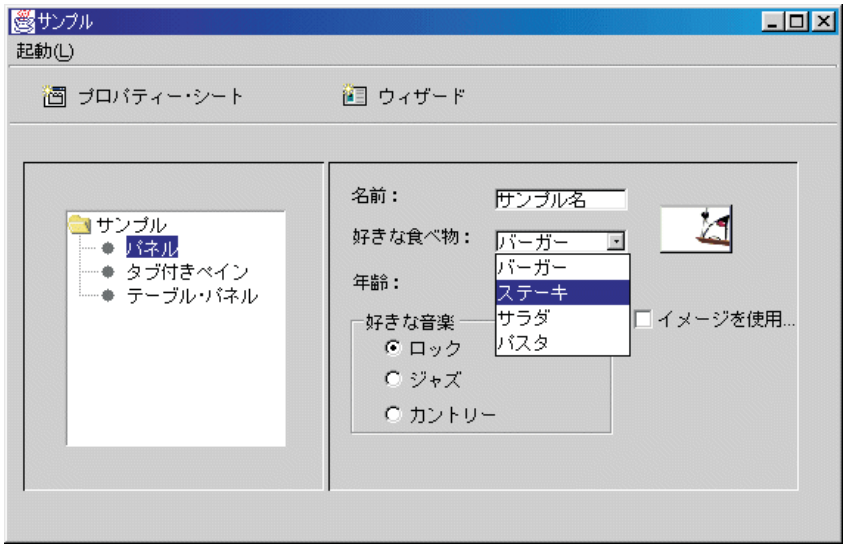


図 18 では、 GUI ビルダーのサンプル・メイン・ウィンドウの左側のペインでの「**タブ付きペイン**」の選択により、どのように右側のペインにタブ付きペインのサンプルが表示されるかを示しています。

図 18: 左側のペインでの「タブ付きペイン」の選択

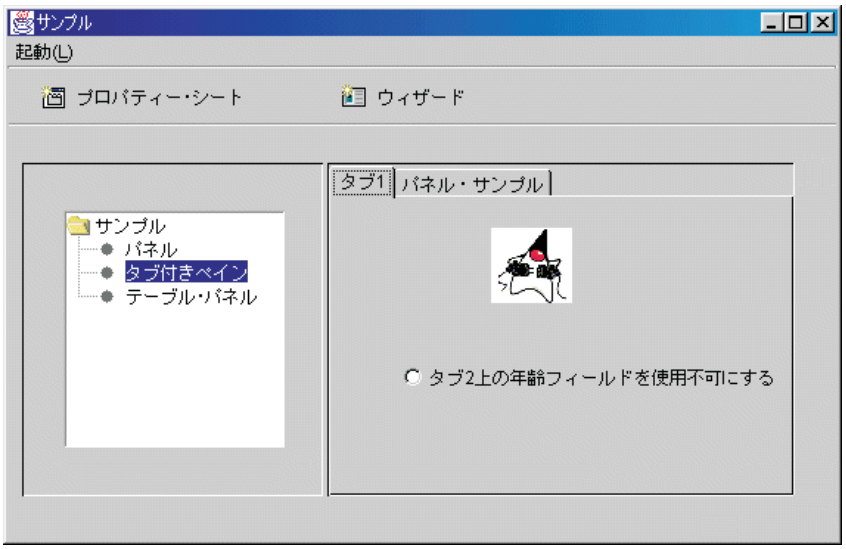
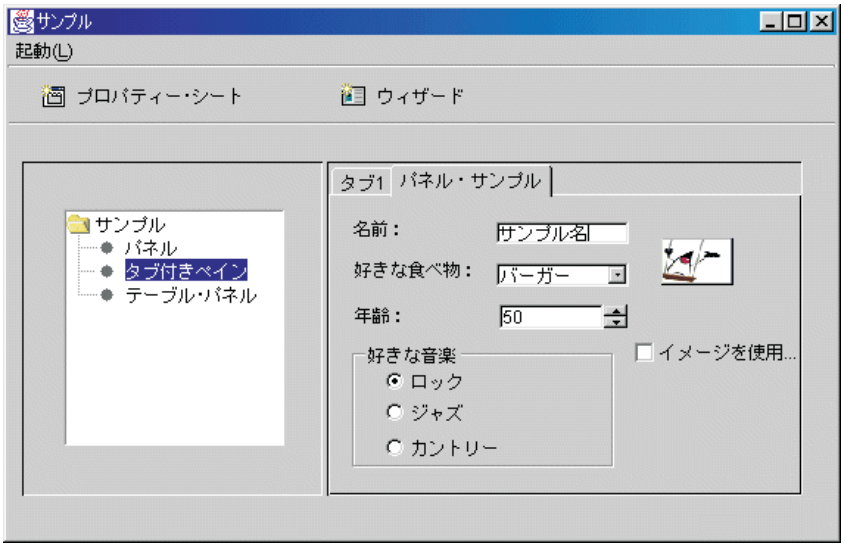


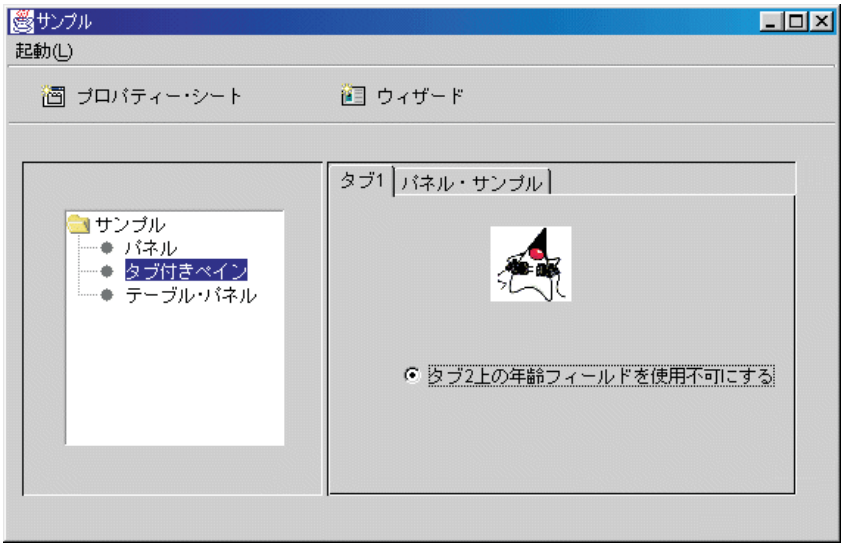
図 19 では、右側のペインでの「**パネル・サンプル**」タブの選択の結果を示しています。

図 19: 右側のペインでの「パネル・サンプル」タブの選択



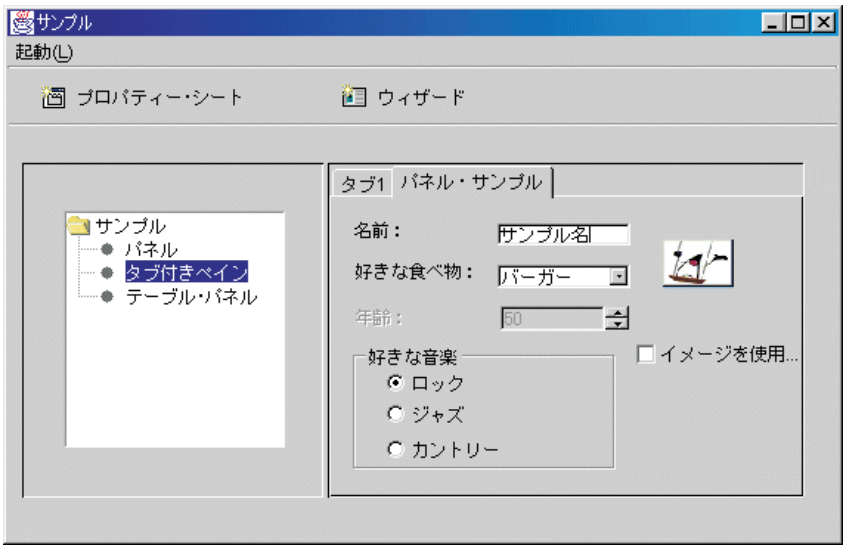
(右側のペインの)「**タブ 1**」を再度選択し、「**タブ 2 上の年齢フィールドを使用不可にする**」をクリックし、チェックを外します。

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



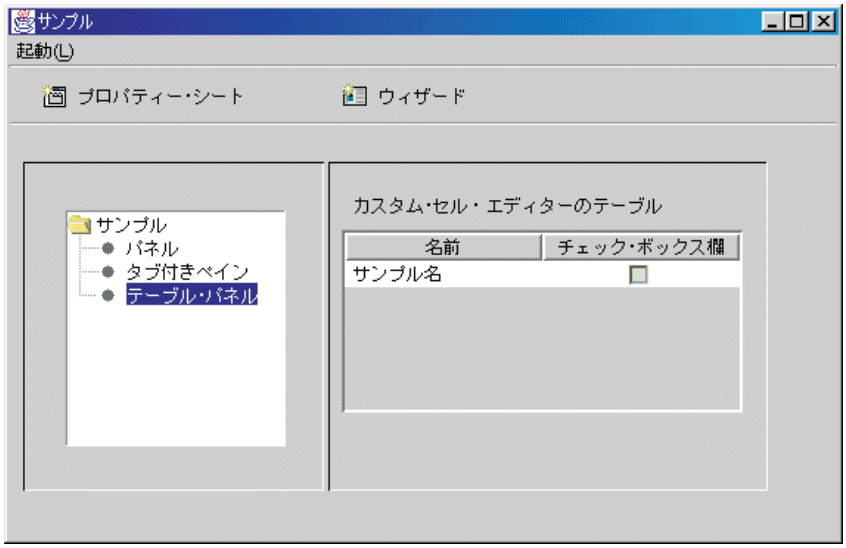
「**タブ 2 上の年齢フィールドを使用不可にする**」オプションを選択すると、図 21 に示されているように、「**パネル・サンプル**」タブの「**年齢**」フィールドが使用不能になり、ぼかし表示になります。

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



HTML クラスの例

以下の例では、IBM Toolbox for Java 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
- HTMLDocument クラスの例:
  - 例: HTMLDocument を使用して HTML データを作成する
  - 例: HTMLDocument を使用して XSL FO データを作成する
- 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: 順序なしリストを作成する
- 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。

コードに関するライセンス情報および特記事項

IBM は、お客様に、すべてのプログラム・コードのサンプルを使用することができる非独占的な著作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。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

こ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、いかなる保証も適用されません。商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任の保証の適用も一切ありません。

例: HTML フォーム・クラスを使用する

以下の IBM Toolbox for Java 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、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////////////////////
//
// This source is an example of using the 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
{
  // Determines if user already exists in the list of registrants.
  private static boolean found = false;

  // Registration information will be stored here
  String regPath = "c:¥¥registration.txt";


  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 IBM i servers" from form
data = req.getParameter("list1");
if (data != null)
{
   output.write("Number of IBM i 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 about 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",
                                        "System i",
                                        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 System i 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);
// 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();

      }
}
```

HTML クラスの出力例

以下に示すのは、HTML クラスの例を実行すると生成されるサンプル出力です。

- Customer Name: Fred Flinstone
  Email address: flinstone@bedrock.com
  Currently Using Toolbox: yes
  Requested More Information: yes
  Multiple Versions: v4r2,v4r4
  Using Java or Interested In: applications,servlets
  Platforms: NT,Linux
  Number of System i servers: three
  Comments: The Toolbox is being used by our entire Programming department to build customer applications!
  Attachment File: U:¥wiedrich¥servlet¥temp.html
  (C) Copyright IBM Corp. 1999, 1999
- Customer Name: Barney Rubble
  Email address: rubble@bedrock.com
  Currently Using Toolbox: yes
  AS400 Version: v4r4
  Using Java or Interested In: servlets
  Platforms: OS2
  Number of System i servers: FiveOrMore
  (C) Copyright IBM Corp. 1999, 1999
- Customer Name: George Jetson
  Email address: jetson@sprocket.com
  Requested More Information: yes
  AS400 Version: v4r2
  Using Java or Interested In: applications
  Platforms: NT,Other
  Other Platforms: Solaris

Number of System i servers: one
Comments: This is my first time using this! Very Cool!
(C) Copyright IBM Corp. 1999, 1999

- Customer Name: Clark Kent
  Email address: superman@krypton.com
  AS400 Version: v4r2
  Number of System i servers: one
  (C) Copyright IBM Corp. 1999, 1999

関連資料

645 ページの『例: HTML フォーム・クラスを使用する』
以下の IBM Toolbox for Java 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。

例: HTMLTree クラスを使用する

この例は、IBM Toolbox for Java HTML パッケージ・クラスを使用して、HTML およびファイル・ツリーを作成する方法を示します。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////////////////////
//
// 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 (3 つのファイルの中の 1 つ)

この IBM Toolbox for Java コード例は、他の 2 つのファイル例とともに、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、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////////////////////
//
// This source is an example of using the 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 (3 つのファイルの 2 番目)

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、 コードに関するライセンス情報および特記事項をお読みください。

```
///////////////////////////////////////////////////////////////////////////////
//
// 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");

      // IBM Toolbox for Java uses a set of default icons to represents expanded,
      // collapsed, and documents within the HTMLTree. To enhance those icons,
      // IBM Toolbox for Java 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 (3 つのファイルの 3 番目)

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、 コードに関するライセンス情報および特記事項をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// 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());
            }
            // Display this HTMLHeading if no FileTreeElement has been selected.
            else
            {
                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。

表 3. Customer Account Balances - January 1, 2000

ACCOUNT	NAME	BALANCE
0000001	Customer1	100.00

表 3. Customer Account Balances - January 1, 2000 (続き)

ACCOUNT	NAME	BALANCE
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 IBM i 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。

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、以下のアクションを実行する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 IBM i 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。

コードに関するライセンス情報および特記事項

IBM は、お客様に、すべてのプログラム・コードのサンプルを使用することができる非独占的な著作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。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

こ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、いかなる保証も適用されません。商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任の保証の適用も一切ありません。

例: 簡単なデータ検索例

以下の例は、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。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

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

以下の例は、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。

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

以下の例は、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。

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、IBM Toolbox for Java 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。

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。

コードに関するライセンス情報および特記事項

IBM は、お客様に、すべてのプログラム・コードのサンプルを使用することができる非独占的な著作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。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

こ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、いかなる保証も適用されません。商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任の保証の適用も一切ありません。

例: PDFContext と共に JSPReportProcessor を使用する

この例は JSPReportProcessor および PDFContext クラスを使用して、指定した URL からデータを取得し、そのデータを PDF フォーマットに変換します。そのデータは次に、PDF 文書としてファイルにストリームされます。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

JSPRunReport と一緒に使用できる JSP ソース・ファイルの例の内容を表示するには、JSPcust_table.jsp を参照してください。 JSP 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。この ZIP 圧縮されたファイルには、XSLReportProcessor の例 (PCLRunReport) と一緒に使用できる XML および XSL ファイルの例も含まれます。

```
////////////////////////////////////////////////////////////////////////////////
//
// 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. そのデータは次に、PDF 文書としてファイルにストリームされます。
//
// 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

このサンプル・ファイルは、『例: PDFContext と共に JSPReportProcessor を使用する』トピックと一緒に使用します。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
<?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","Or","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

今後、XSLReportProcessor クラスはサポートされないので、以下の例は、使用しないでください。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////////////////////////
//
// 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 server name       */
        String user = "<user>";       /* Insert user profile name */
        String pass = "<password>";   /* Insert password          */

        AS400 system = new AS400(sys, user, pass);

        /* Insert 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

今後、XSLReportProcessor クラスはサポートされないので、以下の例は、使用しないでください。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
<?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

今後、XSLReportProcessor クラスはサポートされないので、以下の例は、使用しないでください。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
<?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、IBM Toolbox for Java 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

- 例: 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。

コードに関するライセンス情報および特記事項

IBM は、お客様に、すべてのプログラム・コードのサンプルを使用することができる非独占的な著作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。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

こ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、いかなる保証も適用されません。商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任の保証の適用も一切ありません。

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、システム・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();
```

コードのサンプルに関する特記事項

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。

IBM は、お客様に、すべてのプログラム・コードのサンプルを使用することができる非独占的な著作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。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

こ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、いかなる保証も適用されません。商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任の保証の適用も一切ありません。

関連情報

RJobList Javadoc

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: Resource からの属性値の取り出し

IBM Toolbox for Java Resource の1 つの具象サブクラスは com.ibm.as400.resource.RUser ですが、これは IBM i 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);
```

関連情報

RUser Javadoc

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、これはシステム・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();
```

RJob Javadoc

例: 汎用コードを使用してリソースにアクセスする

Resource、ResourceList、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、ResourceList、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。

例

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

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);
    }
}
```

Example: Using ResourceMetaData to reset every attribute of a ChangeableResource

This is an example of generic code that resets all attributes of a ChangeableResource to their default values:

```
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();
}
```

関連情報

ResourceMetaData Javadoc

例: RFML

この項では、IBM Toolbox for Java RFML コンポーネントの資料全体で取り上げられているコード例をリストしています。

- 例: RFML の使用と IBM Toolbox for Java Record クラスの使用の比較
- 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。

コードに関するライセンス情報および特記事項

IBM は、お客様に、すべてのプログラム・コードのサンプルを使用することができる非独占的な著作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。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

こ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、いかなる保証も適用されません。商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任の保証の適用も一切ありません。

例: RFML ソース・ファイル

この RFML ソース・ファイル例では、 IBM 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、 コードに関するライセンス情報および特記事項をお読みください。

```
<?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>
```

関連資料

444 ページの『例: IBM 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 IBM Toolbox for Java の Record クラスを使用する時の相違点を示します。

例: プロファイル・トークン信任状を使用して IBM i スレッド ID を交換する

以下のコード例では、プロファイル・トークン信任状を使用して、IBM i スレッド ID を交換し、特定のユーザーに代わって代行処理作業を実行する方法を示します。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
| // Create a single-use ProfileTokenCredential with a 60 second timeout.
| AS400 system = new AS400();
| ProfileTokenCredential pt = new ProfileTokenCredential();
| pt.setSystem(system);
| pt.setTimeoutInterval(60);
| pt.setTokenType(ProfileTokenCredential.TYPE_SINGLE_USE);
|
| // A valid user ID and password must be substituted.
| pt.setTokenExtended("user", "password");
|
| // Swap the OS/400 thread identity, retrieving a credential to
| // Swap back to the original identity later.
| AS400Credential cr = pt.swap(true);
|
| // Create a second AS400 object
| AS400 system2 = new AS400();
| CommandCall cmd = new CommandCall(system);
|
| // 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();
| system.disconnectAllServices();
| system2.disconnectAllServices();
```

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 クラスを使用する
- 例: RecordListRowData クラスを使用する
- 例: SQLResultSetRowData クラスを使用する
- 例: HTMLFormConverter クラスを使用する
- 例: ListMetaData クラスを使用する
- 例: SQLResultSetMetaData クラスを使用する
- 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。

コードのサンプルに関する特記事項

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。

IBM は、お客様に、すべてのプログラム・コードのサンプルを使用することができる非独占的な著作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。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

こ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、いかなる保証も適用されません。商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任の保証の適用も一切ありません。

例: ListRowData クラスを使用する

3 つの部分に分かれたこの例は、ListRowData クラスを使用して HTML を生成し、表示する方法を示しています。

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』
- 701 ページの『HTMLTableConverter を使用して Java ソースから生成された HTML ソース』
- 701 ページの『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た HTML ソース

上記の Java ソースの例における 256 ページの『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 クラスを使用する

この例は、RecordListRowData クラスがどのように稼働するかを示し、HTMLTableConverted クラスを使用して生成された HTML ソースを示し、さらに、生成された HTML がブラウザーでどのように表示されるかを示します。

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た 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>CNUM</th>
<th>LNAM</th>
<th>INIT</th>
<th>STR</th>
<th>CTY</th>
<th>STATE</th>
<th>ZIP</th>
<th>CTLMT</th>
<th>CHGCOD</th>
<th>BDUE</th>
<th>CTDUE</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。

CNUM	LNAM	INIT	STR	CTY	STATE	ZIP	CTLMT	CHGCOD	BDUE	CTDUE
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 クラスを使用する

この例は、SQLResultSetRowData クラスがどのように稼働するかを示し、生成された HTML ソースを示し、さらに、生成された HTML がブラウザーでどのように表示されるかを示します。

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た 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 クラスを使用する

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 IBM Toolbox for Java 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();
  }

}
```

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。 IBM 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。

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

IBM 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。

コードに関するライセンス情報および特記事項

IBM は、お客様に、すべてのプログラム・コードのサンプルを使用することができる非独占的な著作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。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

こ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、いかなる保証も適用されません。商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任の保証の適用も一切ありません。

初めて IBM 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
   ```

[簡単なプログラミング例]

例: CommandCall を使用する

この例は、IBM Toolbox for Java アクセス・クラス CommandCall を使用します。

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。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////
//
// This source is an example of "Job List".
//
////////////////////////////////////////////////
//
// The access classes are in the com.ibm.as400.access.package.
// Import this package to use the IBM Toolbox for Java classes.
//
////////////////////////////////////////////////

import com.ibm.as400.access.*;

public class CmdCall
{
   public static void main(String[] args)
   {
      // Like other Java classes, IBM Toolbox for Java classes
      // throw exceptions when something goes wrong.  These must
      // be caught by programs that use IBM Toolbox for Java.
      try Note 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. IBM 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、IBM 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. IBM 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、IBM i メッセージのリストになります。 IBM Toolbox for Java 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。

例: メッセージ待ち行列を使用する (3 部の 1)

このソースは、IBM Toolbox for Java メッセージ待ち行列の例です。

[次の部]

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。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////////////////////////
//
// 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 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、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スローします。
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 によってスロー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]

例: メッセージ待ち行列を使用する (3 部の 2)

このソースは、IBM Toolbox for Java メッセージ待ち行列の例です。

[前の部 | 次の部]

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。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////////////////////
//
// 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 IBM i 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]

例: メッセージ待ち行列を使用する (3 部の 3)

このソースは、IBM Toolbox for Java メッセージ待ち行列の例です。

[前の部]

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。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////////////////////
//
// 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]

例: レコード・レベルのアクセスを使用する (2 部の 1)

このプログラムは、サーバーの名前と表示するファイルをユーザーにプロンプトでたずねます。そのファイルは、存在していなければならず、レコードが入っていなければなりません。ファイル内の各レコードは、System.out に表示されます。

[次の部]

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。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////////////////////
//
// 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 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]

例: レコード・レベルのアクセスを使用する (2 部の 2)

この例は、レコード・レベル・アクセスの使用を例示します。

[前の部]

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。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////////////////////////
//
// 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]

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)

このプログラムは、IBM Toolbox for Java JDBC ドライバーを使用して、テーブルを作成し、そこにデータを取り込みます。

[次の部]

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。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////////////////////////
//
// 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 SQL 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]

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)

このプログラムは、IBM Toolbox for Java JDBC ドライバーを使用して、テーブルを照会し、その内容を出力します。

[前の部]

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。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////////////////////////
//
// 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]

例: サーバー・ジョブのリストを GUI で表示する

この例は、IBM Toolbox for Java VJobList クラスを使用します。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////////////////////////
//
// Example using the  vaccess class, VJobList.
//
// This source is an example of "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 の JFC 1.1 に付属しています。
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スローします。
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、ステップ 9 で作成したエラー・ハンドラーを、 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、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、制御フレームの中央に追加します。
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 show 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。

例: プログラミングに関するヒント

このトピックでは、IBM Toolbox for Java プログラミングに関するヒント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。

接続の管理

- 例: CommandCall オブジェクトを使用してシステムへの接続を 1 つ作成する
- 例: CommandCall オブジェクトを使用してシステムへの接続を 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 を使用してシステムに事前接続する
- 例: AS400ConnectionPool を使用してシステム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 プログラムをシステムに事前接続する方法
- 例: Java プログラムをシステムから切断する方法
- 例: disconnectService() と run() を使用して、 Java プログラムをシステムから切断してから、再接続する方法
- 例: Java プログラムをシステムから切断し、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

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。

コードに関するライセンス情報および特記事項

IBM は、お客様に、すべてのプログラム・コードのサンプルを使用することができる非独占的な著作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。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

こ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、いかなる保証も適用されません。商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任の保証の適用も一切ありません。

例: ToolboxME

この項では、 IBM Toolbox for Java 2 Micro Edition の資料全体で取り上げられているコード例をリストします。

- 399 ページの『ToolboxME の例: JdbcDemo.java』
- 『例: ToolboxME、MIDP、および JDBC』
- 755 ページの『例: ToolboxME、MIDP、および IBM Toolbox for Java』

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。

コードに関するライセンス情報および特記事項

IBM は、お客様に、すべてのプログラム・コードのサンプルを使用することができる非独占的な著作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。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

こ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、いかなる保証も適用されません。商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任の保証の適用も一切ありません。

例: ToolboxME、MIDP、および JDBC

以下のソースは、IBM Toolbox for Java 2 Micro Edition 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、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、 コードに関するライセンス情報および特記事項をお読みください。

```
///////////////////////////////////////////////////////////////////////////////
//
// ToolboxME 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 are 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 do not 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 will 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 are 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 do not 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 do not need right now.
     */
    public void pauseApp()
    {
        display.setCurrent(null);
    }

    /**
     * Destroy must cleanup everything.
     */
    public void destroyApp(boolean unconditional)
    {
    }
}
```

関連情報

385 ページの『Mobile Information Device Profile (MIDP)』

例: ToolboxME、MIDP、および IBM Toolbox for Java

以下のソースは、IBM Toolbox for Java 2 Micro Edition 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 IBM i のデータおよびサービスにアクセスしする 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には、Tier0 デバイスがこれらの機能を使用す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があります。

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、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、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////////////////////
//
// ToolboxME example. This program is an example that shows how
// ToolboxME can use PCML to access data and services.
//
// 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("ToolboxME 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);
// TBD temporary
private TextField signonPwdText_ = new TextField("Password", "JTEAM1", 10, TextField.PASSWORD);
private TextField signonServerText_ = new TextField("MEServer", "localhost", 10, TextField.ANY);
private StringItem signonStatusText_ = new StringItem("Status", NOT_SIGNED_ON);

// TextFields for the CommandCall form.
// TBD: max size; TBD: TextBox???
private TextField cmdText_ = new TextField("Command", "CRTLIB FRED", 256, TextField.ANY);
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 server.
            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 server.
                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 IBM Toolbox for Java information.
        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 " +
                          "IBM Toolbox for Java 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 server if the Midlet is being destroyed or exited.
        if (system_ != null)
        {
            try
            {
                system_.disconnect();
            }
            catch (Exception e)
            {
            }
        }
    }
}
```

関連情報

385 ページの『Mobile Information Device Profile (MIDP)』

例: ユーティリティー・クラス

このトピックでは、IBM Toolbox for Java 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。

AS/400ToolboxJarMaker

- 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 300KB 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、 400KB 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のプロンプトを出し、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。

コードに関するライセンス情報および特記事項

IBM は、お客様に、すべてのプログラム・コードのサンプルを使用することができる非独占的な著作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。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

こ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、いかなる保証も適用されません。商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任の保証の適用も一切ありません。

例: CommandPrompter を使用する

このプログラム例は、CommandPrompter、CommandCall、および AS400Message クラスを使用して、コマンドを求めるプロンプトの発行、コマンドの実行、そしてコマンドが稼働しない場合に戻されるメッセージの表示を行います。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////////////////////
//
// 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

このトピックでは、IBM Toolbox for Java 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 IBM i 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

統合ファイル・システム

- 例: IFSFileDialog を使用する
- 例: IFSFileSystemView を使用する
- 例: IFSTextFileDocument を使用する

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

メッセージ

- 例: VMessageQueue を使用する

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 IBM i 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

印刷

- 例: VPrinter を使用する
- 例: VPrinterOutput

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
- 例: RecordListFormPane を使用する

SpooledFileViewer

- 例: スプール・ファイル・ビューアーを作成して、システム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

SQL

- 例: SQLQueryBuilderPane を使用する
- 例: SQLResultSetTablePane を使用する

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。

コードに関するライセンス情報および特記事項

IBM は、お客様に、すべてのプログラム・コードのサンプルを使用することができる非独占的な著作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。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

こ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、いかなる保証も適用されません。商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任の保証の適用も一切ありません。

例: VUserList を使用する

このプログラム例は、システム上のユーザーのリストをリスト・ペインに示し、1 つ以上のユーザーの選択を可能にします。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////////////////
//
// 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

このプログラムは、コマンド呼び出しから戻されたメッセージの詳細ビューを示します。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// VMessageList example.  This program presents a detailed
// 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

この例は、IFS 中の特定のディレクトリーのツリー・ビューを示します。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// 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、 コードに関するライセンス情報および特記事項をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// 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

この IBM Toolbox for Java プログラム例は、サーバー・コマンドを呼び出すメニュー項目を使用する方法を例示します。また、ダイアログに戻されるすべてのメッセージを表示します。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////////////////
//
// 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

このプログラムは、サーバー・データ待ち行列と関連付けられている文書を使用する方法を示します。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
//////////////////////////////////////////////////////////////////////
//
// 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);
        }
    }
}
```

例: IFSFileDialog を使用する

この例は、IBM Toolbox for Java IFSFileDialog クラスの使用を説明しています。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
///////////////////////////////////////////////////////
//
// File Dialog example.
//
///////////////////////////////////////////////////////
```

```
import java.io.*;
import java.awt.*;
import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;

public class FileDialogExample 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)
      {
         // The first parameter is the system that contains the files.
         String system = parameters[0];

         try
         {
             // Create an AS400 object for the server that contains the files.
             // Connect to the file server on the server.  Connect now so
             // the sign-on screen is displayed now.
             AS400 as400 = new AS400(system);
             as400.connectService(AS400.FILE);

             // Create a frame to hold the dialog.
             Frame frame = new Frame();

             // Create the file dialog object.
             IFSFileDialog fileDialog = new IFSFileDialog(frame, "File Open", as400);

             // Create the list of filters the user can choose then add the filters
             // to the dialog.
             FileFilter[] filterList = {
                new FileFilter("All files (*.*)", "*.*"),
                new FileFilter("Executables (*.exe)", "*.exe"),
                new FileFilter("HTML files (*.html)", "*.html"),
                new FileFilter("Images (*.gif)", "*.gif"),
                new FileFilter("Text files (*.txt)", "*.txt")};

             fileDialog.setFileFilter(filterList, 0);

             // Set the text for the "OK" button on the dialog.
             fileDialog.setOkButtonText("Open");

             // Set the text for the "Cancel" button on the dialog.
             fileDialog.setCancelButtonText("Cancel");

             // Set the initial directory for the dialog.
             fileDialog.setDirectory("/");

             // Display the dialog and wait until the user presses OK or Cancel
             int pressed = fileDialog.showDialog();

             // If the user pressed OK, get the fully qualified path and name
             // of the file they chose.
             if (pressed == IFSFileDialog.OK)
             {
                System.out.println("User selected: " +
                                   fileDialog.getAbsolutePath());
             }

             // Else if the user pressed cancel, display a message.

             else if (pressed == IFSFileDialog.CANCEL)
```

```
            {
               System.out.println("User pressed cancel");
            }

            else
               System.out.println("User didn't press Open or Cancel");

         }
         catch(Exception e)
         {
            // If any of the above operations failed say the dialog operation
            // failed and output the exception.

            System.out.println("Dialog 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("  FileDialogExample system");
         System.out.println("");
         System.out.println("Where");
         System.out.println("");
         System.out.println("  system = IBM i");
         System.out.println("");
         System.out.println("For example:");
         System.out.println("");
         System.out.println("");
         System.out.println("  FileDialogExample mySystem");
         System.out.println("");
         System.out.println("");
      }

      System.exit(0);
   }
}
```

例: IFSTextFileDocument を使用する

このプログラムは、統合ファイル・システム内のテキスト・ファイルと関連付けられている文書を使用する方法を説明します。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////////////////
//
// IFS text file document example.  This program demonstrates how to
// use a document that is associated with a text file in the
// integrated file system.
//
// Command syntax:
//    IFSTextFileDocumentExample system path
//
////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
```

```
import java.awt.*;
import java.awt.event.*;

public class IFSTextFileDocumentExample
{



    private static IFSTextFileDocument  document;
    private static JTextPane            text;



    public static void main (String[] args)
    {
        // If a system or path was not specified, then display
        // help text and exit.
        if (args.length != 2)
        {
            System.out.println("Usage:  IFSTextFileDocumentExample system path");
            return;
        }

        try
        {
            // Create two frames.
            JFrame f = new JFrame ("IFS text file document example");

            // 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 the text file 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 and load the IFS text file document.
            document = new IFSTextFileDocument (system, args[1]);
            document.addErrorListener (errorHandler);
            document.addWorkingListener (cursorAdapter);
            document.load ();

            // Create the text pane used to present the document.
            text = new JTextPane (document);
            text.setSize (new Dimension (500, 500));

            // Set up a scroll pane to use with the text pane.
            JScrollPane scroll = new JScrollPane (text);
            scroll.setHorizontalScrollBarPolicy (JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
            scroll.setVerticalScrollBarPolicy (JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED);

            // Create a menu bar with a single menu.
            MenuBar menuBar = new MenuBar ();
            Menu menu = new Menu ("File");
            menuBar.add (menu);

            // Add menu items to load and save.
            MenuItem load = new MenuItem ("Load");
            load.addActionListener (new ActionListener ()
            {
                public void actionPerformed (ActionEvent event)
                {
                    document.load ();
```

```
            }
        });
        menu.add (load);

        MenuItem save = new MenuItem ("Save");
        save.addActionListener (new ActionListener ()
        {
            public void actionPerformed (ActionEvent event)
            {
                document.save ();
            }
        });
        menu.add (save);

        // 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 BorderLayout ());
        f.getContentPane ().add ("Center", scroll);
        f.setMenuBar (menuBar);
        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。

例: AS400JDBCDataSourcePane を使用する

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。

```
// 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 ();
```

関連情報

AS400JDBCDataSourcePane Javadoc

例: VJobList を使用してジョブのリストを表示する

このプログラムは、エクスプローラー・ペインにジョブ・リストを示します。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// 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: VMessageQueue を使用する

このプログラムは、エクスプローラー・ペインにメッセージ・キューを示します。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
///////////////////////////////////////////////////////////////////////
//
// Message queue example.  This program presents a message queue in an
// explorer pane.
//
```

```
// Command syntax:
//    VMessageQueueExample system
//
// This source is an example of IBM Toolbox for Java "VMessageQueue".
//
///////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class VMessageQueueExample
{


    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:  VMessageQueueExample system");
            return;
        }

        try
        {
            // Create an AS400 object.  The system name was passed
            // as the first command line argument.
            AS400 system = new AS400 (args[0]);

            // Force the user to sign on so that we know the user id.
            system.connectService (AS400.COMMAND);

            // Create a VMessageQueue object which represents the
            // current user's message queue.
            VMessageQueue queue = new VMessageQueue (system,
                QSYSObjectPathName.toPath ("QUSRSYS", system.getUserId (),
                "MSGQ"));

            // Create a frame.
            JFrame f = new JFrame ("Message queue 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 message queue.
            // Use load to load the information from the system.
            AS400ExplorerPane explorerPane = new AS400ExplorerPane (queue);
            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

このプログラムは、サーバー上でプログラムを呼び出すボタンを使用する方法を例示します。これは、入力および出力のパラメーターを介して、サーバー・プログラムとデータを交換します。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// 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);
        }
    }


}
```

例: VPrinter を使用する

このプログラム例は、プリンターとそのスプール・ファイルをエクスプローラー・ペインに示します。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////////////////////////
//
// VPrinter example.  This program presents a printer and its spooled
// files in an explorer pane.
//
// Command syntax:
//    VPrinterExample system
//
////////////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class VPrinterExample
{


    public static void main (String[] args)
    {

        // If the user does not supply a printer name then show printer information
        // for a printer called OS2VPRT;
        String printerName = "OS2VPRT";

        // If a system was not specified, then display help text and
        // exit.
        if (args.length == 0)
        {
            System.out.println("Usage:  VPrinterExample system printer");
            return;
        }


        // If the user specified a name, use it instead of the default.
        if (args.length > 1)
           printerName = args[1];
```

```
        try
        {
            // Create an AS400 object.  The system name was passed
            // as the first command line argument.
            AS400 system = new AS400 (args[0]);


            // Create a Printer object (from the Toolbox access package)
            // which represents the printer, then create a VPrinter
            // object to graphically show the spooled files on the printer.
            Printer   printer = new Printer(system, printerName);
            VPrinter vprinter = new VPrinter(printer);

            // Create a frame to hold our window.
            JFrame f = new JFrame ("VPrinter 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 printer and its spooled
            // files.  Use load to load the information from the system.
            AS400ExplorerPane explorerPane = new AS400ExplorerPane (vprinter);
            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: VPrinters を使用する

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
///////////////////////////////////////////////////////////////////////
//
// 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);
        }
    }


}
```

VPrinterOutput の例

このプログラム例は、サーバー上のスプール・ファイルのリストを示します。すべてのスプール・ファイルを表示できますが、特定のユーザーのスプール・ファイルのみを表示することもできます。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
////////////////////////////////////////////////////////////////////////////
//
// VPrinterOutput example.  This program presents a list of spooled
// files on the server.  All spooled files, or spooled files for
// a specific user can be displayed.
//
// Command syntax:
//    VPrinterOutputExample system <user>
//
//    (User is optional, if not specified all spooled files on the system
//    will be displayed.  Caution - listing all spooled files on the system
//    and take a long time)
//
////////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class VPrinterOutputExample
{


    public static void main (String[] args)
    {

        // If a system was not specified, display help text and exit.
        if (args.length == 0)
        {
                        System.out.println("Usage:  VPrinterOutputExample system <user>");
            return;
        }

        try
        {
            // Create an AS400 object.  The system name was passed
            // as the first command line argument.
            AS400 system = new AS400 (args[0]);
            system.connectService(AS400.PRINT);

            // Create the VPrinterOutput object.
            VPrinterOutput printerOutput = new VPrinterOutput(system);

            // If a user was specified as a command line parameter, tell
            // the printerObject to get spooled files only for that user.
                        if (args.length > 1)
               printerOutput.setUserFilter(args[1]);


            // Create a frame to hold our window.
            JFrame f = new JFrame ("VPrinterOutput Example");

            // Create an error dialog adapter.  This will display
            // any errors to the user.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);


            // Create an details pane to present the list of spooled files.
```

```
            // Use load to load the information from the system.
            AS400DetailsPane detailsPane = new AS400DetailsPane (printerOutput);
            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: SQLQueryBuilderPane を使用する

このプログラムは、ユーザーが SQL 照会を作成するための照会ビルダーを示します。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// SQLQueryBuilderPane example.  This program presents a query builder
// that allows the user to build a SQL query.
//
// Command syntax:
//    SQLQueryBuilderPaneExample system
//
// This source is an example of IBM Toolbox for Java "SQLQueryBuilderPane",
// and "SQLResultSetFormPane".
//
///////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.sql.*;

public class SQLQueryBuilderPaneExample
{


    // This connection is shared by all components.
    private SQLConnection connection;



    // This error handler is shared by all components.
    private ErrorDialogAdapter errorHandler;
```

```
// The query builder pane.
private SQLQueryBuilderPane queryBuilderPane;



// This is the main java calls.  Here we create an instance of our
// class and call our own Main() method.  We do this to avoid
// problems with static.  Java has some restrictions with static
// methods using non-static data, especially when it comes to
// inner classes.  The code is cleaner if we keep the amount of
// static data and methods to a minimum.
public static void main (String[] args)
{
   SQLQueryBuilderPaneExample me = new SQLQueryBuilderPaneExample();
   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:  SQLQueryBuilderPaneExample system");
        return;
    }

    try
    {
        // Register the IBM Toolbox for Java JDBC driver.
        DriverManager.registerDriver (new AS400JDBCDriver ());

        // Create an SQLConnection object.  The system name was passed
        // as the first command line argument.
        connection = new SQLConnection ("jdbc:as400://" + args[0]);

        // Create a frame.
        JFrame f = new JFrame ("SQLQueryBuilderPane example");

        // Create an error dialog adapter.  This will display
        // any errors to the user.
        errorHandler = new ErrorDialogAdapter (f);

        // Create a SQL query builder pane to present the query
        // builder.  Load the data that is needed for the query
        // builder from the system.
        queryBuilderPane = new SQLQueryBuilderPane (connection);
        queryBuilderPane.addErrorListener (errorHandler);
        queryBuilderPane.load ();

        // Create a button which will display the results of
        // the generated query in a form pane in another frame.
        JButton resultSetButton = new JButton ("Show result set");
        resultSetButton.addActionListener (new ActionListener ()
        {
            public void actionPerformed (ActionEvent event)
            {
                showFormPane (queryBuilderPane.getQuery ());
            }
        });

        // When the frame closes, exit.
        f.addWindowListener (new WindowAdapter ()
        {
            public void windowClosing (WindowEvent event)
```

```
            {
                System.exit (0);
            }
        });

        // Layout the frame with the query builder pane.
        f.getContentPane ().setLayout (new BorderLayout ());
        f.getContentPane ().add ("Center", queryBuilderPane);
        f.getContentPane ().add ("South", resultSetButton);
        f.pack ();
        f.show ();
    }
    catch (Exception e)
    {
       System.out.println ("Error: " + e.getMessage ());
       System.exit (0);
    }
}



private void showFormPane (String query)
{
    // Create a new frame for the results of the query.
    JFrame f = new JFrame (query);

    // Create a SQL result set form pane to present the results
    // of the query.  Load the results from the system.
    SQLResultSetFormPane formPane = new SQLResultSetFormPane (connection, query);
    formPane.addErrorListener (errorHandler);
    formPane.load ();

    // Layout the frame with the form pane.
    f.getContentPane ().setLayout (new BorderLayout ());
    f.getContentPane ().add ("Center", formPane);
    f.pack ();
    f.show ();
}


}
```

例: SQLResultSetTablePane を使用する

このプログラムは、テーブルの内容を「テーブル・パネル」に示します。ユーザーが任意の SQL ステートメントに入力するときに使用できる SQLStatementDocument があります。それ以外に、ユーザーがテーブルのすべての行を削除するのに使用できるボタンもあります。

注: 法律上の重要な情報に関しては、 コードに関するライセンス情報および特記事項をお読みください。

```
///////////////////////////////////////////////////////////////////////////
//
// SQLResultSetTablePane example.  This program presents the contents of
// a table in a table pane.  There is a SQLStatementDocument that allows
// the user to type in any SQL statement.  In addition, there is a button
// that allows the user to delete all rows of the table.
//
// Command syntax:
//    SQLResultSetTablePaneExample  system  table
//
// This source is an example of IBM Toolbox for Java "SQLQueryBuilderPane",
// "SQLResultSetFormPane", and "SQLStatementButton".
//
///////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
```

```
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.sql.*;

public class SQLResultSetTablePaneExample
{



    private static SQLStatementDocument     document;
    private static SQLResultSetTablePane    tablePane;



    public static void main (String[] args)
    {
        // If a system was not specified, then display
        // help text and exit.
        if (args.length != 2)
        {
            System.out.println("Usage:  SQLResultSetTablePaneExample  system  table");
            return;
        }

        try
        {
            // Register the IBM Toolbox for Java JDBC driver.
            DriverManager.registerDriver (new AS400JDBCDriver ());

            // Create an SQLConnection object.  The system name was passed
            // as the first command line argument.  This connection is
            // shared by all components.
            SQLConnection connection = new SQLConnection ("jdbc:as400://" + args[0]);

            // Create a frame.
            JFrame f = new JFrame ("SQLResultSetTablePane example");

            // Create an error dialog adapter.  This will display
            // any errors to the user.  This error handler is shared
            // by all components.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

            // Create a SQL statement document which allows the
            // user to enter a query.
            document = new SQLStatementDocument (connection, "");
            document.addErrorListener (errorHandler);

            // Create a text field for presenting the document.
            JTextField textField = new JTextField (document,
                "Enter a SQL statement here.", 50);

            // Create a button that deletes all rows of the table.
            SQLStatementButton deleteAllButton = new SQLStatementButton ("Delete all rows");
            deleteAllButton.setConnection (connection);
            deleteAllButton.setSQLStatement ("DELETE FROM " + args[1]);
            deleteAllButton.addErrorListener (errorHandler);

            // Create a SQL result set table pane to present the results
            // of a query.  Load the contents immediately.
            tablePane = new SQLResultSetTablePane (connection, "SELECT * FROM " + args[1]);
            tablePane.addErrorListener (errorHandler);
            tablePane.load ();

            // When enter is pressed in the text field,
            // execute the SQL statement and update the table pane.
```

```
            textField.addKeyListener (new KeyAdapter ()
            {
                public void keyPressed (KeyEvent event)
                {
                    if (event.getKeyCode () == KeyEvent.VK_ENTER)
                    {
                        // If the SQL statement is a SELECT, then
                        // let the table pane execute it, otherwise,
                        // let the document execute it.
                        String sql = document.getSQLStatement ();
                        if (sql.toUpperCase ().startsWith ("SELECT"))
                        {
                            try
                            {
                                tablePane.setQuery (sql);
                            }
                            catch (Exception e)
                            {
                                // Ignore.
                            }
                            tablePane.load ();
                        }
                        else
                            document.execute ();
                    }
                }
            });

            // When all rows are deleted using the button, then
            // update the table pane.
            deleteAllButton.addActionCompletedListener (new ActionCompletedListener ()
            {
                public void actionCompleted (ActionCompletedEvent event)
                {
                    tablePane.load ();
                }
            });

            // When the frame closes, exit.
            f.addWindowListener (new WindowAdapter ()
            {
                public void windowClosing (WindowEvent event)
                {
                    System.exit (0);
                }
            });

            // Layout the frame with the query builder pane.
            f.getContentPane ().setLayout (new BorderLayout ());
            f.getContentPane ().add ("North", textField);
            f.getContentPane ().add ("Center", tablePane);
            f.getContentPane ().add ("South", deleteAllButton);
            f.pack ();
            f.show ();
        }
        catch (Exception e)
        {
           System.out.println ("Error: " + e.getMessage ());
           System.exit (0);
        }
    }


}
```

例: XPCML

このセクションでは、IBM Toolbox for Java XPCML コンポーネント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。

- 804 ページの『例: 既存の XPCML 文書を圧縮する』
- 804 ページの『例: 既存の XPCML 文書を圧縮する (Java コードを含む)』
- 807 ページの『例: 圧縮された XPCML を使用して ProgramCallDocument オブジェクトを作成する』
- 807 ページの『例: プログラム呼び出しの結果を圧縮された XPCML として取得する』
- 『例: プログラム呼び出しの結果を XPCML として取り出す』
- 801 ページの『例: パラメーター値を XPCML として渡す』
- 802 ページの『例: パラメーター値の配列を XPCML として渡す』
- 479 ページの『例: PCML 文書を XPCML 文書に変換する』

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: プログラム呼び出しの結果を XPCML として取り出す

以下の例は、XPCML ProgramCallDocument を構成し、IBM i プログラムを呼び出し、プログラム呼び出しの結果を XPCML として取り出す方法を示しています。

この例は、以下のコンポーネントを前提にしています。

- XPCML 文書 qgyolaus.xpcml。入力値を使用してプログラムおよびパラメーターの仕様を定義します。
- ProgramCallDocument オブジェクトを構成し、XPCML ファイルを使用してから、プログラム QGYOLAUS を呼び出す Java コード。
- プログラム呼び出しの結果。 Java コードにより XPCML として生成され、ファイル XPCMLOut.xpcml に保管されます。

元の XPCML および生成される XPCML 中で配列データが指定される方法に注意してください。出力パラメーターのエレメント qgyolaus.receiver は、カウントを listInfo.rcdsReturned に設定する属性のある XPCML arrayOfStructParm です。以下の例のコードには、QGYOLAUS 出力の一部のみ含まれています。例にすべての出力を含めると、コードの <arrayOfStructParm> XPCML タグの下に 89 人のユーザーがリストされることになります。

構造体の配列の場合、XPCML は <struct_i> XPCML タグを使用して、個々の structParm エレメントを区切ります。個々の <struct_i> タグは、このタグで囲まれたデータがタイプ autu0150 struct のエレメントの 1 つであることを示します。 <struct_i> タグの索引属性は、構造体の配列のエレメントを指定します。

arrayOfStringParm および arrayOfIntParm などの単純なタイプの配列の場合、 <i> XPCML タグは配列のエレメントをリストします。

XPCML 文書 qgyolaus.xpcml

```
<xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="xpcml.xsd" version="4.0">

<!-- XPCML source for calling "Open List of Authorized Users" -->
<!-- (QGYOLAUS) API -->

  <!-- Format AUTU0150 - Other formats are available -->
  <struct name="autu0150">
    <stringParm name="name" length="10"/>
    <stringParm name="userOrGroup" length="1"/>
    <stringParm name="groupMembers" length="1"/>
    <stringParm name="description" length="50"/>
  </struct>

  <!-- List information structure (common for "Open List" type APIs) -->
  <struct name="listInfo">
    <intParm name="totalRcds"/>
    <intParm name="rcdsReturned">0</rcdsReturned>
    <hexBinaryParm name="rqsHandle" totalBytes="4"/>
    <intParm name="rcdLength"/>
    <stringParm name="infoComplete" length="1"/>
    <stringParm name="dateCreated" length="7"/>
    <stringParm name="timeCreated" length="6"/>
    <stringParm name="listStatus" length="1"/>
    <hexBinaryParm totalBytes="1"/>
    <unsignedIntParm name="lengthOfInfo"/>
    <intParm name="firstRecord"/>
    <hexBinaryParm totalBytes="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">
   <parameterList>
     // Output values --- array of the autu0150 struct
     <arrayOfStructParm name="receiver" count="listInfo.rcdsReturned"
        passDirection="out" outputSize="receiverLength" struct="autu0150"/>
     // Input values
     <intParm name="receiverLength" passDirection="in">16384</intParm>
     <structParm name="listInfo" passDirection="out" struct="listInfo"/>
     // Input values
     <intParm name="rcdsToReturn" passDirection="in">264</intParm>
     <stringParm name="format" passDirection="in" length="10">
        AUTU0150</stringParm>
     <stringParm name="selection" passDirection="in" length="10">
        *USER</stringParm>
     <stringParm name="member" passDirection="in" length="10">
        *NONE</stringParm>
     <intParm name="errorCode" passDirection="in">0</intParm>
   </parameterList>
  </program>
```

ProgramCallDocument オブジェクトを構成し XPCML を使用してプログラム QGYOLAUS を呼び出す Java コード

```
system = new AS400();
// Create a ProgramCallDocument into which to parse the file.
ProgramCallDocument xpcmlDoc =
   new ProgramCallDocument(system, "QGYOLAUS.xpcml");

// Call QGYOLAUS
boolean rc = xpcmlDoc.callProgram("QGYOLAUS");

// Obtain program call results as XPCML and store them
// in file XPCMLOut.xpcml
if (rc)     // Program was successful
   xpcmlDoc.generateXPCML("QGYOLAUS","XPCMLOut.xpcml");
```

XPCML として生成されてファイル XPCMLOut.xpcml に保管されるプログラム呼び出しの結果

```
<?xml version="1.0"?>
<xpcml version="4.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation='xpcml.xsd' >

  <program name="QGYOLAUS" path="/QSYS.lib/QGY.lib/QGYOLAUS.pgm"
     parseOrder="listInfo receiver">
  <parameterList>
      <arrayOfStructParm name="receiver" passDirection="out"
          count="listInfo.rcdsReturned" outputSize="receiverLength"
          struct="autu0150">
        <struct_i index="0">
          <stringParm name="name" length="10">JANEDOW</stringParm>
          <stringParm name="userOrGroup" length="1">0</stringParm>
          <stringParm name="groupMembers" length="1">0</stringParm>
          <stringParm name="description" length="50">
             Jane Doe</stringParm>
        </struct_i>
        <struct_i index="1">
          <stringParm name="name" length="10">BOBS</stringParm>
          <stringParm name="userOrGroup" length="1">0</stringParm>
          <stringParm name="groupMembers" length="1">0</stringParm>
          <stringParm name="description" length="50">
             Bob Smith</stringParm>
        </struct_i>

        <!-- More records here depending on how many users output. -->
        <!-- In this case 89 user records are listed here.    -->

      </arrayOfStructParm>     <!-- End of user array -->
      <intParm name="receiverLength" passDirection="in">
         16384</intParm>
      <structParm name="listInfo" passDirection="out"
         struct="listInfo">
          <intParm name="totalRcds">89</intParm>
          <intParm name="rcdsReturned">89</intParm>
          <hexBinaryParm name="rqsHandle" totalBytes="4">
             00000001==</hexBinaryParm>
          <intParm name="rcdLength">62</intParm>
          <stringParm name="infoComplete" length="1">C</stringParm>
          <stringParm name="dateCreated" length="7">
             1030321</stringParm>
          <stringParm name="timeCreated" length="6">
             120927</stringParm>
          <stringParm name="listStatus" length="1">2</stringParm>
          <hexBinaryParm totalBytes="1"></hexBinaryParm>
          <unsignedIntParm name="lengthOfInfo">
             5518</unsignedIntParm>
          <intParm name="firstRecord">1</intParm>
```

```
      </structParm>
      <intParm name="rcdsToReturn" passDirection="in">264</intParm>
      <stringParm name="format" passDirection="in" length="10">
         AUTU0150</stringParm>
      <stringParm name="selection" passDirection="in" length="10">
         *USER</stringParm>
      <stringParm name="member" passDirection="in" length="10">
         *NONE</stringParm>
      <intParm name="errorCode" passDirection="in">0</intParm>
  </parameterList>
  </program>
</xpcml>
```

例: パラメーター値を XPCML として渡す

プログラム・パラメーター値を XPCML ソース・ファイル中に設定できます。 XPCML が読み取られて解析される際に、値が XPCML として渡されたパラメーターごとに ProgramCallDocument setValue メソッドが自動的に呼び出されます。したがって、ユーザーは Java コードを作成して複雑な構造や配列の値を設定する必要がなくなります。

以下の例では、XPCML は prog1 および prog2 の 2 種類のプログラムを呼び出します。両方のプログラムとも入力パラメーター s1Ref を使用しています。最初の例では、プログラム呼び出しごとに違う s1Ref 値を設定します。 2 つ目の例では、プログラム呼び出しごとに同じ s1Ref 値を設定し、入力パラメーターの定数データ値を設定する便利な方法を示します。

例: さまざまな入力パラメーター値を渡す

以下の例では、XML パーサーが文書を読み取って解析した後に、エレメント prog1.s1Ref.s2Ref.s2p1[0] の値は prog1Val_1 になり、エレメント prog1.s1Ref.s2Ref.s2p1[1] の値は prog1Val_2 になります。

```
    <xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="xpcml.xsd" version="4.0">

<struct name="s1" >
   <stringParm name="s1p1"/>
   <structParm name="s2Ref" struct="s2"/>
</struct>

<struct name="s2">
  <stringParm name="s2p1" length="10"/>
  <arrayOfStringParm name="parm1" count="2"/>
</struct>

<program name="prog1" path="/QSYS.LIB/W95LIB.LIB/PROG1.PGM">
 <parameterList>
   <structParm name="s1Ref" struct="s1" passDirection="in" >
     <stringParm name="s1p1">prog1Val</stringParm>
     <structParm name="s2Ref" struct="s2">
         <stringParm name="s2p1" length="10">prog1Val</stringParm>
         <arrayOfStringParm name="parm1" count="2">
            <i>prog1Val_1</i>
            <i>prog1Val_2</i>
         </arrayOfStringParm>
     </structParm>
   </structParm>
 </parameterList>
</program>

<program name="prog2" path="/QSYS.LIB/W95LIB.LIB/PROG1.PGM">
 <parameterList>
   <structParm name="s1Ref" struct="s1" passDirection="in" >
     <stringParm name="s1p1">prog2Val</stringParm>
     <structParm name="s2Ref" struct="s2">
```

```
          <stringParm name="s2p1" length="10">prog2Val</stringParm>
          <arrayOfStringParm name="parm1" count="2">
             <i>prog2Val_1</i>
             <i>prog2Val_2</i>
          </arrayOfStringParm>
       </structParm>
     </structParm>
   </parameterList>
  </program>
 </xpcml>
```

例: 入力パラメーターの定数値を渡す

以下の例では、XML パーサーが文書を読み取って解析した後に、エレメント prog1.s1Ref.s2Ref.s2p1[0] の値は constantVal_1 になり、エレメント prog1.s1Ref.s2Ref.s2p1[1] の値は constantVal_2 になります。

```
<xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="xpcml.xsd" version="4.0">

 <struct name="s1" >
    <stringParm name="s1p1">constantVal</stringParm>
    <structParm name="s2Ref" struct="s2"/>
 </struct>

 <struct name="s2">
   <stringParm name="s2p1" length="10">constantVal</stringParm>
   <arrayOfStringParm name="parm1" count="2">
      <i>constantVal_1</i>
      <i>constantVal_2</i>
   </arrayOfStringParm>
 </struct>

 <program name="prog1" path="/QSYS.LIB/W95LIB.LIB/PROG1.PGM">
  <parameterList>
    <structParm name="s1Ref" struct="s1" passDirection="in" />
  </parameterList>
 </program>

 <program name="prog2" path="/QSYS.LIB/W95LIB.LIB/PROG1.PGM">
  <parameterList>
    <structParm name="s1Ref" struct="s1" passDirection="in" />
  </parameterList>
 </program>
</xpcml>
```

例: パラメーター値の配列を XPCML として渡す

以下の例は、structParm 配列データおよび構造体の配列を使用して、パラメーター値の配列を渡す方法を示しています。

XPCML を使用して配列データを渡す際には、カウント属性を使用しなければなりません。

- 配列エレメントに対するカウント属性を指定する。
- カウント属性を、文書の解析時に配列に含まれるエレメントの数に設定する。

```
<?xml version="1.0" encoding="UTF-8"?>
<xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="xpcml.xsd" version="4.0">

 <struct name="s1" >
    <stringParm name="s1p1"/>
    <struct name="s1Array">
      <stringParm name="s1Ap1"/>
    </struct>
 </struct>
```

```
 <struct name="s2">
    <stringParm name="s2p1"/>
 </struct>

 <program name="prog1" path="/QSYS.LIB/W95LIB.LIB/PROG1.PGM">
  <parameterList>
    <structParm name="s1Ref" struct="s1" passDirection="in" >
        <stringParm name="s1p1">Value 1</stringParm>
        <arrayOfStruct name="s1Array" count="2">
           <struct_i>
               <stringParm name="s1Ap1">Value 1</stringParm>
           </struct_i>
           <struct_i>
               <stringParm name="s1Ap1">Value 2</stringParm>
           </struct_i>
         </arrayOfStruct>
    </structParm>
    <arrayOfStructParm name="s2Ref" struct="s2" count="2" passDirection="in" >
      <struct_i>
          <stringParm name="s2p1">Value 1</stringParm>
      </struct_i>
      <struct_i>
          <stringParm name="s2p1">Value 2</stringParm>
      </struct_i>
    </arrayOfStructParm>
  </parameterList>
 </program>
</xpcml>
```

例えば、以下の XPCML は 3 つの intParm の配列を指定し、 1 つ目のエレメントを 12、2 つ目を 100、3 つ目を 4 に設定します。

```
<?xml version="1.0"?>
<xpcml version="4.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation='xpcml.xsd' >
  <program name="prog1" path="/QSYS.lib/MYLIG.lib/PROG1.pgm">
  <parameterList>
    <arrayOfIntParm name="intArray" count="3">
      <i>12</i>
      <i>100</i>
      <i>4</i>
    </arrayOfIntParm>
  </parameterList>
  </program>
</xpcml>
```

<i> および <struct_i> タグの索引属性を使用して配列値を設定する

<i> および <struct_i> タグの索引属性を使用して、配列値の設定に役立てることができます。以下の例で、XPCML は配列の 1 つ目のエレメントを 4、 2 つ目を 100、3 つ目を 12 に設定します。

```
<?xml version="1.0"?>
<xpcml version="4.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation='xpcml.xsd' >

  <program name="prog1" path="/QSYS.lib/MYLIG.lib/PROG1.pgm">
  <parameterList>
    <arrayOfIntParm name="intArray" count="3">
      <i index="2">12</i>
      <i index="1">100</i>
      <i index="0">4</i>
    </arrayOfIntParm>
  </parameterList>
  </program>
</xpcml>
```

例: 既存の XPCML 文書を圧縮する

以下の例は、既存の XPCML 文書を圧縮する方法を示しています。この例には、元の XPCML ソース、圧縮結果の XPCML、および拡張されたスキーマが含まれています。

元の XPCML ソース

```
<?xml version="1.0" encoding="UTF-8"?>
 <xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="xpcml.xsd" version="4.0">
  <program name="prog1" path="/QSYS.LIB/W95LIB.LIB/PROG1.PGM">
    <parameterList>
       <stringParm name="parm1" passDirection="in" passMode="value"
             minvrm="V5R2M0" ccsid="37" length="10">Value 1</stringParm>
    </parameterList>
  </program>
 </xpcml>
```

圧縮された XPCML ソース

```
<?xml version="1.0" encoding="UTF-8"?>
<xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="myXSD.xsd" version="4.0">
 <program name="prog1" path="/QSYS.LIB/W95LIB.LIB/PROG1.PGM">
    <parameterList>
       <parm1_>Value 1</parm1_>
    </parameterList>
 </program>
</xpcml>
```

生成されたスキーマ

```
<!-- parm1's XSD definition -->
<xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'>
 <!-- Link back to XPCML.xsd -->
 <xs:include schemaLocation='xpcml.xsd'/>
  <xs:element name="parm1_" substitutionGroup="stringParmGroup" >
   <xs:complexType>
     <xs:simpleContent>
     <xs:restriction base="stringParmType">
        <!-- Attributes defined for parm1 -->
        <xs:attribute name="name" type="string50" fixed="parm1" />
        <xs:attribute name="length" type="xs:string" fixed="10" />
        <xs:attribute name="passMode" type="xs:string" fixed="value" />
        <xs:attribute name="ccsid" type="xs:string" fixed="37" />
        <xs:attribute name="minvrm" type="xs:string" fixed="V5R2M0" />
     </xs:restriction>
     </xs:simpleContent>
   </xs:complexType>
  </xs:element>
</schema>
```

例: 既存の XPCML 文書を圧縮する (Java コードを含む)

以下の例は、既存の XPCML 文書を圧縮する方法を示しています。この例には、元の XPCML ソース、圧縮結果の XPCML、condenseXPCML() という Java コード、および拡張されたスキーマ中に新しく生成された数種のタイプ定義が含まれています。

元の XPCML ソース

```
<!-- Fully specified XPCML source -->
<xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="xpcml.xsd" version="4.0">

<struct name="qualifiedJobName">
   <stringParm name="jobName" length="10">*</stringParm>
```

```
   <stringParm name="userName" length="10"/>
   <stringParm name="jobNumber" length="6"/>
</struct>

<struct name="jobi0100">
   <intParm name="numberOfBytesReturned"/>
   <intParm name="numberOfBytesAvailable"/>
   <structParm name="qualifiedJobName" struct="qualifiedJobName"/>
   <hexBinaryParm name="internalJobIdentifier" totalBytes="16"/>
   <stringParm name="jobStatus" length="10"/>
   <stringParm name="jobType" length="1"/>
   <stringParm name="jobSubtype" length="1"/>
   <stringParm length="2"/>
   <intParm name="runPriority"/>
   <intParm name="timeSlice"/>
   <intParm name="defaultWait"/>
   <stringParm name="purge" length="10"/>
</struct>

<program name="qusrjobi_jobi0100" path="/QSYS.LIB/QUSRJOBI.PGM">
   <parameterList>
      <structParm name="receiverVariable" passDirection="out"
         outputSize="lengthOfReceiverVariable" struct="jobi0100"/>
      <intParm name="lengthOfReceiverVariable" passDirection="in">86</intParm>
      <stringParm name="formatName" passDirection="in" length="8">JOBI0100</stringParm>
      <structParm name="qualifiedJobName" passDirection="in" struct="qualifiedJobName"/>
      <hexBinaryParm name="internalJobIdentifier"
         passDirection="in" totalBytes="16"> </hexBinaryParm>
      <intParm name="errorCode" passDirection="in">0</intParm>
   </parameterList>
  </program>
</xpcml>
```

元の XPCML ソースを圧縮する Java コード

```
try {
   FileInputStream fullStream = new FileInputStream("myXPCML.xpcml");
   FileOutputStream condensedStream = new FileOutputStream("myCondensedXPCML.xpcml");
   FileOutputStream xsdStream = new FileOutputStream("myXSD.xsd");
   ProgramCallDocument.condenseXPCML(fullStream, xsdStream, condensedStream, "myXSD.xsd");

}
catch (Exception e) {
  System.out.println("error:  - "+e.getMessage());
  e.printStackTrace();
}
```

圧縮された XPCML ソース: myCondensedXPCML.xpcml

```
<xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="myXSD.xsd" version="4.0">

<struct name="qualifiedJobName">
   <jobName_>*</jobName_>
   <userName_/>
   <jobNumber_/>
</struct>

<struct name="jobi0100">
   <numberOfBytesReturned_/>
   <numberOfBytesAvailable_/>
   <structParm name="qualifiedJobName" struct="qualifiedJobName"/>
   <internalJobIdentifier_/>
   <jobStatus_/>
   <jobType_/>
   <jobSubtype_/>
   <stringParm length="2"/>
```

```
   <runPriority_/>
   <timeSlice_/>
   <defaultWait_/>
   <purge_/>
</struct>

<program name="qusrjobi_jobi0100" path="/QSYS.LIB/QUSRJOBI.PGM">
   <parameterList>
      <structParm name="receiverVariable" passDirection="out"
         outputSize="lengthOfReceiverVariable" struct="jobi0100"/>
      <lengthOfReceiverVariable_>86</lengthOfReceiverVariable_>
      <formatName_>JOBI0100</formatName_>
      <structParm name="qualifiedJobName" passDirection="in" struct="qualifiedJobName"/>
      <internalJobIdentifier_> </internalJobIdentifier_>
      <errorCode_>0</errorCode_>
   </parameterList>
  </program>
</xpcml>
```

生成されたスキーマ: myXSD.xsd 中の数種のタイプ定義

```
<xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'>
<xs:include schemaLocation='xpcml.xsd'/>

<xs:element name="jobName_" substitutionGroup="stringParmGroup" >
   <xs:complexType>
      <xs:simpleContent>
      <xs:restriction base="stringParmType">
         <xs:attribute name="name" type="string50" fixed="jobName" />
         <xs:attribute name="length" type="xs:string" fixed="10" />
      </xs:restriction>
      </xs:simpleContent>
   </xs:complexType>
</xs:element>

<xs:element name="userName_" substitutionGroup="stringParmGroup" >
   <xs:complexType>
      <xs:simpleContent>
      <xs:restriction base="stringParmType">
         <xs:attribute name="name" type="string50" fixed="userName" />
         <xs:attribute name="length" type="xs:string" fixed="10" />
      </xs:restriction>
      </xs:simpleContent>
   </xs:complexType>
</xs:element>

<xs:element name="jobNumber_" substitutionGroup="stringParmGroup" >
   <xs:complexType>
      <xs:simpleContent>
      <xs:restriction base="stringParmType">
         <xs:attribute name="name" type="string50" fixed="jobNumber" />
         <xs:attribute name="length" type="xs:string" fixed="6" />
      </xs:restriction>
      </xs:simpleContent>
   </xs:complexType>
</xs:element>

<xs:element name="lengthOfReceiverVariable_" substitutionGroup="intParmGroup" >
   <xs:complexType>
      <xs:simpleContent>
      <xs:restriction base="intParmType">
         <xs:attribute name="name" type="string50" fixed="lengthOfReceiverVariable" />
         <xs:attribute name="passDirection" type="passDirectionType" fixed="in" />
      </xs:restriction>
      </xs:simpleContent>
   </xs:complexType>
</xs:element>
```

```
<xs:element name="formatName_" substitutionGroup="stringParmGroup" >
   <xs:complexType>
      <xs:simpleContent>
      <xs:restriction base="stringParmType">
         <xs:attribute name="name" type="string50" fixed="formatName" />
         <xs:attribute name="length" type="xs:string" fixed="8" />
         <xs:attribute name="passDirection" type="passDirectionType" fixed="in" />
      </xs:restriction>
      </xs:simpleContent>
   </xs:complexType>
</xs:element>

<!-- More type definitions for each newly defined type follow here -->
</xs:schema>
```

例: 圧縮された XPCML を使用して ProgramCallDocument オブジェクトを作成する

ProgramCallDocument コンストラクターの中には、 condensedXPCML ソース・ファイルおよび対応するスキーマ (.xsd ファイル) を受け入れるものもあります。このコンストラクターを使用すると、圧縮された XPCML を使用して ProgramCallDocument オブジェクトを作成できます。

前述のコンストラクターには、以下のパラメーターを指定する必要があります。

- 圧縮された XPCML ファイルを指定したストリング
- condenseXPCML() を実行して作成したタイプ定義を含む InputStream

これらのコンストラクターを使用して、圧縮された XPCML ファイルをロードして解析できます。さらに、このプロセスでは解析エラーがログに記録されます。解析の完了後に、コンストラクターは ProgramCallDocument オブジェクトを作成します。

以下の Java コードの例は、圧縮された XPCML を使用して ProgramCallDocument オブジェクトを作成します。例のコードは、以下の点を前提にしています。

- 圧縮された XPCML ファイルの名前は myCondensedXPCML.xpcml
- 拡張されたスキーマの名前は myXSD.xsd

さらにこのコードは、 ProgramCallDocument オブジェクトを使用してプログラム qusrjobi_jobi0100 を実行します。

```
AS400 system = new AS400();
// Create a ProgramCallDocument and parse the file.
ProgramCallDocument xpcmlDoc =
  new ProgramCallDocument(system,
                          "myCondensedXPCML.xpcml",
                          new FileInputStream("myXSD.xsd"));
boolean rc = xpcmlDoc.callProgram("qusrjobi_jobi0100");
```

注: プログラムの呼び出しに (ProgramCallDocument オブジェクトの作成後に) 使用する XPCML コードは、 PCML の場合に使用するコードと同じです。

例: プログラム呼び出しの結果を圧縮された XPCML として取得する

プログラム呼び出しの結果を圧縮された XPCML として取得する場合にも、圧縮されていない XPCML として取得する場合にも、ProgramCallDocument.generateXPCML() を呼び出すことによって、同一のプロセスを使用します。

setXsdName() を使用して、拡張されたスキーマの名前を指定してください。 generateXPCML() はこのスキーマを使用して、圧縮された XPCML 形式で <xpcml> タグの noNamespaceSchemaLocation 属性を生成します。

setXsdName() を使用することは、プログラム呼び出しの結果 (圧縮された XPCML 形式) を別の ProgramCallDocument オブジェクトのソースとして使用する場合に重要です。パーサーが解析時に使用するスキーマ・ファイルを認識できるように、拡張されたスキーマの名前を指定しなければなりません。

例えば、以下のコードはプログラム呼び出しから結果を取得し、圧縮された XPCML を生成します。

```
AS400 system = new AS400();

// Create a ProgramCallDocument and parse the file.
ProgramCallDocument xpcmlDoc =
   new ProgramCallDocument(system, "myCondensedXPCML.xpcml", new FileInputStream("myXSD.xsd"));

boolean rc = xpcmlDoc.callProgram("qusrjobi_jobi0100");

if (rc)     // Program was successful
{
  xpcmlDoc.setXsdName("myXSD.xsd");
  xpcmlDoc.generateXPCML("qusrjobi_jobi0100","XPCMLOut.xpcml");
}
```

以下のコードは、プログラム呼び出しの結果を圧縮された XPCML として取得する例を示しています。

```
<xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="myXSD.xsd" version="4.0">

<program name="qusrjobi_jobi0100" path="/QSYS.LIB/QUSRJOBI.PGM">
   <parameterList>
   <structParm name="receiverVariable" passDirection="out"
       outputSize="lengthOfReceiverVariable" struct="jobi0100"/>
            <numberOfBytesReturned_>100</numberOfBytesReturned_>
            <numberOfBytesAvailable_>100</numberOfBytesAvailable_>
            <structParm name="qualifiedJobName"
                struct="qualifiedJobName">
                <jobName_>*</jobName_>
                <userName_/>
                <jobNumber_/>
            </structParm>
            <internalJobIdentifier_/>
            <jobStatus_>ACTIVE</jobStatus_>
            <jobType_>PJ</jobType>
            <jobSubtype_/>
            <stringParm length="2"/>
            <runPriority_>5</runPriority>
            <timeSlice_/>
            <defaultWait_>10</defaultWait_>
            <purge_/>
        </structParm>
        <lengthOfReceiverVariable_>86</lengthOfReceiverVariable_>
        <formatName_>JOBI0100</formatName_>
        <structParm name="qualifiedJobName" passDirection="in" struct="qualifiedJobName"/>
        <internalJobIdentifier_> </internalJobIdentifier_>
        <errorCode_>0</errorCode_>
</parameterList>
  </program>
</xpcml>
```

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 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  : IBM Toolbox for Java を使用する Java プログラマーおよび IBM 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とるための有効な手段が提供されます。

IBM Toolbox for Java 2 Micro Edition のリソース

ToolboxME 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 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)

- ワイヤレス、モバイル、および組み込みデバイス用の Java テクノロジー (英語)  : 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:
  - WebSphere Everyplace Micro Environment 
  - Sun Java Wireless Toolkit for CLDC (英語) 

Java

Java 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。

- Java デベロッパーのためのソース (英語)  : 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 Naming and Directory Interface

- Java Naming and Directory Interface (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。
- System i LDAP : IBM i 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 SE セキュリティー (英語) : 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 Java 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。

- 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。

- 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、新興の XML 規格および XML ツールについての情報が提供されています。
- 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。
- 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 i : IBM HTTP Server for i 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。
- IBM i Access (英語) : ダウンロード、FAQ、付加的なサイトへのリンクを含め、IBM i Access についての情報が提供されています。
- IBM WebSphere Host On-Demand : S/390®、 IBM i、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。

コードに関するライセンス情報および特記事項

IBM は、お客様に、すべてのプログラム・コードのサンプルを使用することができる非独占的な著作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。

強行法規で除外を禁止されている場合を除き、IBM、そのプログラム開発者、および供給者は「プログラム」および「プログラム」に対する技術的サポートがある場合にはその技術的サポートについて、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。

いかなる場合においても、IBM および IBM のサプライヤーならびに IBM ビジネス・パートナーは、その予見の有無を問わず発生した以下のものについて賠償責任を負いません。

1. データの喪失、または損傷。
2. 直接損害、特別損害、付随的損害、間接損害、または経済上の結果的損害
3. 逸失した利益、ビジネス上の収益、あるいは節約すべかりし費用

国または地域によっては、法律の強行規定により、上記の責任の制限が適用されない場合があります。

使用条件

これらの資料は、以下の条件に同意していただける場合に限りご使用いただけます。

個人使用: これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、非商業的な個人による使用目的に限り複製することができます。ただし、IBM の明示的な承諾をえずに、これらの資料またはその一部について、二次的著作物を作成したり、配布 (頒布、送信を含む) または表示 (上映を含む) することはできません。

商業的使用: これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、お客様の企業内に限り、複製、配布、および表示することができます。 ただし、IBM の明示的な承諾をえずにこれらの資料の二次的著作物を作成したり、お客様の企業外で資料またはその一部を複製、配布、または表示することはできません。

ここで明示的に許可されているもの以外に、資料や資料内に含まれる情報、データ、ソフトウェア、またはその他の知的所有権に対するいかなる許可、ライセンス、または権利を明示的にも黙示的にも付与するものではありません。

資料の使用が IBM の利益を損なうと判断された場合や、上記の条件が適切に守られていないと判断された場合、IBM はいつでも自らの判断により、ここで与えた許可を撤回できるものとさせていただきます。

お客様がこの情報をダウンロード、輸出、または再輸出する際には、米国のすべての輸出入関連法規を含む、すべての関連法規を遵守するものとします。

IBM は、これらの資料の内容についていかなる保証もしません。これらの資料は、特定物として現存するままの状態で提供され、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任なしで提供されます。

付録. 特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒242-8502
神奈川県大和市下鶴間 1623 番 14 号
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム（本プログラムを含む）との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Corporation
Software Interoperability Coordinator, Department YBWA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、IBM 機械コードのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

表示されている IBM の価格は IBM が小売り価格として提示しているもので、現行価格であり、通知なしに変更されるものです。卸価格は、異なる場合があります。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。サンプル・プログラムは現存するままの状態で提供されるものであり、いかなる保証も提供されません。IBM は、お客様の当該サンプル・プログラムの使用から生ずるいかなる損害に対しても一切の責任を負いません。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生的創作物にも、次のように、著作権表示を入れていただく必要があります。

© (お客様の会社名) (西暦年). このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。 © Copyright IBM Corp. _年を入れる_.

この情報をソフトコピーでご覧になっている場合は、写真やカラーの図表は表示されない場合があります。

プログラミング・インターフェース情報

本書に記述されているプログラミング・インターフェースを使用すると、ユーザーは IBM Toolbox for Java のサービスを利用するためのプログラムを作成できます。

商標

IBM、IBM ロゴおよび ibm.com は、世界の多くの国で登録された International Business Machines Corp. の商標です。他の製品名およびサービス名は、IBM または各社の商標です。現時点での IBM の商標リストについては、「Copyright and trademark information」(www.ibm.com/legal/copytrade.shtml) をご覧ください。

Adobe、Adobe ロゴ、PostScript、PostScript ロゴは、Adobe Systems Incorporated の米国およびその他の国における登録商標または商標です。

Linux は、Linus Torvalds の米国およびその他の国における商標です。

Microsoft、Windows、Windows NT および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

UNIX は The Open Group の米国およびその他の国における登録商標です。

Java およびすべての Java 関連の商標およびロゴは Sun Microsystems, Inc.の米国およびその他の国における商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

使用条件

これらの資料は、以下の条件に同意していただける場合に限りご使用いただけます。

個人使用: これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、非商業的な個人による使用目的に限り複製することができます。ただし、IBM の明示的な承諾をえずに、これらの資料またはその一部について、二次的著作物を作成したり、配布 (頒布、送信を含む) または表示 (上映を含む) することはできません。

商業的使用: これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、お客様の企業内に限り、複製、配布、および表示することができます。 ただし、IBM の明示的な承諾をえずにこれらの資料の二次的著作物を作成したり、お客様の企業外で資料またはその一部を複製、配布、または表示することはできません。

ここで明示的に許可されているもの以外に、資料や資料内に含まれる情報、データ、ソフトウェア、またはその他の知的所有権に対するいかなる許可、ライセンス、または権利を明示的にも黙示的にも付与するものではありません。

資料の使用が IBM の利益を損なうと判断された場合や、上記の条件が適切に守られていないと判断された場合、IBM はいつでも自らの判断により、ここで与えた許可を撤回できるものとさせていただきます。

お客様がこの情報をダウンロード、輸出、または再輸出する際には、米国のすべての輸出入関連法規を含む、すべての関連法規を遵守するものとします。

IBM は、これらの資料の内容についていかなる保証もしません。これらの資料は、特定物として現存するままの状態で提供され、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任なしで提供されます。

Printed in Japan

日本アイ・ビー・エム株式会社
〒103-8510 東京都中央区日本橋箱崎町19-21